BIO/CS 423 Computational Biology
Spring 2024
Course Handouts
Dr. Tammy VanDeGrift

Name: _____

If found, call/email:_____

HANDOUTS – Please bring this booklet to all class sessions

For latest calendar and syllabus: see Moodle (learning.up.edu)

Welcome to BIO/CS 423 😊

Lab Instructions

1. **Software**: All lab software is installed on the windows engineering build, accessible via desktop.up.edu. It is also installed on the computers available in Shiley 206 – our assigned classroom and the open computer lab in Shiley 208.

2. **Prelabs**: Prelabs are to be completed individually and are due at the start of the lab day. Prelabs are designed to ensure you have the background knowledge of concepts from class, the textbook, and other resources.

3**. Lab work sessions**: You are expected to attend the class sessions for lab days. You will work with another student during the lab days to complete lab checkpoints. In some cases, you will work in groups of three. The lab groups will be posted prior to each lab session and may be adjusted due to absences.

4. **Lab communication**: You are expected to work collaboratively on the labs and ask questions of the instructor. The instructor is happy to look at your work as you make progress.

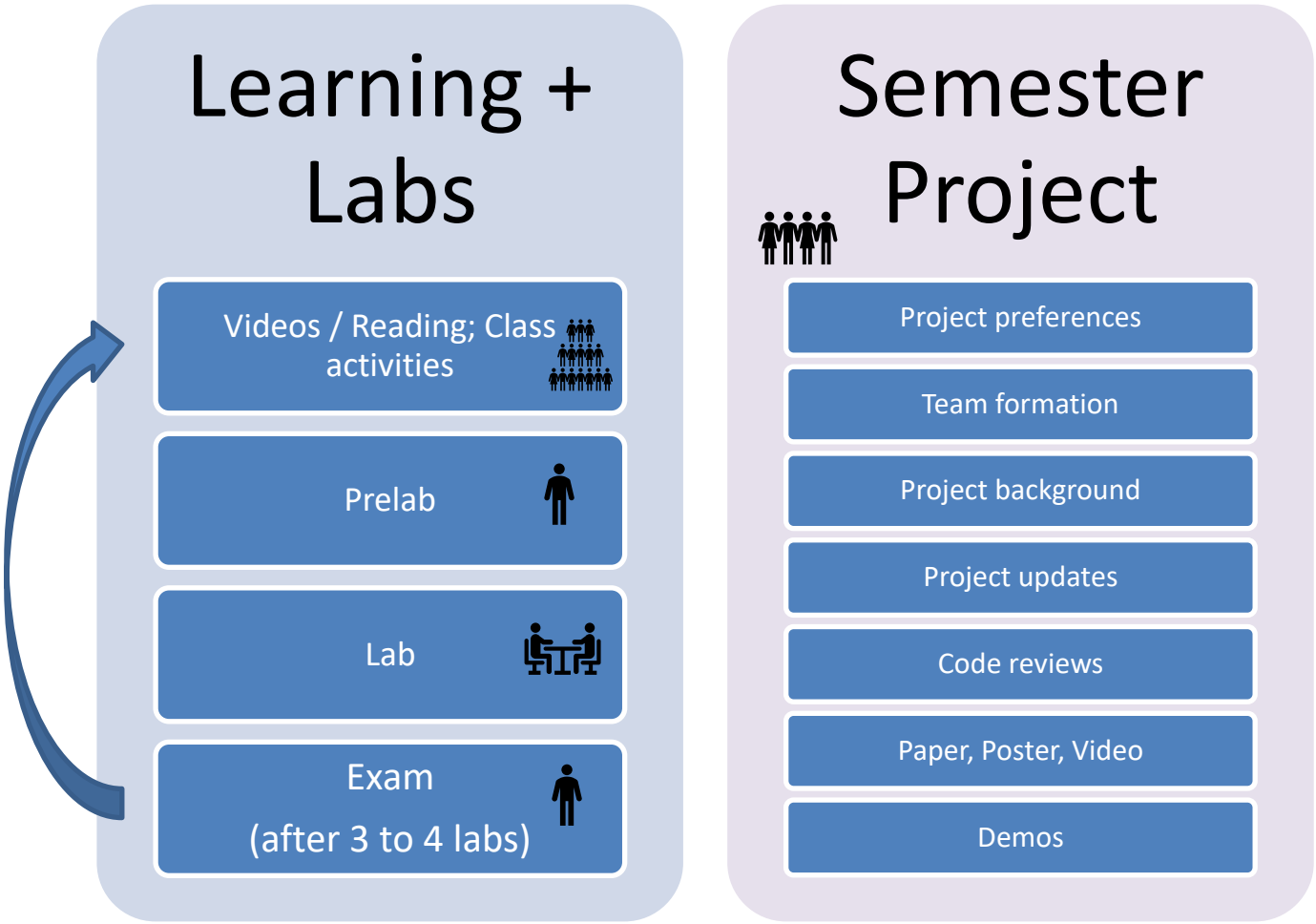
5. **Questions**: If you have a question during the lab, please ask in the instructor.

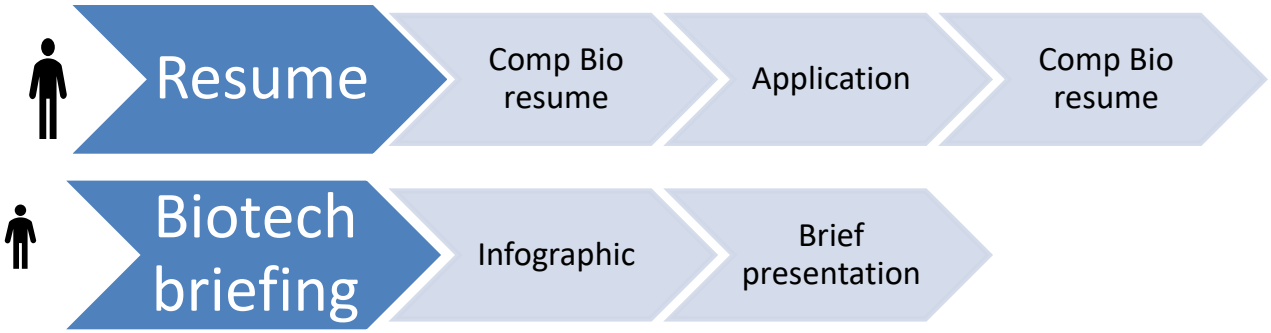
6. **Lab exercises**: You and your partner should download the lab handout and insert your responses in-line with the questions/activities. You and your partner(s) will work through lab exercises together. Some labs may be finished during the class session and some may be longer, which will require you to finish the lab outside class time.

7. **Lab submissions**: Submit work for the lab write-up as pdf file along with any code/test tiles as a .zip file to Moodle by the deadline. The expectation is that pairings will complete the work together outside class time. Check Moodle for lab deadlines.

8. **Late days**: You have two free late days to submit prelabs and/or labs late. You may submit two items up to 24 hours late or submit one item up to 48 hours late. For partnered labs, all members will be "charged" late days for late work. However, if one partner has remaining late days and one partner does not, you may use the maximum late days of both partners.

Course Design for Learning





First look at python

Top line shows path to the python executable, so this can be run as a script in addition to IDLE

```
#!/usr/bin/env python

# demo code for first day of CS 423

sequence = "ATCGATTCG"
firstC = sequence.find("C")
print(firstC)

# look at iteration

count = 0
while count < 10:
    print("I like programming")
    count = count + 1

for c in sequence:
    print(c)

for i in range(10): #iterate 10 times
    print(i)

for i in range(2,10):
    print(i)

for j in range(2,len(sequence)):
    print(sequence[j])

# In general, when processing strings, try to use built-in string functions
# rather than writing loops. String functions are faster.
```

First Lecture Activities

Open up the python documentation in case you need help: https://docs.python.org/3/index.html

There is a brief guide to python on pages 11 to 13 for you to review, too.

Activity 1: With a partner, write a python script called `percentA.py` that defines a DNA sequence (you can choose the DNA sequence bases – it need not match below) and **prints** the overall percentage of adenine (A) nucleotides.

Hint: check out the `len`, `count`, `and str` functions

```
-------------------------------------
sequence = "ATCGATTCG"
numA =



print(                                    )
```

Activity 2: With a partner, write a python script called `third.py` that defines a function called **everyThird** that takes in a string called `sequence` as a parameter. The function should print out every third character in the string, starting with the 0th character. For example, if the input string is "computers", then it should print the letters c, p, and e. Call the function at the bottom of the script to test it.

```
def everyThird(sequence):
```

Activity 3: Below are two versions of functions to convert the content of a FASTA file into a string. A FASTA file is a specific text file format where the first line starts with the ">" character and contains file notes on that first line. The remaining lines of the file contain biological data characters, such as DNA nucleotides. Read through both versions and mark the differences. This function will be part of lab 1. Reading from FASTA files will be a common task in the course.

Do you have any questions about the code?

```
#!/usr/bin/env python

#########################################################
# convertFileToSequence - takes a FASTA file and returns the
# DNA sequence as a string
# opens file and explicity closes file
#########################################################
def convertFileToSequence(filename):
    clean = -1 # used for determining output
    # read in file
    file = open(filename, 'r')
    try:
        # read in first line
        header = file.readline()
        if (header[0] == '>'):
            print("in FASTA format")
            clean = 1
            # read rest of file
            sequence = file.read()
            # remove all return and newline characters
            sequence = sequence.replace("\r", "")
            sequence = sequence.replace("\n", "")
        else:
            print("invalid format")

    finally:
        # close file
        file.close()

        if(clean == -1): return -1  # so other functions can check
        return sequence
```

```
#########################################################
# convertFileToSequence2 - takes a FASTA file and returns the
# DNA sequence as a string
# uses handy file handling code that automatically closes the
# file when leaving with block
#########################################################
def convertFileToSequence2(filename):
    clean = -1
    with open(filename, 'r') as file:
        # read in first line
        header = file.readline()
        if (header[0] == '>'):
            print("in FASTA format")
            clean = 1
            # read rest of file
            sequence = file.read()
            # remove all return and newline characters
            sequence = sequence.replace("\r", "")
            sequence = sequence.replace("\n", "")
        else:
            print("invalid format")
    if(clean == -1): return -1  # so other functions can check
    return sequence
```

Quick python reference: files, randomness, lists, math, keyboard input

You should take some time to review the python documentation here:
Python standard library information: https://docs.python.org/3/library/
All documentation: https://docs.python.org/3/

Files

```
file = open(filename)              ## opens file (not append mode)
file = open(filename, 'a')         ## opens file in append mode
file = open(filename, 'w')         ## opens file in write mode
with open(filename) as file:       ## automatically closes file at end of block
file.read()                        ## reads entire file and returns the string
file.write(value)                  ## writes value to file
file.readline()                    ## reads one line of the file and returns it as a
                                   ## string
file.close()                       ## closes file – be sure to do this when you are
                                   ## finished with it or use the with control structure
```

Note: You may need to put "\r\n" in a string to write to a file to get the newline on Windows

Random Numbers

```
import random                      # to load the methods for generating random numbers
random.random()                    # generates floating point number in [0.0, 1.0)
random.randint(start, stop)        # generates random integer in [start, stop]
random.randrange(int)              # generates random integer in [0, int-1]
random.gauss(mu, sigma)            # generates random number from Gaussian distribution
                                   # with mean mu and standard deviation sigma
random.shuffle(list)               # returns a shuffled version of the list
                                   # many others that can be found in the documentation
```

Lists

This is a collection in python; similar to an array in Java, but with the following differences:

- Items need not be the same type
- Can use + to combine lists

Example:

```
aList = ["hi", 4, 7.5, "bye"]

alist[0]      # returns "hi"
```

Unlike strings, lists are mutable objects in python:

```
aList[2] = 8.5             # now, aList is ["hi", 4, 8.5, "bye"]
```

Operators on Lists

```
+             combines two lists into a single list
*             concatenates list number of times given by second argument
in            just like with strings, returns True if item is in list
                    "hi" in aList  returns True
```

```
not in       returns opposite of in
len(list)    returns size of list
[start:end]  returns sublist starting at position start and up to but not including
                    end
```

Methods on Lists:

```
list.append(item)          # adds item to end of list
list.insert(i, item)       # puts item in ith position and shifts others
list.pop()                 # removes and returns last item
list.pop(i)                # removes and returns ith item
list.reverse()             # reverses list
list.index(item)           # returns index of first occurrence of item
list.count(item)           # returns number of times item appears in list
list.remove(item)          # removes first occurrence of item
list.sort()                # sorts list (as long as items in list are comparable)
max(list)                  # returns maximum element in list
min(list)                  # returns minimum element in list
```

Input from keyboard:

```
userInput = input("Enter your name: ")  # will prompt user and store what they type
                                        # into userInput as a string
```

Math

There is a math module (see documentation for all math functions)
Need to import module if you want to use these functions:

```
import math
```

Similar to what you find on a calculator:

```
math.cos(val)
math.sin(val)
math.sqrt(val)
math.fabs(val)      # absolute value
math.pi             # constant
math.e              # constant
math.floor(val)
math.ceil(val)
```

…. many more

Program Organization, Compilation, and Execution

By the way, python can be written in object-oriented style, but we will mostly use the procedural approach for now. But, as programs get bigger and more complex, it might make sense to write OO code.

You can define your functions in a python file and execute functions below those (so define at the top and use at the bottom). You could also define your functions in one file and import that file in a different file that calls those functions.

You can run your python program outside the interpreter with a Linux shell command (that's what the first line helps you do). In a shell, just type the name of the python code:

```
$ lab1.py
$./lab1.py                if . is not in your path
```

You can also run your python program from a Windows shell like this:

```
$ python lab1.py
```

Frequently Asked Questions

1. What is the .py file?

    This is where you define your python code.

2. What is the .pyc file?

    When you first import a module, python may create a .pyc file that is compiled python byte code to help speed up execution. If the .pyc has the same timestamp as the .py file, the .pyc is used. If they are different, the .py file is reread/recompiled. You may not see these get created.

3. Why is the top line #!/usr/bin/env python?

    If you run your python scripts outside IDLE or another IDE and run it from a shell, this tells the shell environment where to find the python executable.

Molecular Biology

See Moodle resources – guest lecture by Dr. Susan Murray, Biology

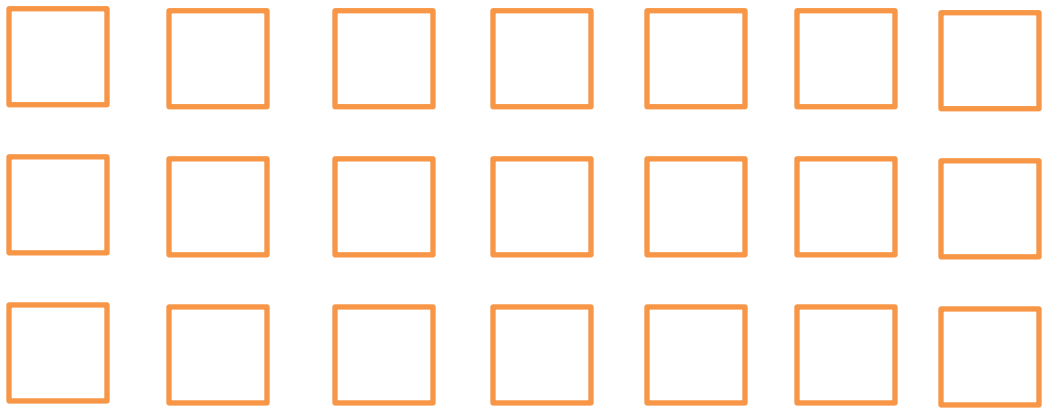
Space for Notes:

Intro to Algorithms: Single Pile Nim

Two-player game

Rule: On the player's turn, he/she can cross out 1, 2, or 3 boxes.

Winner: The player who crosses out the final box.



1. Choose a partner and play the game, once in each other's coursepack. Give each partner the chance to start the game.

2. What is the winning "strategy"?

Two Pile Nim

Two-player game

Rule: On the player's turn, he/she can cross out any number of boxes from 1 to the number that remain open in a **single** pile.

Winner: The person who crosses out the final box.

To play: Draw as many boxes as you would like to start in each pile. Then play the game.

1. With your partner and play the game, once in each other's coursepack. Give each partner the chance to start the game.

2. What is the winning "strategy"?

```
#!/usr/bin/env python

#Code to demonstrate running time of two different implementations
#Lecture 6
#CS 423

#Activity: Assume sequence has N nucleotides. Assume there are P instances
#of substring TATAAA in sequence.
#How many nucleotides are examined or copied in each function below?
#Give your answer in terms of N and P.

# idea: keep searching from previously found substring
def TATAPosition(sequence):
    foundPos = sequence.find("TATAAA", 0)  #start searching from position 0
    while(foundPos >= 0):                  #stop once find returns -1
        print(foundPos)
        foundPos = sequence.find("TATAAA", foundPos + 1)
            # start search from string index (foundPos + 1)

# idea: every time the substring is found, remove the first part of the sequence
def TATAPosition2(sequence):
    offset = 0
    position = sequence.find("TATAAA")  #find first occurrence of TATAAA
    while(position >= 0):               #keep going if found an occurrence
        print(position + offset)
        offset = position + offset + 1
        sequence = sequence[position + 1:len(sequence)]
        position = sequence.find("TATAAA")

# code to execute
sequence = "AAAATATAAAGGGCCTATAAATATAAAAAATATAATATAAATATAAACCCCCTATAAA"
#sequence = "GGCCTTAAGG"
#sequence = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATATAAAGGGG"
TATAPosition(sequence)
TATAPosition2(sequence)
```

Code Runtime Analysis (see prior page)

Activity: Assume sequence has N nucleotides. Assume there are P instances of substring TATAAA in sequence.

1. How many nucleotides are examined or copied in `TATAPosition`? Give your answer in terms of N and P.

2. How many nucleotides are examined or copied in `TATAPosition2`? Give your answer in terms of N and P.

Motifs and Profiles

Table 1: Positions 3 – 9 of CRP site in E. coli

```
TTGTGGC
TTTTGAT
AAGTGTC
ATTTGCA
CTGTGAG
ATGCAAA
GTGTTAA
ATTTGAA
TTGTGAT
ATTTATT
ACGTGAT
ATGTGAG
TTGTGAG
CTGTAAC
CTGTGAA
TTGTGAC
GCCTGAC
TTGTGAT
TTGTGAT
GTGTGAA
CTGTGAC
ATGCGAC
TTGTGAG
```

Do you see a pattern?

Table 2: Profile of CRP site

	3	4	5	6	7	8	9
A	.35	.043	0	.043	.13	.83	.26
C	.17	.087	.043	.043	0	.043	.3
G	.13	0	.78	0	.83	.043	.17
T	.35	.87	.17	.91	.043	.087	.26

Table 3: LLR weight matrix of CRP site

	3	4	5	6	7	8	9
A	.48	-2.5	-inf	-2.5	-.94	1.7	.061
C	-.52	-1.5	-2.5	-2.5	-inf	-2.5	.28
G	-.94	-inf	1.6	-inf	1.7	-2.5	-.52
T	.48	1.8	-.52	1.9	-2.5	-1.5	.061

Practice – motif for start codon:

Hypothetical translation start sites (In general "ATG" is the start codon for a gene, but it turns out that it could be "GTG" or "TTG"). Here's an example **profile** for the start codon. Answer the questions below.

	1	2	3
A	.625	0	0
C	0	0	0
G	.25	0	1
T	.125	1	0

a. P(ATG is a start codon) =

b. P(GTG is a start codon) =

c. P(CTG is a start codon) =

d. Likelihood Ratio(ATG is a start codon given uniform background distribution of nucleotides) =

e. Likelihood Ratio(GTG is a start codon given uniform distribution) =

f. Likelihood Ratio(CTG is a start codon given uniform distribution) =

g. Complete the weight matrix of LLRs below for the start codon site [you will likely need a calculator]. Assume a uniform background distribution for nucleotides.

	1	2	3
A			
C			
G			
T			

h. LLR(ATG) =

i. LLR(GTG) =

j. LLR(CTG) =

Practice:

Create your own motif profile of any length (3 to 6 nucleotides) and determine the probabilities, likelihood ratios, and log likelihood ratios.

Mutations

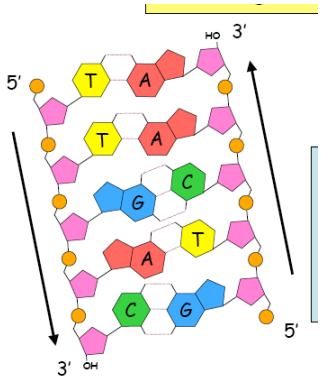


Figure 1: DNA nucleotide bonds

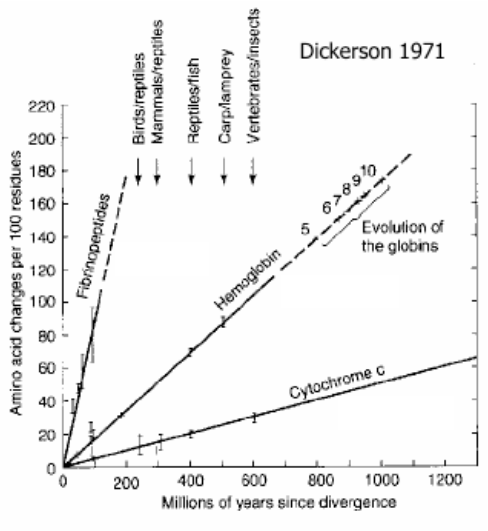


Figure 2: Amino acid substitutions over time for three proteins

Distance Metrics

Hamming distance = **H** = # of differences between the two strings

Proportion distance = **p distance** = $\frac{\text{H = \# differences}}{\text{length of string}}$

What is the H distance for the top two strings in Table 1? _____

What is the p distance for the top two strings in Table 1? _____

Table 1: Sequence Evolution (bold are changes, underscore are final changes)

ACCATGGAATTTTATACCCT
A**G**CATGG**T**ATTTTATAC**G**CT
AGCAT**C**G**A**A**A**TTTATACG**A**T
A**T**CAT**G**GAAATT**A**ATAC**C**AT
A**T**CATG**C**AA**TA**T**A**ATACCA**G**

Jukes Cantor Correction

Table 2: Substitution rates for DNA nucleotides

	A	C	G	T
A	(1-3a)	a	a	a
C	a	(1-3a)	a	a
G	a	a	(1-3a)	a
T	a	a	a	(1-3a)

For any nucleotide, the probability of a substitution at a given time is 3a (add up the probabilities in a single column). Over a period of time t, the number of substitutions is 3at.

Since we are looking at a model such that the sequences diverge from a common sequence, there is a rate of 3at on each branch. Thus, the actual rate of substitutions (K) that take place over time t is 3at + 3at (one for each branch). K = 6at.

But, we don't get to observe all the substitutions, since some may change and change back to the original. We need an estimate for the "actual" rate of substitutions, but we don't know the value for 'a' and we don't know the value for 't'. Hmmm…

What we want:
K = actual rate of substitutions = (correction factor) * (p distance)

Derivation of Jukes Cantor Correction Factor:

Probability that nucleotide/residue will be the same at time (t+1) given time (t):
= (Probability that it is residue r)(Probability that it was residue r at time t) +
(Probability that it changes to r)(Probability that it was a different nucleotide at time t)

$P_{same(t+1)} = (1-3a)P_{same(t)} + a(1-P_{same(t)})$

Solving with differential equations, we get:

$P_{ii(t)}$ = Prob(nucleotide stays the same at time t) = $¼ + ¾(e^{-4at})$
$P_{ij(t)}$ = Prob(nucleotide goes from i to j at time t) = $¼ - ¼(e^{-4at})$

Now, we can compute the probability that two sequences differ at a given position:

p = 1 – (probability they are identical)
p = 1 – (prob of both staying the same + prob of both changing to same base)

$$p = 1 - [(P_{AA(t)})(P_{AA(t)}) + (P_{AT(t)})(P_{AT(t)}) + (P_{AC(t)})(P_{AC(t)}) + (P_{AG(t)})(P_{AG(t)})]$$

Substituting the values from above, we get:

$$p = 1 - [(¼ + ¾(e^{-4at}))(¼ + ¾(e^{-4at})) + 3[(¼ - ¼(e^{-4at}))(¼ - ¼(e^{-4at}))]]$$
$$p = 1 - [[1/16+ 3/16(e^{-4at}) + 3/16(e^{-4at})] + 9/16(e^{-8at})] + 3[(1/16 – 1/16(e^{-4at}) – 1/16(e^{-4at}) + 1/16(e^{-8at})]$$
$$p = 1 - [(1/16 + 6/16(e^{-4at}) + 9/16(e^{-8at})] + 3[(1/16 – 2/16(e^{-4at}) + 1/16(e^{-8at})]$$
$$p = 1 - [(1/16 + 3/16) + (6/16 -6/16)(e^{-4at}) + (9/16 + 3/16(e^{-8at}))]$$
$$p = 1 - [1/4 + 12/16(e^{-8at})]$$
$$p = 1 - [¼ + ¾(e^{-8at})]$$
$$p = ¾ - ¾(e^{-8at})$$
$$p = ¾ (1 - e^{-8at})$$

Now, rearranging to isolate the term 8at:

$$(4/3)p = (1- e^{-8at})$$
$$e^{-8at} = 1 – (4/3)p$$
$$-8at = ln(1-(4/3)p)$$
$$8at = -ln(1-(4/3)p)$$

Since, we know the number of substitutions from a single source is K = 6at, we can determine K by taking ¾ of the value for 8at:

$$K = -3/4\ ln(1 – (4/3)\ p)$$

This is the Jukes Cantor correction for DNA mutations.

<u>Example</u>
If we find that that the proportion distance p is: .2 for two DNA sequences, then
K = -3/4 ln(1 – (4/3)*.2)

K = .2326 # so this gives us an estimate of the actual rate of change
# not just the observed substitution rate
# makes sense, since there are “hidden” changes, so K should be bigger than .2

<u>What about amino acid changes?</u>
These are more complicated since the probability of each change is not 0.25 like it is in DNA. If we extend the math to amino acids, the correction would be K = (-19/20) ln (1 – (20/19)p). However, amino acid substitutions are usually modeled from actual data. The PAM and BLOSUM matrices showcase scores of amino acid substitutions.

Practice With Distance Calculations

In small groups, complete the following questions:

Calculating sequence differences

Assume these are two DNA sequences that evolved from a common ancestral sequence:

```
AATTGCTAGACTACTTAGGG
ATATGCGAGACTTGTTCCGG
```

1. What is the Hamming distance? H = _____

2. What is the p distance? p = _____

3. What is the Jukes Cantor estimate? K = _____

$K = -3/4 \ln(1 – (4/3) p)$

4. Given that the sequence has 20 nucleotides, the estimate K is for the rate of change at a single position. Using K, how many nucleotide substitutions are estimated? _____

Intro to Sequence Alignment

A. Why would we want to align DNA sequences to see if they are similar?

Example: multiple sclerosis
Hypothesis: T cells (normally identify foreign agents in the body) mistakenly identify the proteins in the nerve's myelin sheaths as foreign

B. How would you test this hypothesis?

C. Consider the following DNA sequences:
ATATATATA
TATATATAT

What is the Hamming distance for these sequences? _____

D. But, how really different are these sequences?

E. Assume you have the following operations: delete one nucleotide, insert one nucleotide, and substitute one nucleotide for another (D, I, S). How many operations does it take to make the sequences in part C have a Hamming distance of 0? _____

F. How many operations (D, I, S) does it take to get from the following top sequence to the bottom sequence?
TGCATAT
ATCCGAT

_____ What are those operations?

G. Given two strings (sequences of characters from the same alphabet), what is the optimal alignment of those sequences?

Example:
s1 = ACGCTG
s2 = CATGT

How would you align these to have the most matches?

Example 1 alignment:

```
A     C     -     -     G     C     T     G
-     C     A     T     G     -     T     -
```

Example 2 alignment:

```
A     C     G     C     T     G     -
-     C     -     A     T     G     T
```

When aligning, the widest this will be is N + M where N and M are the lengths of the strings.

How wide is alignment 1? _____

How wide is alignment 2? _____

Qualitatively, which do you think is a better alignment? _____

Vocabulary:
Match: column with same letter
Mismatch: column with differing letters
- Insertion: column with space on top
- Deletion: column with space on bottom
- Indel: insertion or deletion

In immediate above example, the C/A column is a mismatch, the C/C column is a match, the A/- column is a deletion, the -/T column is an insertion.

In the alignments above, circle all mismatches. Put rectangle around all insertions. Put squiggles around all deletions.

H. Also need a way to "score" the alignment, so we can determine the optimal alignment. For simplicity, let's assume every match is worth 2 points and every mismatch is worth -1. Then, in the example in part G, the score is -1 + 2 + -1 + -1 + 2 + -1 + 2 + -1 = 1.

Let d(x,y) be the score of aligning symbols x and y.

The length of a string S, denoted by |S| is the number of symbols in S. We'll denote the symbols of S as S[0], S[1], S[2]…etc.

An alignment A between strings S and T maps them into strings S' and T' where S' and T' may contain space characters, where:

|S'| = |T'|

The removal of spaces in S' gives the string S and the removal of spaces in T' gives the string T

The alignment score of A, aligning strings S and T is:

$$\sum_{i=0}^{n-1} d(S'[i], T'[i])$$

add together the distance for each character in the alignment; n is the total width of the aligned sequences

I. String alignment problem (formalized):
Given two strings S and T (sequences of characters from the same alphabet), what is the optimal (maximum scoring) alignment of S and T?

Can you think of an obvious algorithm to solve the string alignment problem?

How long does your algorithm take (assume sequence S has N characters and assume sequence T has N characters?

String Alignment and Intro to Dynamic Programming

String Alignment Overview

Given two strings S and T (sequences of characters from the same alphabet), what is the optimal (maximum scoring) alignment of S and T?

A. Can you think of an obvious algorithm for the string alignment problem (from last lecture)?

Idea: look at all possible "aligned" subsequences of each string where the other parts match up with dashes on the other string

|S| = n

```
For all i, 0 <= i <= n do
       For all subsequences A of S with |A| = i do
              For all subsequences B of T with |B| = i do
                     Form an alignment that matches A[k] with B[k] 0 <= k < i
                            And matches all other characters with dashes
                     Determine the score of this alignment
                     Keep maximum alignment
Return maximum alignment
```

B. But, what is the running time of this algorithm?

Outer loop: runs n times

For all subsequences A of S: (n choose i) subsequences with length i

For all subsequences B of T: (n choose i) subsequences with length i

Forming alignment: (2n – i) characters long [since there are n characters in S and (n – i) unmatched characters with T]

So, we have a running time of:

$\sum_{i=0}^{n}\binom{n}{i}\ \binom{n}{i} * (2n - i)\ >$

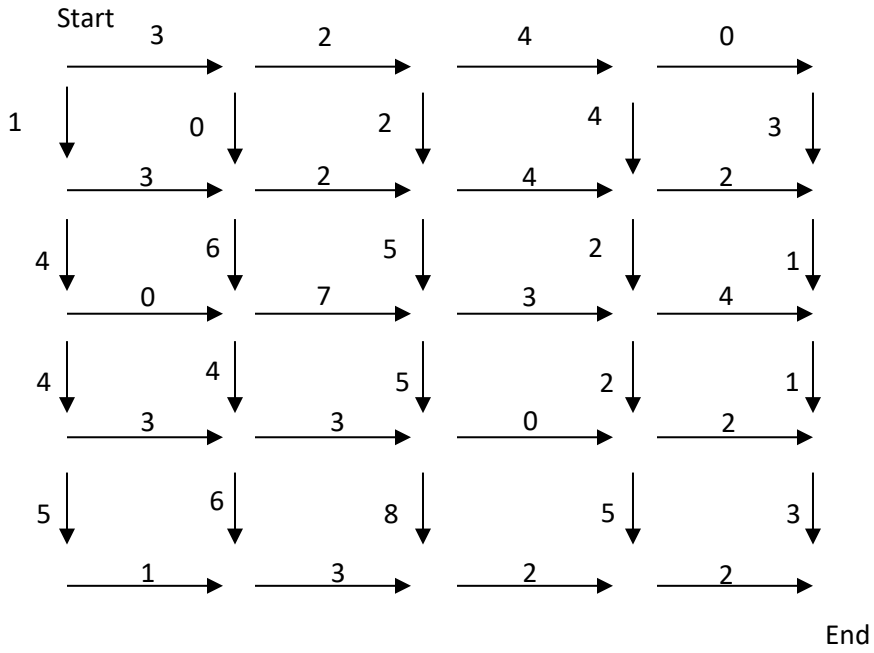
$n * \sum_{i=0}^{n}\binom{n}{i}\ \binom{n}{i}\ = n * \binom{2n}{n} > 2^{2n}$ for n > 3 // Stirling's approximation

This running time is not feasible in practice, so we need to do better. We will use a computing technique called dynamic programming.

Manhattan Problem (Example of Dynamic Programming)

Manhattan Problem: Find the highest weight path in a weighted grid (graph) from a start node to an end node.

Graph is like blocks in the city – can travel east and south. Each edge on the block has a weight (cost).



C. Can you find the path with the highest weight?

Let's label each vertex as [i][j] where Start is at position [0][0] and the one to the right of start is position [0][1]. The max total cost from the start to vertex at [i][j] is stored as v[i][j].

We'll create a table that contains the highest scoring path to each of the vertices:

Table of Costs from Start to each Vertex (First row completed):

0	3	5	9	9

The top row is simply the cost of the path along the top line.

Table of Costs from Start to each Vertex (Left column and first row completed):

0	3	5	9	9
1				
5				
9				
14				

The left column is simply the cost of the path along the left line of the graph.

Now, to fill in the rest of the table:

v[i][j] = max { v[i-1][j] + weight of edge from v[i-1][j] to v[i][j]
v[i][j-1] + weight of edge from v[i][j-1] to v[i][j]

D. Final Table (complete the entries above in the table)

Manhattan Algorithm:

Input: G is a graph n nodes wide and m nodes tall with weights along edges with start node at [0,0] and end node at [n-1,m-1]

```
v[0][0] = 0
for j = 1 to (n-1): // fill in top row
       v[0][j] = v[0][j-1] + weight([0][j-1] to [0][j])
for i = 1 to (m-1): // fill in left column
       v[i][0] = v[i-1][0] + weight([i-1][0] to [i][0])
for i = 1 to (m-1): // fill in middle cells
       for j = 1 to (n-1):
              v[i][j] =    max { v[i-1][j] + weight([i-1][j] to [i][j])
                               { v[i][j-1] + weight([i][j-1] to [i][j])

return v[n-1][m-1]  // highest scoring path
```

E. What is the running time of the Manhattan algorithm assuming the graph is n nodes wide and m nodes tall?

O( )

Global String Alignment

Problem: Given two strings, S and T, what is the alignment with the maximum score and what is the score?

Example Input: What alignment is the optimal alignment for the following DNA sequences:

> DNA Sequence 1
ACTGCGATTGACGTACGATCATCGTACGATCATCATGCTGAGCTATCATCATCGTACTGA
TCGTAGACTACGTAGCTAGCATGCAGTCTGATGACGTCATGCTGACGTAGCATGC

> DNA Sequence 2
GACTAGCAGCGAGAGATCTCTCGAGTATGCGAGAGCTGATGCATCTACGTATGCAGTCGT
GCTAATGCGAGCGTATACGCGGGCATGTAGAGACTTCCTAGTAC

Shorter Example: Suppose we want to align:
AGCGTTA
ACGTGA

Table 1: Dynamic programming table for global string alignment

		A	G	C	G	T	T	A
A								
C								
G					X			
T								
G			y					
A								

x represents the best score for aligning AGCG and ACG
y represents the best score for aligning AG and ACGTG

What does each table entry mean?

The north cell represents: aligning the left sequence's last character with a gap
The west cell represents: aligning the top sequence's last character with a gap
The northwest cell represents: aligning the last character of each sequence together

Let c[I,j] be the cell entry in the ith row and jth column.
g = gap penalty (insertion or deletion)
m = match/mismatch score

Global Alignment Algorithm:

```
c[0][0] = 0                 // no alignment
for i from 1 to p           // fill in gap penalties in left column
       c[i][0] = g*i
for j from 1 to q           // fill in gap penalties in top row
       c[0][j] = g*j
for i from 1 to p           // fill in table left to right, top to bottom
       for j from 1 to q
              c[i][j] = max(c[i][j-1] + g, c[i-1][j] + g, c[i-1][j-1] + m)

return c[p][q]
```

Example:

Let's assume these costs:

g = -6, match = 5, mismatch = -4

We want to align:

AGCGTTA

ACGTGA

Complete the table (left column done for you):

		A	G	C	G	T	T	A
	0							
A	-6							
C	-12							
G	-18							
T	-24							
G	-30							
A	-36							

So, the optimal score is 15 for these two sequences. Now, how do we actually get the alignment? We figure out from which direction the maximum value came from in the table.

15 came from 10 + 5 at the end, so:

```
A
|
A
```

Then, 10 came from 14 – 4: (diagonal), so:

```
TA
||
GA
```

Then, 14 came from 9 + 5 (diagonal), so:

```
TTA
```

```
|||
TGA
```

Then, 9 came from 4 + 5 (diagonal), so:

```
GTTA
||||
GTGA
```

Then, 4 came from -1 + 5 (diagonally), so:

```
CGTTA
|||||
CGTGA
```

Then, -1 came from 5 – 6 (left), so:

```
GCGTTA
||||||
-CGTGA
```

Then, 5 came from 0 + 5 (diagonal), so:

```
AGCGTTA
A-CGTGA
```

There, we have it! So, not only should we keep track of the table values, we should keep another table that holds from which direction the maximum value came from.

What is the running time for this global alignment algorithm, if S has length N and T has length M?

O( )

Now, you try aligning these with scoring system: g = -6, match = 5, mismatch = -4

		T	A	T	G	C	T
	0						
T							
G							
A							
C							
A							
G							
T							

What is the optimal alignment score?

What about aligning these two sequences?

```
---TGGTAGATTC-C--CACGAGATCTACCGAG-TATGAGTAGGGGGAC-GTTCGCT-C-GG
|  ||  |  |  | |||||||||  ||||||  ||||||||  ||  |  ||  |  |  ||
GCCT-CTA-ACACACTGCACGAGATCAACCGAGATATGAGTA---ATACAG—CGGTACGGG
```

The score for this alignment is 60. But, what if we were really looking for the best substring of the one sequence aligned with the best substring of the second sequence?

Global alignment: align complete strings
Local alignment: align best matching substrings

Best local alignment of sequences above has a score of 105:

```
CACGAGATCTACCGAG-TATGAGTA
|||||||||  ||||||  ||||||||
CACGAGATCAACCGAGATATGAGTA
```

Local String Alignment

Given two strings, S and T, what is the maximum alignment among any substrings S' of S and any substrings T' of T?

c – table storing alignment costs
d – table storing alignment directions (F, L, T, D)

Algorithm:

```
c[0][0] = 0          // no alignment
d[0][0] = "F"        // start location of alignment
maxValue = 0         // max value of alignment found
maxRowPos = 0        // to keep track of the location of the maximum alignment
maxColPos = 0
for i from 1 to p          // fill in gap penalties in left column
       c[i][0] = 0
       d[i][0] = "F"
for j from 1 to q          // fill in gap penalties in top row
       c[0][j] = 0
       d[0][j] = "F"
for i from 1 to p          // fill in table left to right, top to bottom
       for j from 1 to q
              c[i][j] = max(0, c[i][j-1] + g, c[i-1][j] + g, c[i-1][j-1] + m)
              //add 0 for choice to start new alignment
              d[i][j] = "F", "L", "T", or "D" – wherever max value came from
              // "F" is if max is 0
              if(c[i][j] > maxValue)
                     maxValue = c[i][j]
                     maxRowPos = i
                     maxColPos = j
return c[maxRowPos][maxColPos]
```

Note: The only major differences between global and local alignment is that we can restart the optimal alignment at any time (hence, the 0 as part of the max). Also, the optimal alignment may end at any position in the table, so we find the maximum value of all table entries as the maximum local alignment score. Recovering the alignment follows the same process as global alignment.

Use the scores: g = -6 match = 5 mismatch = -4

Table 1: Complete the cost table below following the local alignment algorithm

		A	G	A	T	C	A	C
	0	0	0	0	0	0	0	0
C	0							
G	0							
A	0							
C	0							
A	0							
G	0							

Table 2: Complete the direction table below

		A	G	A	T	C	A	C
	F							
C								
G								
A								
C								
A								
G								

What is the optimal alignment and cost?

Longest Common Subsequence

Given two strings S and T, what is the longest common subsequence between the two strings?

(In biology, this is important if we want to find the longest sequence of DNA that is exactly the same and in the same order in two strings.)

Assume S = S[1]S[2]S[3]…S[q] //note: in python strings start with index 0
Assume T = T[1]T[2]T[3]…T[p]

Algorithm

```
c[0][0] = 0          // no alignment
d[0][0] = "F"        // start location of LCS
for i from 1 to p    // fill in left column
       c[i][0] = 0
       d[i][0] = "F"
for j from 1 to q    // fill in right column
       c[0][j] = 0
       d[0][j] = "F"
for i from 1 to p    // fill in table left to right, top to bottom
       for j from 1 to q
              if T[i] == S[j] // have a match
                     c[i][j] = max(c[i][j-1], c[i-1][j], c[i-1][j-1] + 1)
              else
                     c[i][j] = max(c[i][j-1], c[i-1][j])
              d[i][j] = "L", "T", or "D" – wherever max value came from
return c[p][q]
```

Note: this is the global alignment algorithm except we do not penalize gaps and we add one for a match.

Table 3: Complete the cost table for LCS

		A	G	A	T	C	A	C
	0	0	0	0	0	0	0	0
C	0							
G	0							
A	0							
C	0							
A	0							
G	0							

Table 4: Complete the direction table

		A	G	A	T	C	A	C
	F	F	F	F	F	F	F	F
C	F							
G	F							
A	F							
C	F							
A	F							
G	F							

What is the longest common subsequence and its length?

Global alignment with affine gaps

In biology, a long gap in a string alignment might be due to a single copying error. Instead of penalizing each gap with score g, the affine gap model has a gap opening score and a gap extension score.

```
ATC--GA
ATCCCGA
```

For the alignment above in the regular model, each gap has an alignment score of -6, but the CC may have resulted from a single insertion or a single deletion.

Suppose the gap opening (GO) score is -7 and the gap extension (GE) score is -2. With the affine gap model, a gap with length 3 has an alignment score of (-7 + -2 + -2 + -2) = -13.

Input: strings S and T, where |S| = n, |T| = m

Algorithm Idea:
Keep track of four tables (S across top, T down left side):

c is the best alignment cost (like our cost table earlier)
e is the best alignment cost where the best score is matching char in S with "-"
f is the best alignment cost where the best score is matching "-" with char in T
g is the best alignment cost where the best score matches char in S with char in T

Note: we need the extra tables to extend the gap score correctly.

Global Alignment Algorithm (affine gap penalty)
Initialize tables:

```
c[0][0] = 0
c[i][0] = GO + GE*i        // fill left column, i from 1 to m
c[0][j] = GO + GE*j        // fill top row, j from 1 to n
e[i][0] = -inf             // fill left column, i from 1 to m
f[0][j] = -inf             // fill top row, j from 1 to n
```

Fill in tables, i from 1 to m, j from 1 to n:

```
g[i][j] = c[i-1][j-1] + m                          //  score for last char in S aligned with last char in T
e[i][j] = max(e[i][j-1] + GE, c[i][j-1] + GO + GE) // max of new gap or extending gap with char in S
f[i][j] = max(f[i-1][j] + GE, c[i-1][j] + GO + GE) // max of new gap or extending gap with char in T
c[i][j] = max(g[i][j], e[i][j], f[I][j])           // best cost for alignment
```

Note: reconstructing the alignment -- know from which table max came from.
Note: can use this model with local alignment (just need to look for max value in table)

GO = -7, GE = -2, match = 5, mismatch = -4

c table

		A	G	A	T	T	T	C
	0	-9	-11	-13	-15	-17	-19	-21
C	-9 (top)	-4	-13	-15	-17	-19	-21	-14
C	-11 (top)	-13	-8	-17	-19	-21	-23	-16
A	-13	-6 (g)	-15	-3	-12	-14	-16	-18
G	-15	-15	-1 (g)	-10	-7	-14	-16	-18
A	-17	-10	-10	4 (g)	-5 (e) *	-7 (e)	-9 (e)	-11
C	-19	-19	-12	-5	0	-9	-11	-4 (g)

* comes from max(-5, -16, -14)

e table

		A	G	A	T	T	T	C
	-inf							
C	-inf	-18	-13	-15	-17	-19	-21	-23
C	-inf	-20	-22	-17	-19	-21	-23	-25
A	-inf	-22	-15	-17	-12	-14	-16	-18
G	-inf	-24	-24	-10	-12	-14	-16	-18
A	-inf	-26	-19	-19	-5 *	-7	-9	-11
C	-inf	-28	-28	-21	-14	-9	-11	-13

* comes from max(-19 + (-2), 4 + (-7) + (-2)) = -5

f table

		A	G	A	T	T	T	C
	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
C		-18	-20	-22	-24	-26	-28	-30
C		-13	-22	-24	-26	-28	-30	-23
A		-15	-17	-26	-28	-30	-32	-25
G		-15	-19	-12	-21	-23	-25	-27
A		-17	-10	-14	-16 *	-23	-25	-27
C		-19	-12	-5	-14	-16	-18	-20

* comes from max(-21 + (-2), -7 + (-7) + (-2)) = -16

g table

		A	G	A	T	T	T	C
C		-4	-13	-15	-17	-19	-21	-14
C		-13	-8	-17	-19	-21	-23	-16
A		-6	-17	-3	-21	-23	-25	-27
G		-17	-1	-19	-7	-16	-18	-20
A		-10	-19	4	-14 *	-11	-18	-20
C		-21	-14	-14	0	-9	-11	-4

* comes from -10 + (-4) [cost in NW cell in c table + cost of mismatch)

```
--AGATTTC
CCAGA---C
Matches: 20 GapTop: -11 GapBottom: -13
Total score: -4
```

BLAST (Basic Local Alignment Search Tool)

1. Suppose you have just discovered a new gene, whose associated protein is 1000 amino acids in length. You are curious to see if this protein sequence is similar to proteins in other organisms. The database includes 1000 organisms with average genome length of 1M (1,000,000) amino acids.

How long will it take to do the local alignment algorithm that we saw for aligning two DNA sequences? (Note that the local alignment algorithm could be extended to align protein sequences, with 20 characters instead of 4 DNA nucleotides.)

Hmmm, this seems like it would take too long. BLAST to the rescue. BLAST does local alignment heuristically (faster, but not necessarily optimal).

Here is the URL for BLAST: http://blast.ncbi.nlm.nih.gov/Blast.cgi

2. BLAST includes nucleotide matching, protein matching, and translated nucleotide matching. We will focus first on protein blast (search protein database using a protein query).

Here's how BLASTp works with protein sequences:

1. Database sequences are pre-tagged and scored
2. Your query string is broken into words of length 6 (by default, can set word length to 2 or 3 in query in algorithm parameters)
3. Similar words to query words meeting a certain threshold are found in the database sequences
4. Extensions to pairs of matches are found (high-scoring segment pairs HSPs)
5. The local alignment between high scoring matches is reported.

Example of BLASTp:
Your query protein: LWGFAHAYHESKWAAHNQEILTPLV

Breaking query string into words
The query string is split into words of length 3:
LWG
WGF
GFA
…..

Scores of words

Each of these words (note: there are 20*20*20 distinct words) has a score with other words. Similar words with scores greater than some threshold T (default is 11) are retained for querying. BLOSUM62 matrix is used for alignment scores by default. Look at the row/column of the aligned amino acids in the words to get score. Each number in BLOSUM62 is added together for the total score.

Assume the query sequence has word **LWG**. Calculate similar word scores to **LWG** below.

a. What is the score of the word **LWG** using BLOSUM62? _____
b. What is the score of the word **IWG** using BLOSUM62? _____
c. What is the score of the word **MWG** using BLOSUM62? _____
d. What is the score of the word **VWG** using BLOSUM62? _____
e. What is the score of the word **LYG** using BLOSUM62? _____
f. What is the score of the word **FWS** using BLOSUM62? _____

Finding similar words

Words scoring >= threshold (default 11) are located in the database (note: database is indexed for quick retrieval of locations of each word).

If two word pairs from the query string are found in the database within a certain distance A (A = 40 for protein blasts) from each other, then the pair is determined a "hit".

Hit extensions and alignment

The hits are then extended in each direction (left and right), aligning it with the query sequence until the alignment score starts decreasing (current version blast also does gaps with penalties, opening penalty of 11 and extension penalty of 1 by default).

Sequences with high alignment scores are reported along with the alignment.

3. Try it: Run BLASTp with an input sequence (example yeast protein YBL105C on Moodle).

a. Look at the results. What is the best scoring sequence? _____

b. Look at the match with organism Zygosaccharomyces bailii CLIB 213. What do you think the + represents between F and Y in the first two aligned AAs?

_____

c. In the same match with Zygosaccharomyces bailii CLIB 213, how many of the AAs are identical? _____ How many are positive? _____ How many are gaps? _____

4. Now that you have seen how to run a BLASTp query, let's discuss the results. Remember, this algorithm is a heuristic – it finds high-scoring alignments, but sacrifices optimality for speed. So, how do you know if the similarity score is meaningful? What would you determine as a "high-scoring" match in BLAST?

BLAST Statistics

E value – expected number of alignments with optimal local alignment score of S or higher in a random database (basically gives us the number of **expected** false positives)

Lower E values are better (means the sequence that BLAST found is less likely to have occurred by chance).

$$E = Kmne^{(-LS)}$$

m = length of query string
n = size of database
K = constant based on extreme value distribution for alignment scores
L = constant based on extreme value distribution for alignment scores
S = alignment score

If S is bigger, what happens to E? _____
If m is bigger (query is longer), what happens to E? _____
If n is bigger (database is bigger), what happens to E? _____

bit scores
A raw score is just the score computed given the alignment. A bit score is a **normalized** score of alignments (kind of like standard deviation). The bit scores can then be compared across searches.

Let S be the raw alignment score. The bit score S' is defined as follows:

$$S' = (LS - \ln K) / \ln(2)$$

There is a relationship between E and S', just based on mathematical substitution:

$$E = mn2^{(-S')}$$

There is also a relationship with p:
p = probability that an alignment occurs by chance against random database with score S or better

$$p = 1 - e^{(-E)} \quad // \text{ e is the natural number}$$

So, the bigger the E (# of alignments by chance), p is _____.
The smaller the E, p is _____.

BLAST uses the E value since it is easier the reason about the number of expected alignments scoring S or better that would occur by chance rather than the chance of getting an alignment score at least S.

The L and K values are given by the extreme value distribution. There are different distributions for different conditions: affine gap penalties, linear gap penalties, shorter sequences (alignment scores will be smaller), and longer sequences.

Derivation of E:

$$E = kmne^{(-LS)}$$

$$-LS = -lnk - ln2 * S' \qquad [from\ defn\ of\ S' = (LS - lnk)/ln2)]$$

$$E = kmne^{(-lnk - ln2 * S')}$$

$$E = kmne^{-(lnk)}\, e^{-ln2 * S'}$$

$$E = \frac{kmn}{k} * 2^{-S'}$$

$$E = mn * 2^{-S'}$$

```
#  Matrix made by matblas from blosum62.iij
#  * column uses minimum score
#  BLOSUM Clustered Scoring Matrix in 1/2 Bit Units
#  Blocks Database = /data/blocks_5.0/blocks.dat
#  Cluster Percentage: >= 62
#  Entropy =   0.6979, Expected =  -0.5209
   A  R  N  D  C  Q  E  G  H  I  L  K  M  F  P  S  T  W  Y  V  B  Z  X  *
A  4 -1 -2 -2  0 -1 -1  0 -2 -1 -1 -1 -1 -2 -1  1  0 -3 -2  0 -2 -1  0 -4
R -1  5  0 -2 -3  1  0 -2  0 -3 -2  2 -1 -3 -2 -1 -1 -3 -2 -3 -1  0 -1 -4
N -2  0  6  1 -3  0  0  0  1 -3 -3  0 -2 -3 -2  1  0 -4 -2 -3  3  0 -1 -4
D -2 -2  1  6 -3  0  2 -1 -1 -3 -4 -1 -3 -3 -1  0 -1 -4 -3 -3  4  1 -1 -4
C  0 -3 -3 -3  9 -3 -4 -3 -3 -1 -1 -3 -1 -2 -3 -1 -1 -2 -2 -1 -3 -3 -2 -4
Q -1  1  0  0 -3  5  2 -2  0 -3 -2  1  0 -3 -1  0 -1 -2 -1 -2  0  3 -1 -4
E -1  0  0  2 -4  2  5 -2  0 -3 -3  1 -2 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
G  0 -2  0 -1 -3 -2 -2  6 -2 -4 -4 -2 -3 -3 -2  0 -2 -2 -3 -3 -1 -2 -1 -4
H -2  0  1 -1 -3  0  0 -2  8 -3 -3 -1 -2 -1 -2 -1 -2 -2  2 -3  0  0 -1 -4
I -1 -3 -3 -3 -1 -3 -3 -4 -3  4  2 -3  1  0 -3 -2 -1 -3 -1  3 -3 -3 -1 -4
L -1 -2 -3 -4 -1 -2 -3 -4 -3  2  4 -2  2  0 -3 -2 -1 -2 -1  1 -4 -3 -1 -4
K -1  2  0 -1 -3  1  1 -2 -1 -3 -2  5 -1 -3 -1  0 -1 -3 -2 -2  0  1 -1 -4
M -1 -1 -2 -3 -1  0 -2 -3 -2  1  2 -1  5  0 -2 -1 -1 -1 -1  1 -3 -1 -1 -4
F -2 -3 -3 -3 -2 -3 -3 -3 -1  0  0 -3  0  6 -4 -2 -2  1  3 -1 -3 -3 -1 -4
P -1 -2 -2 -1 -3 -1 -1 -2 -2 -3 -3 -1 -2 -4  7 -1 -1 -4 -3 -2 -2 -1 -2 -4
S  1 -1  1  0 -1  0  0  0 -1 -2 -2  0 -1 -2 -1  4  1 -3 -2 -2  0  0  0 -4
T  0 -1  0 -1 -1 -1 -1 -2 -2 -1 -1 -1 -1 -2 -1  1  5 -2 -2  0 -1 -1  0 -4
W -3 -3 -4 -4 -2 -2 -3 -2 -2 -3 -2 -3 -1  1 -4 -3 -2 11  2 -3 -4 -3 -2 -4
Y -2 -2 -2 -3 -2 -1 -2 -3  2 -1 -1 -2 -1  3 -3 -2 -2  2  7 -1 -3 -2 -1 -4
V  0 -3 -3 -3 -1 -2 -2 -3 -3  3  1 -2  1 -1 -2 -2  0 -3 -1  4 -3 -2 -1 -4
B -2 -1  3  4 -3  0  1 -1  0 -3 -4  0 -3 -3 -2  0 -1 -4 -3 -3  4  1 -1 -4
Z -1  0  0  1 -3  3  4 -2  0 -3 -3  1 -1 -3 -1  0 -1 -3 -2 -2  1  4 -1 -4
X  0 -1 -1 -1 -2 -1 -1 -1 -1 -1 -1 -1 -1 -1 -2  0  0 -2 -1 -1 -1 -1 -1 -4
* -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4 -4  1
```

Multiple Sequence Alignment

Problem: Given k strings (such as DNA or proteins), what is the best multiple global alignment and alignment cost?

Here's an example alignment of 3 strings:

```
--T--AC-C-AGT--TATGT-CAGGGGAC
AATTGCCGCC-GTCGT-T-TTCAG----C
-ATTGC-G--ATTCGTAT------GGGAC
```

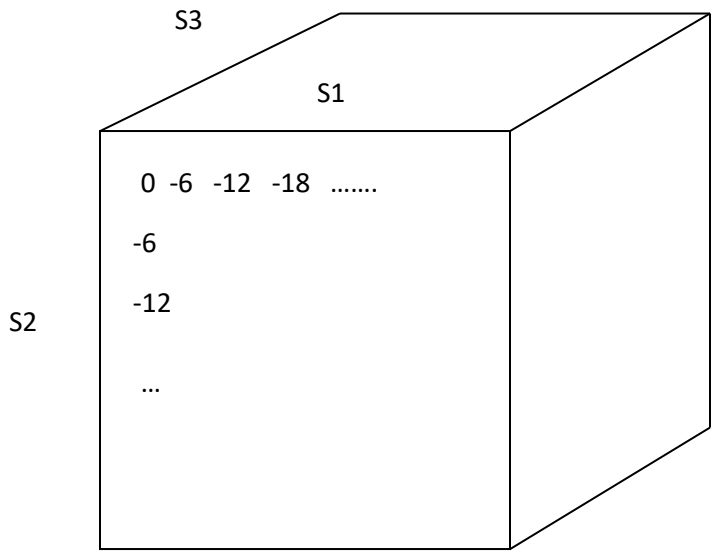


Figure 1: Cost table `t` for aligning three strings

Assume S1 = $a_1a_2a_3….a_M$
Assume S2 = $b_1b_2b_3…b_N$
Assume S3 = $c_1c_2c_3…c_P$

We create a cost table `t` of size (M+1) x (N+1) x (P+1). [Note: Also need to keep direction table to reconstruct alignment]

1. In the above 3-sequence alignment example, circle an aligned character set where they are all the same.

2. In the above 3-sequence alignment example, put a rectangle around an aligned character set where there are two sequences with a common character and the third sequence has a gap.

3. In the above 3-sequence alignment example, put an arrow toward an aligned character set where there are sequences with mismatched characters.

4. What are the possible character alignments for 3 sequences?

Algorithm for multiple global alignment for three strings (with no affine gaps):

`t[0][0][0] = 0`
Fill in top row with `i * cost(`a_i`, -, -)`.
The table above shows c((a_i, -, -)) = -6.
Fill in left column with `j * cost(-,` b_j`, -)`.
The table above shows c((-, b_j, -)) = -6.
Fill in left column top row (depth of cube) with `k * cost(-, -,` c_k`)`.
The table above shows c((-, -, c_k)) = -6.

Fill in table cells t[i][j][k] (i, j, k > 0) as:
`max` of the following 7 values:

`t[i-1][j][k] + c((`a_i`, -, -))`
`t[i][j-1][k] + c((-,` b_j`, -))`
`t[i][j][k-1] + c((-, -,` c_k`))`
`t[i-1][j-1][k] + c((`a_i`,` b_j`, -))`
`t[i-1][j][k-1] + c((`a_i`, -,` c_k`))`
`t[i][j-1][k-1] + c((-,` b_j`,` c_k`))`
`t[i-1][j-1][k-1] + c((`a_i`,` b_j`,` c_k`))`

The best global alignment score for the three sequences is in `t[M][N][P]`. To recreate the alignments, use three-dimensional direction table (now each cell can be calculated from one of seven directions).

5. What is the running time of calculating the optimal alignment between 3 sequences of lengths M, N, and P, respectively?

6. Now, what would be the running time this multiple sequence alignment algorithm for 100 sequences, each with length ~N?

7. Is this a practical approach for aligning several sequences?

Go to heuristics…

8. What idea(s) do you have for aligning 100 DNA sequences together in a faster way than the algorithm described above?

Ideas:
Idea 1: Calculate pairwise alignment scores for the K choose 2 sequences, where K = the number of sequences to align. Then combine distances and alignments together.

Example:
Best between 1 and 2:
```
AAAATTTT----
----TTTTGGGG
```

Best between 1 and 3:
```
----AAAATTTT
GGGGAAAA----
```

Best between 2 and 3:
```
TTTTGGGG----
----GGGGAAAA
```

9. What is the best overall alignment between the 3 sequences?

Idea 2:

a) Calculate pairwise alignments between each pair of sequences.
b) Perform alignment with most similar pair (highest global alignment score). Lock in this alignment. (So, if there is a gap or match/mismatch in that paired alignment, that will persist in the overall multiple sequence alignment).
c) For the rest of the (K-1) sequences, align the sequence that is most similar to one of the already aligned sequences in the growing alignment.

This is what the original CLUSTAL algorithm does.

CLUSTAL OMEGA extends CLUSTAL to allow the user to specify an external profile for alignment.

UPGMA Algorithm

Problem: Given n x n matrix of pairwise relationships between organisms, construct an evolutionary tree.

Suppose you found global alignment scores of a common gene in 11 organisms:

> Yeast YOR020c
mstllksaksivplmdrvlvqrikaqaktasglylpe
knveklnqaevvavgpgftdangnkvvpqvkvgdqvl
ipqfggstiklgnddevilfrdaeilakiakd

> Neurospora crassa
mattvrsvkslipllдrvlvqrvkaeaktasgiflpe
ssvkdlneakvlavgpgaldkdgkrlpmgvnagdrvl
ipqyggspvkvgeeeytlfrdseilakiae

> Aspergillus nidulans
msllrnvknlaplldrvlvqrvkpeaktasgiflpes
svkeqneakvlavgpgavdrngqripmgvaagdrvlv
pqfggsplkigeeeyhlfrdseilakine

> Schizosaccharomyces pombe (fission yeast)
matklksaksivplldrilvqrikadtktasgiflpe
ksveklsegrvisvgkggynkegklaqpsvavgdrvl
lpayggsnikvgeeeyslyrdhellaiike

> Mortierella alpina
masritkfsktivpmmdrvlvqrikpqqktasgiyip
ekaqealnegyvvavgkglttqegkvvpselaegdkv
llppyggsvvkvdneelilfreseilakiq

> Crypthecodinium cohnii
matgiakrftplldrvlvqrlkpeaktasglflpesa
akapnyatvlavgpggrtrdgdilpmnvkvgdkvvvp
eyggmtlkfedeefqvfrdadimgilne

> Drosophila melanogaster
maaaikkiipmldriliqraealtktkggivlpekav
gkvlegtvlavgpgtrnastgnhipigvkegdrvllp
efggtkvnlegdqkelflfresdilakle

> Homo sapiens
agqafrkflplfdrvlversaaetvtkggimlpeksq
gkvlqatvvavgsgskgkggeiqpvsvkvgdkvllpe
yggtkvvlddkdyflfrdgxilgky

> Geobacillus stearothermophilus
vlkplgdrvvievieteektasgivlpdtakekpqeg
rvvavgkgrvldsgervapevevgdriifskyagtev
kydgkeylilresdilavig

> Mycobacterium tuberculosis
makvnikpledkilvqaneaetttasglvipdtakek
pqegtvvavgpgrwdedgekripldvaegdtviysky
ggteikyngeeylilsardvlavvsk

> Mus musculus (house mouse)
magqafrkflllfdrvlversaaetvtkggimlpeks
qgkvlqatvvavgsggkgksgeiepvsvkvgdkvllp
eyggtkvvlddkdyflfrdsdilgkyvn

Table 1: Pairwise alignment scores between species

Yeast	Neur	Asp	Schiz	Mort	Cryp	Dros	Human	Geo	Myco	Mus	
	49	46	78	45	55	54	44	38	37	42	Yeast
		52	41	40	43	46	44	41	39	43	Neur
			43	48	45	45	40	40	38	39	Asp
				42	53	55	41	41	40	40	Schiz
					43	46	40	43	38	39	Mort
						61	43	34	36	45	Cryp
							49	42	36	49	Dros
								37	32	93	Human
									59	38	Geo
										32	Myco
											Mus

Approach: **UPGMA** – **U**nweighted **P**air **G**roup **M**ethod with **A**rithmetic mean to build evolutionary tree

UPGMA Algorithm:

First, assume each organism is in its own group

Until there is one group,

Merge together closest groups (using mean score)

Define closest as **highest** mean score between groups. Let S be the score between organism i and organism j.

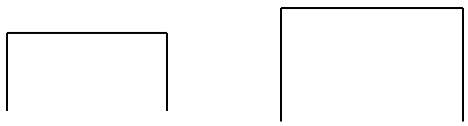
$$\text{mean}(G1, G2) = \frac{\Sigma_{i,j} S(i,j)}{|G1| * |G2|} \quad \text{where i is in G1 and j is in G2}$$

<u>Example of the algorithm applied to Table 1 data</u>

1. Merge human and mus (since maximum score is 93 in the table).

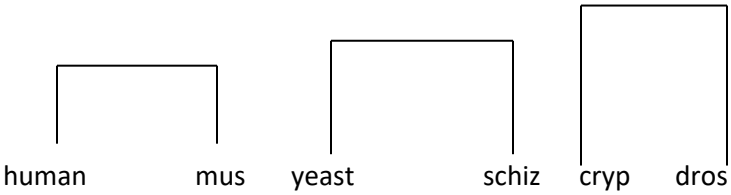
human mus

2. Merge yeast and schiz [score = 78]

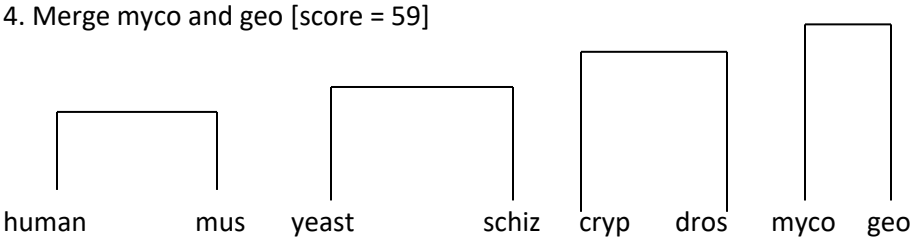


human mus yeast schiz

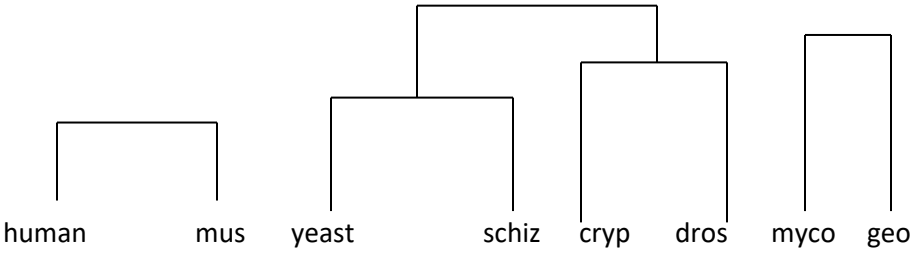
3. Merge cryp and dros [score = 61]



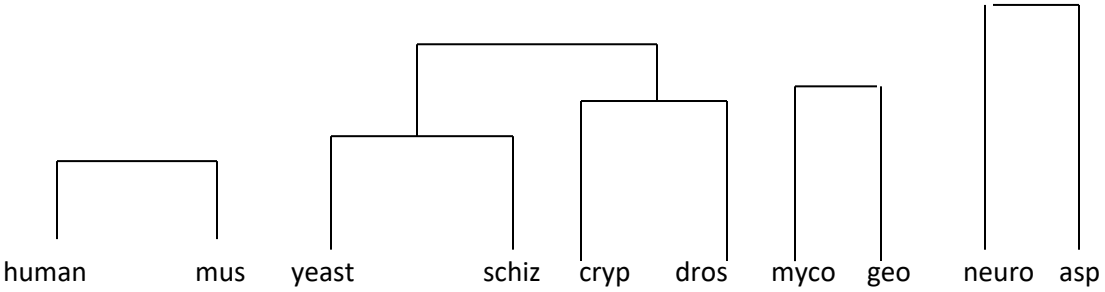
4. Merge myco and geo [score = 59]



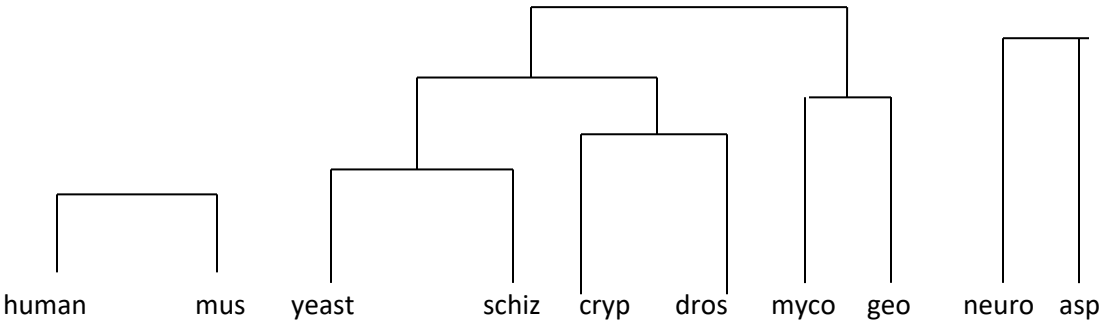
5. Next best score is {yeast, schiz} with {cryp, dros} [mean score = (55 + 54 + 53 + 55) / 4 = 54.25]



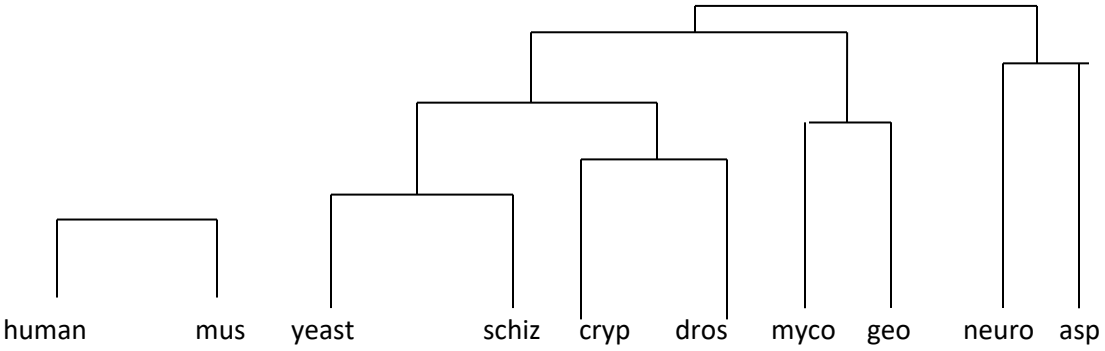
6. Merge nuero and asp [score = 52]



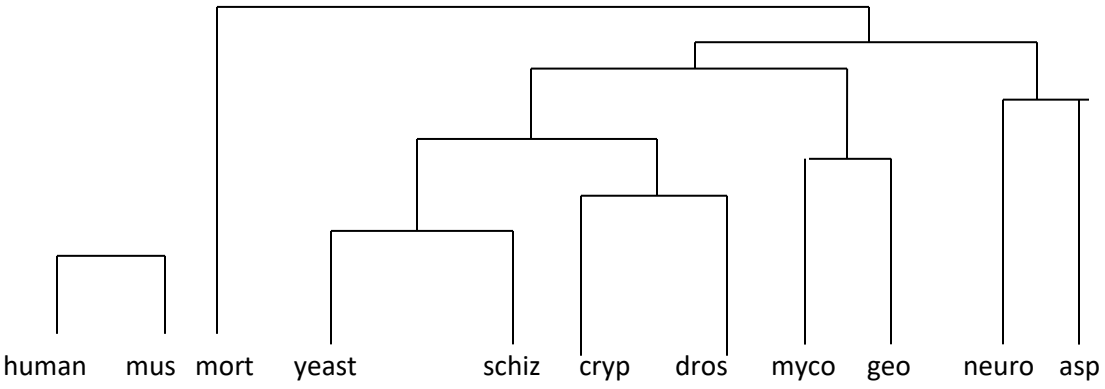
7. Next best is {myco, geo} with {yeast, schiz, cryp, dros} [mean score = (37 + 78 + 55 + 54 + 38 + 41 + 34 + 42)/8 = 47.3]



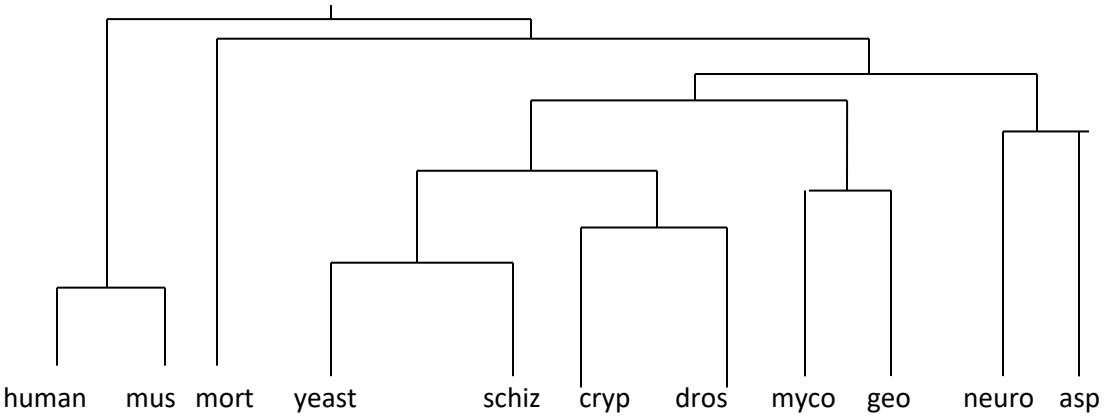
8. Next best is {neuro, asp} with {yeast, schiz, cryp, dros, myco, geo} [mean score = 43.4]



9. Next best is {yeast, schiz, cryp, dros, myco, geo, neuro, asp} with {mort} [mean score = 43.1]



10. Final merge is {human, mus} with {mort, yeast, schiz, cryp, dros, neuro, asp, myco, and geo}.



Example UPGMA Algorithm when using distances between organisms

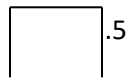
In the matrix with **global alignment scores**, higher numbers are better. But, in a distance matrix, lower numbers mean organisms are more closely related. With distance matrices, we can make the lengths of the tree branches mean the distance between the groups.

The height of a tree branch for G1 and G2 when combining groups G1 and G2 is: D(G1, G2) / 2. Some software programs will not divide by 2, so the height of just one branch indicates the distance.

Try one with *distances* as the matrix contents among 5 organisms:

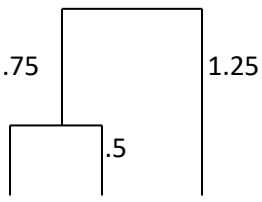
A	B	C	D	E	
	3	1	3	2	A
		4	3	4	B
			2	5	C
				4	D
					E

1. Merge A and C (distance is 1, which is the smallest on the table)



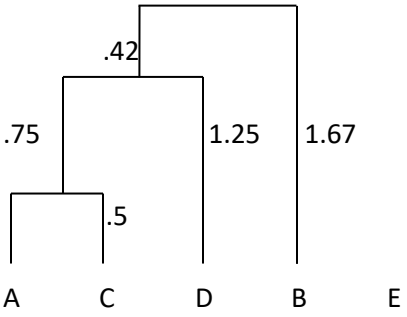
A C D B E

2. B with (A, C) is 3.5, B with D is 3, B with E is 4, D with E is 4, D with (A,C) is 2.5, E with (A,C) is 3.5
Best is D with (A, C).

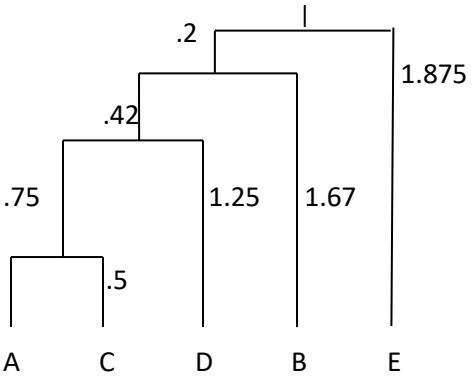


A C D B E

3. B with E is 4. (A, C, D) with B is 3.333, (A, C, D) with E is 3.66.
Best is (A, C, D) with B.



4. (A, C, D, B) with E has cost 3.75.



Note: This tree creation algorithm *assumes a constant evolutionary clock*. But the height can give us an indication of time (assuming a constant clock).

Practice: Build the tree with the following distance matrix

A	B	C	D	E	
	1	3	4	7	A
		8	6	2	B
			9	5	C
				3.5	D
					E

Phylogenetic Trees Overview

Question 1: Why are phylogenetic (evolutionary) trees helpful to construct?

Words: Phylogenetic trees have several names: guide trees, evolutionary trees, phylograms, cladograms

Draw an example phylogenetic tree here:

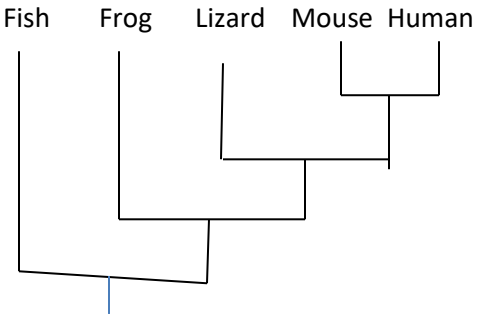
Interpretation:

- Some trees have a **root**; the root is the most recent common ancestor of the organisms in the tree.
- Some trees do not have a root. They just show relationships between organisms. These trees do not make assumptions about common ancestry.
- **Internal nodes** are hypothetical taxonomic units (HTUs).
- **Leaf nodes (terminal nodes)** are operational taxonomic units (OTUs).

Terminology:

- A **clade** is a single common ancestor and its decendants.
- A tree may be scaled, where the length of a branch denotes distance. This is called a **phylogram**.
- A tree may be unscaled, where the lengths of branches do not mean anything. The tree just shows relationships of ancestry. This is called a **cladogram**.

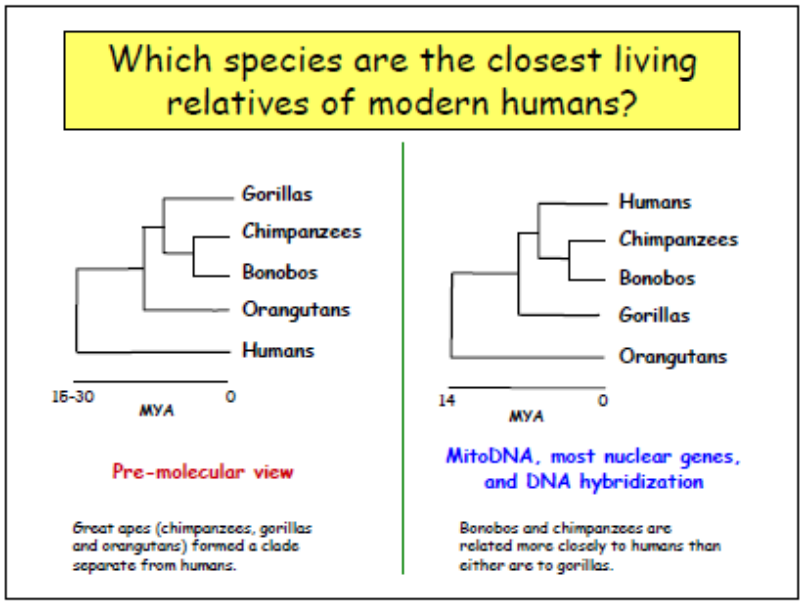
Question 2: Based on the tree below, is the frog more closely related to the fish or the human?



Question 3: Can you draw an equivalent tree to the one in question 2 where the tree looks different?

Question 4: Before bioinformatics, how were evolutionary trees constructed?

Biology example: before genetic data, humans were less related to chimpanzees.



Neighbor Joining Algorithm

Problem: Given a set of pairwise distances between organisms (also called taxa), find an unrooted tree with the organisms as the leaves of the tree.

Solution: UPGMA is one way to solve this problem, but this algorithm creates rooted trees. Another method is called Neighbor Joining (NJ), where at each step an internal node of the tree is created based on minimizing the distance between the joined taxa (groups) and maximizing the distances to all other taxa (groups). Note that in UPGMA, the algorithm seeks to only minimize the distance between the joined groups.

Algorithm:

Given: a distance matrix D (containing the pairwise distances between groups), N is the number of groups

Create a star tree T with a central hub and all groups connected via an edge to the hub

while N >= 3:

Step 1: For each group i, calculate the net divergence R_i from i to all other groups

$R_i = D_{i1} + D_{i2} + \ldots + D_{iN}$

Note: if D_{ij} is not in the matrix, look for D_{ji} instead

Step 2: Calculate a new score matrix M where $M_{ij} = D_{ij} - [(R_i + R_j)/(N- 2)]$

(Note: this M_{ij} value is the distance between i and j minus the distances to all other groups, so finding the minimum M_{ij} gives us close distance between i and j and far distances to all other groups.)

Step 3: Choose groups p and r where M_{pr} is the smallest. Create an internal node in T called u where new edges are added between u and p and u and r.

The edge distances can be calculated as follows:

$E_{pu} = (D_{pr} / 2) + [(R_p - R_u)/(2(N-2)]$

$E_{ru} = D_{pr} - E_{pu}$ // the remaining distance, if either should be negative // give the extra length to the other edge

Step 4: Define new distances from new node u to every other group i

$D_{iu} = (D_{pi} + D_{ri} - D_{pr})/ 2$

Step 5: N = N-1

Example:

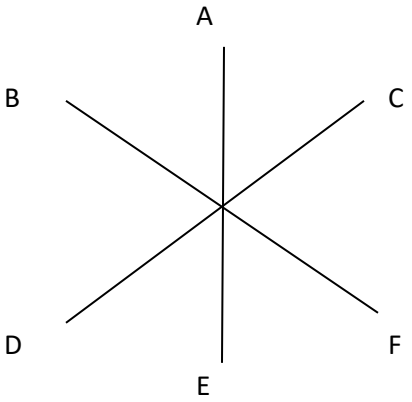
Suppose we have six organisms (or groups or OTUs – operational taxonomic unit) with the following distance matrix:

Table 1: Distance matrix D for six organisms (smaller numbers mean closer-related organisms)

A	B	C	D	E	F	
	5	4	7	6	8	A
		7	10	9	11	B
			7	6	8	C
				5	9	D
					8	E
						F

N = 6

T:



First iteration N= 6:

Step 1:

$R_A = 5 + 4 + 7 + 6 + 8 = 30$

$R_B = 5 + 7 + 10 + 9 + 11 = 42$

$R_C = 4 + 7 + 7 + 6 + 8 = 32$

$R_D = 7 + 10 + 7 + 5 + 9 = 38$

$R_E = 6 + 9 + 6 + 5 + 8 = 34$

$R_F = 8 + 11 + 8 + 9 + 8 = 44$

Step 2:

Matrix M. Let's look at the calculation for M_{AB}:

$M_{AB} = D_{AB} – (R_A + R_B)/(N-2)$
$M_{AB} = 5 – (30 + 42)/4$
$M_{AB} = 5 – (72/4) = 5 – 18 = -13$

Table 2: Matrix M of scores between groups

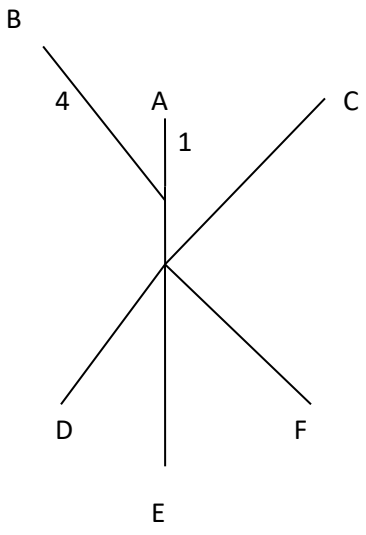
A	B	C	D	E	F	
	-13	-11.5	-10	-10	-10.5	A
		-11.5	-10	-10	-10.5	B
			-10.5	-10.5	-11	C
				-13	-11.5	D
					-11.5	E
						F

Step 3: Choose the smallest M value (-13). We could join groups A and B or D and E. We'll arbitrarily choose A and B to join.

Calculate distance of tree branches:
$E_{AU} = (D_{AB} / 2) + [(R_A – R_B)/(2(N-2)]$
$E_{AU} = (5/2) + (30-42)/(2*4) = 2.5 – 1.5 = 1$

$E_{BU} = D_{AB} – E_{AU}$
$E_{BU} = 5 – 1 = 4$

T:



Step 4:

Let U be the group of {A, B}.

General formula for calculating distances from U to other nodes: $D_{iu} = (D_{pi} + D_{ri} – D_{pr})/ 2$

$D_{CU} = (D_{AC} + D_{BC} – D_{AB}) / 2 = (4 + 7 – 5) / 2 = 3$
$D_{DU} = (D_{AD} + D_{BD} – D_{AB}) / 2 = 6$
$D_{EU} = (D_{AE} + D_{BE} – D_{AB})/2 = 5$
$D_{FU} = (D_{AF} + D_{BF} – D_{AB})/2 = 7$

New matrix D:

Table 3: Matrix D at the end of the first iteration

U	C	D	E	F	
	3	6	5	7	U
		7	6	8	C
			5	9	D
				8	E
					F

<u>Second Iteration: N = 5</u>

Step 1:

$R_U = 3 + 6 + 5 + 7 = 21$
$R_C = 3 + 7 + 6 + 8 = 24$
$R_D = 6 + 7 + 5 + 9 = 27$
$R_E = 5 + 6 + 5 + 8 = 24$
$R_F = 7 + 8 + 9 + 8 = 32$

Step 2:
$M_{ij} = D_{ij} – [(R_i + R_j)/(N– 2)]$

Table 4: Matrix M in second iteration

U	C	D	E	F	
	-12	-10	-10	-10.67	U
		-10	-10	-10.67	C
			-12	-10.67	D
				-10.67	E
					F

Step 3: Choose the smallest M value (-12). We could join groups C and U or D and E. We'll arbitrarily choose C and U to join.

Calculate distance of tree branches:

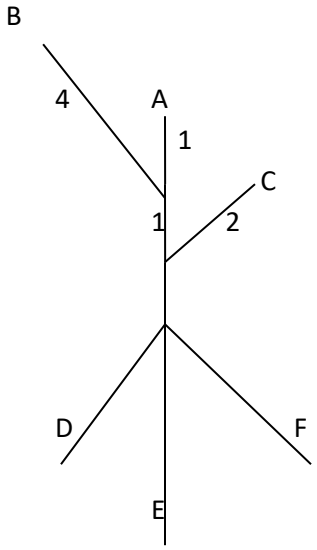
$E_{CQ} = (D_{CU} / 2) + [(R_C – R_U)/(2(N-2)]$

$E_{CQ} = (3/2) + 3/(2*3) = 2$

$E_{UQ} = D_{CU} – E_{CQ}$

$E_{UQ} = 3 – 2 = 1$

T:



Step 4:

Let Q be the group of {U, C}.

$D_{iQ} = (D_{Ui} + D_{Ci} - D_{UC})/ 2$

$D_{DQ} = (D_{UD} + D_{CD} - D_{UC}) / 2 = 5$
$D_{EQ} = (D_{UE} + D_{CE} - D_{UC})/2 = 4$
$D_{FQ} = (D_{UF} + D_{CF} - D_{UC})/2 = 6$

New matrix D:

Table 5: Matrix D at the end of the second iteration

Q	D	E	F	
	5	4	6	Q
		5	9	D
			8	E
				F

Third Iteration: N = 4
Step 1:

$R_Q = 5 + 4 + 6 = 15$
$R_D = 5 + 5 + 9 = 19$
$R_E = 4 + 5 + 8 = 17$
$R_F = 6 + 9 + 8 = 23$

Step 2:
$M_{ij} = D_{ij} - [(R_i + R_j)/(N- 2)]$

Table 6: Matrix M in third iteration

Q	D	E	F	
	-12	-12	-13	Q
		-13	-12	D
			-12	E
				F

Step 3: Choose the smallest M value (-13). We could join groups F and Q or D and E. We'll arbitrarily choose D and E to join.

Calculate distance of tree branches:

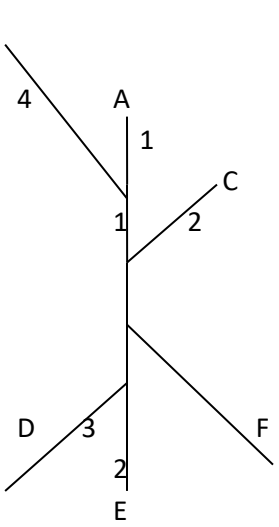
$E_{DP} = (D_{DE} / 2) + [(R_D - R_E)/(2(N-2)]$

$E_{DP} = (5/2) + (19-17)/4 = 2.5 + .5 = 3$

$E_{EP} = D_{DE} - E_{DP}$

$E_{EP} = 5 - 3 = 2$

T:



Step 4:

Let P be the group of {D, E}.

$D_{iP} = (D_{Di} + D_{Ei} - D_{DE})/ 2$

$D_{QP} = (D_{DQ} + D_{EQ} - D_{DE}) / 2 = 2$

$D_{FP} = (D_{DF} + D_{EF} - D_{DE})/2 = 6$

New matrix D:

Table 7: Matrix D at the end of the third iteration

Q	P	F	
	2	6	Q
		6	P
			F

Fourth Iteration: N = 3

Step 1:

$R_Q = 2 + 6 = 8$
$R_P = 2 + 6 = 8$
$R_F = 6 + 6 = 12$

Step 2:
$M_{ij} = D_{ij} – [(R_i + R_j)/(N– 2)]$

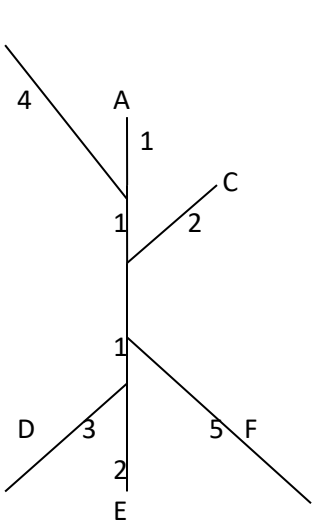
Table 8: Matrix M in fourth iteration

Q	P	F	
	-14	-14	Q
		-14	P
			F

Step 3: Choose the smallest M value (-14). We could join any of the groups. We'll arbitrarily choose to join P and F.

Calculate distance of tree branches:
$E_{PR} = (D_{PF} / 2) + [(R_P – R_F)/(2(N-2)]$
$E_{PR} = (6/2) + (8-12)/2 = 3 – 2 = 1$
$E_{FR} = D_{PF} – E_{PR}$
$E_{FR} = 6 – 1 = 5$

T:



Step 4:

Let R be the group of {P, F}.

$D_{iR} = (D_{Pi} + D_{Fi} - D_{PF})/ 2$

$D_{QR} = (D_{PQ} + D_{FQ} - D_{PF}) / 2 = (2 + 6 - 2) / 2 = 3$

New matrix D:

Table 9: Matrix D at the end of the fourth iteration

Q	R	
	3	Q
		R

DONE

Final Tree:

T:

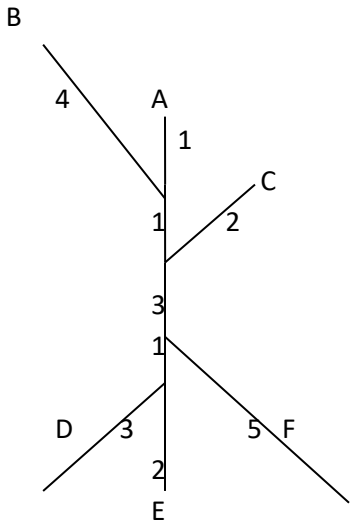


Table 10: Original Distances

A	B	C	D	E	F	
	5	4	7	6	8	A
		7	10	9	11	B
			7	6	8	C
				5	9	D
					8	E
						F

Let's see how close the tree is to the original distances:

A and B is 5 //correct
A and C is 4 //correct
A and D is 9 //7
A and E is 8 //6
A and F is 11 //8
C and B is 7 //correct
B and D is 12 //10
E and B is 11 //9
F and B is 13 //11
D and C is 9 //7
E and C is 8 //6
F and C is 10 //8
E and D is 5 //correct
F and D is 9 //correct
F and E is 8 //correct

What is the runtime of neighbor joining? _____

What happens to the matrix each iteration?_____

Is neighbor joining greedy? _____

Is neighbor joining optimal? _____

Maximum Parsimony for Building Trees

1. How many different rooted trees are there for 2 organisms?

2. How many different rooted trees are there for 3 organisms?

3. How many different unrooted trees are there for 3 organisms?

4. How many different unrooted trees are there for 4 organisms?

5. How many possible trees are there if we want to search for the optimal tree showing relationships among organisms?

Table 1: Number of unrooted and rooted trees given the number of OTUs

# OTUs (organisms)	# Unrooted trees	# Rooted trees
2	1	1
3	1	3
4	3	15
5	15	105
6	105	945
7	954	10395
8	10395	135135
9	135135	34459425
10	34459425	2.13 x 10^(15)

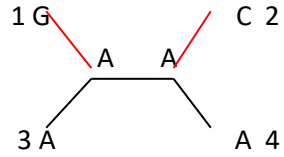
Maximum parsimony method: choose the tree that requires the fewest evolutionary events (substitutions) to explain the sequences at the tree leaves.

Example: Suppose we have a multiple sequence alignment of DNA from 4 organisms (OTUs) and we want to build the maximum parsimony tree.

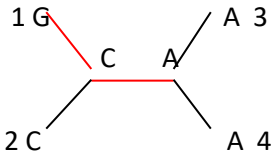
Let's look at the three different trees for site #3 below in the sequences.

Input	Site #								
Seq #	1	2	3	4	5	6	7	8	9
1	A	A	G	A	G	T	T	C	A
2	A	G	C	C	G	T	T	C	T
3	A	G	A	T	A	T	C	C	A
4	A	G	A	G	A	T	C	C	T

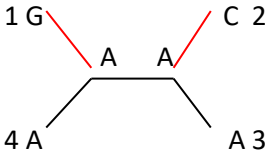
Let's look at site 3 and build the 3 possible trees showing the relationships among the four organisms.



This tree has 2 changes along edges



This tree has 2 changes along edges



This tree has 2 changes along edges

Thus, this site gives no useful information, since no tree is better than the others.

Maximum Parsimony:

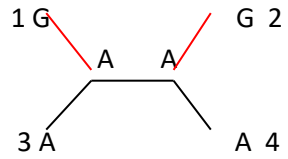
Step 1: Identify informative sites

An informative site is one in which there are at least two different characters at the site, each of which is represented by at least two different sequences.

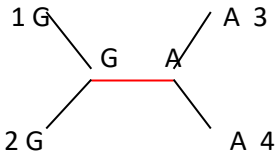
In the example above with the four sequences, which site(s) is/are informative?

Step 2: Build all possible trees for every informative site.

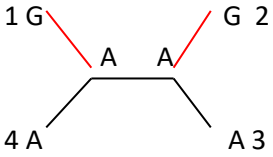
Site 5:



Tree 1 has 2 changes

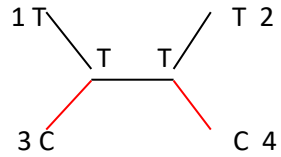


Tree 2 has 1 change

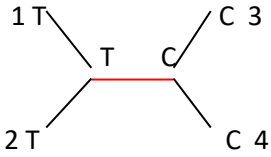


Tree 3 has 2 changes

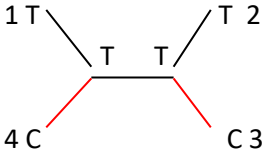
Site 7:



Tree 1 has 2 changes

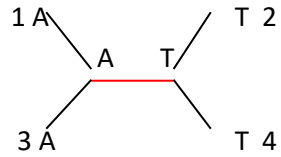


Tree 2 has 1 change

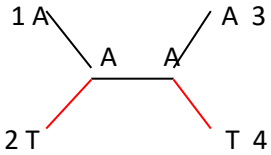


Tree 3 has 2 changes

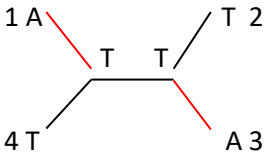
Site 9:



Tree 1 has 1 change



Tree 2 has 2 changes



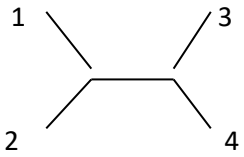
Tree 3 has 2 changes

Step 3: Calculate minimum scoring tree by summing the number of changes for each tree over all sites.

	Site 5	7	9	SUM
Tree1	2	2	1	5
Tree2	1	1	2	4
Tree3	2	2	2	6

Tree2 is the best tree for these four sequences.

Output of maximum parsimony algorithm:



Note: If we use maximum parsimony for 5 organisms (OTUs), we would generate the 15 possible trees for each informative site and choose the best tree. At some point, this exhaustive search technique can no longer be applied due to the huge number of possible trees. Then, we can resort to heuristics for throwing away (pruning) bad trees and extending/modifying the good trees.

How can we tell if the tree we get is robust?

Well, in computational biology, we cannot always verify the accuracy of the results of our heuristics. The trees that are created are just hypotheses. One way we can see if a tree is robust is to do the following:

1. Start with original data and original tree produced
2. For 1000 times
    a. Randomly re-sample sites with replacement
    b. Build tree with re-sampled data
    c. Determine if each clade in original tree is observed in the tree from the random re-sampling
3. Calculate frequency with which each clade appears in the trees
4. If frequency > 70%, the clade designation is robust.

example:

Original sequence data:

	Site #								
Seq #	1	2	3	4	5	6	7	8	9
1	A	A	G	A	G	T	T	C	A
2	A	G	C	C	G	T	T	C	T
3	A	G	A	T	A	T	C	C	A
4	A	G	A	G	A	T	C	C	T

Re-sampled data with replacement for one of the 1000 trials (note that sites 6 and 7 are sampled twice whereas sites 3 and 9 are not used at all – due to the randomness):

	Site #								
Seq #	5	4	7	6	7	6	2	1	8
1	G	A	T	T	T	T	A	A	C
2	G	C	T	T	T	T	G	A	C
3	A	T	C	T	C	T	G	A	C
4	A	G	C	T	C	T	G	A	C

DNA Microarrays

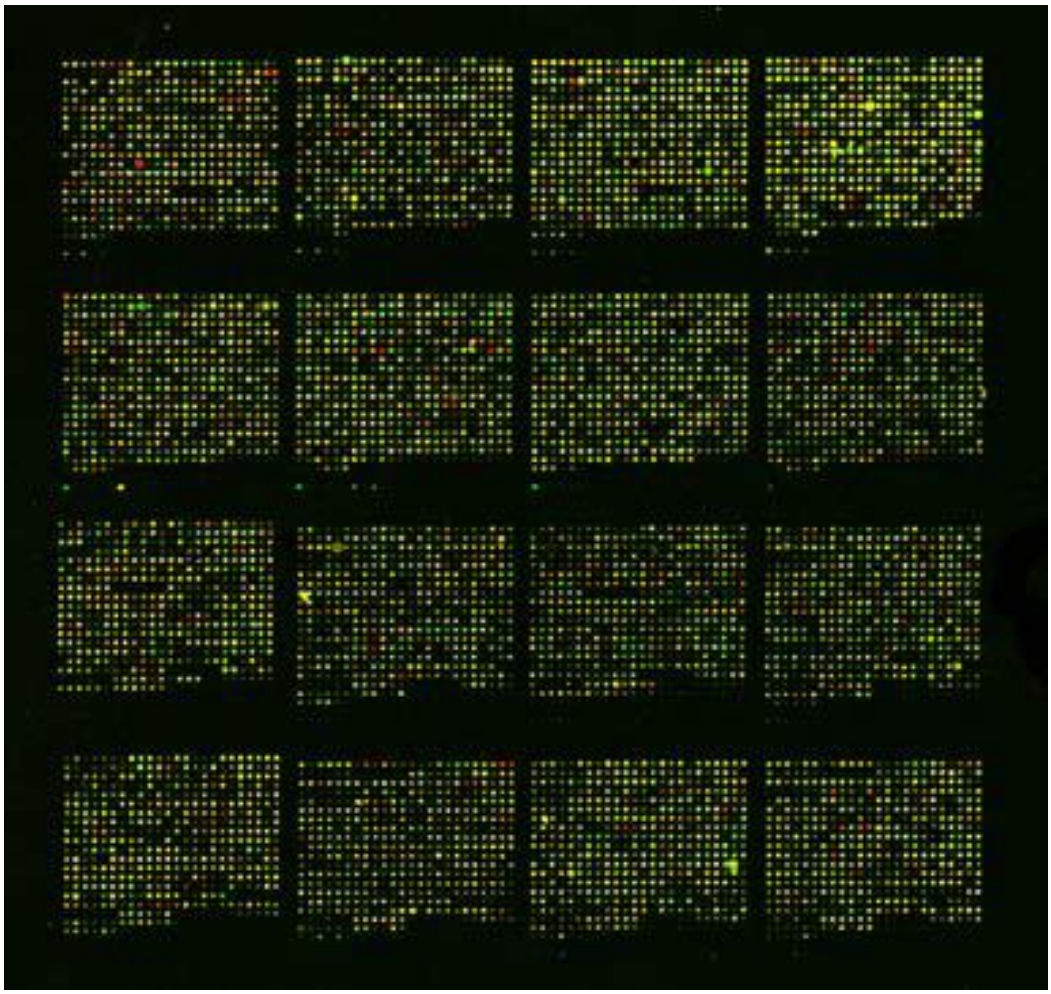


Figure 1: Yeast genome DNA microarray

Colors show intensity of hybridization level of each probe (each spot above) to the target mRNA.

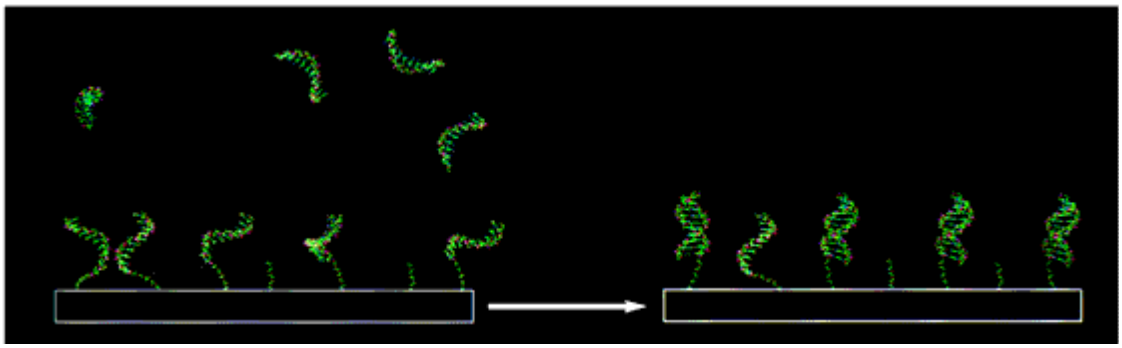


Figure 2: mRNA attaching to probes

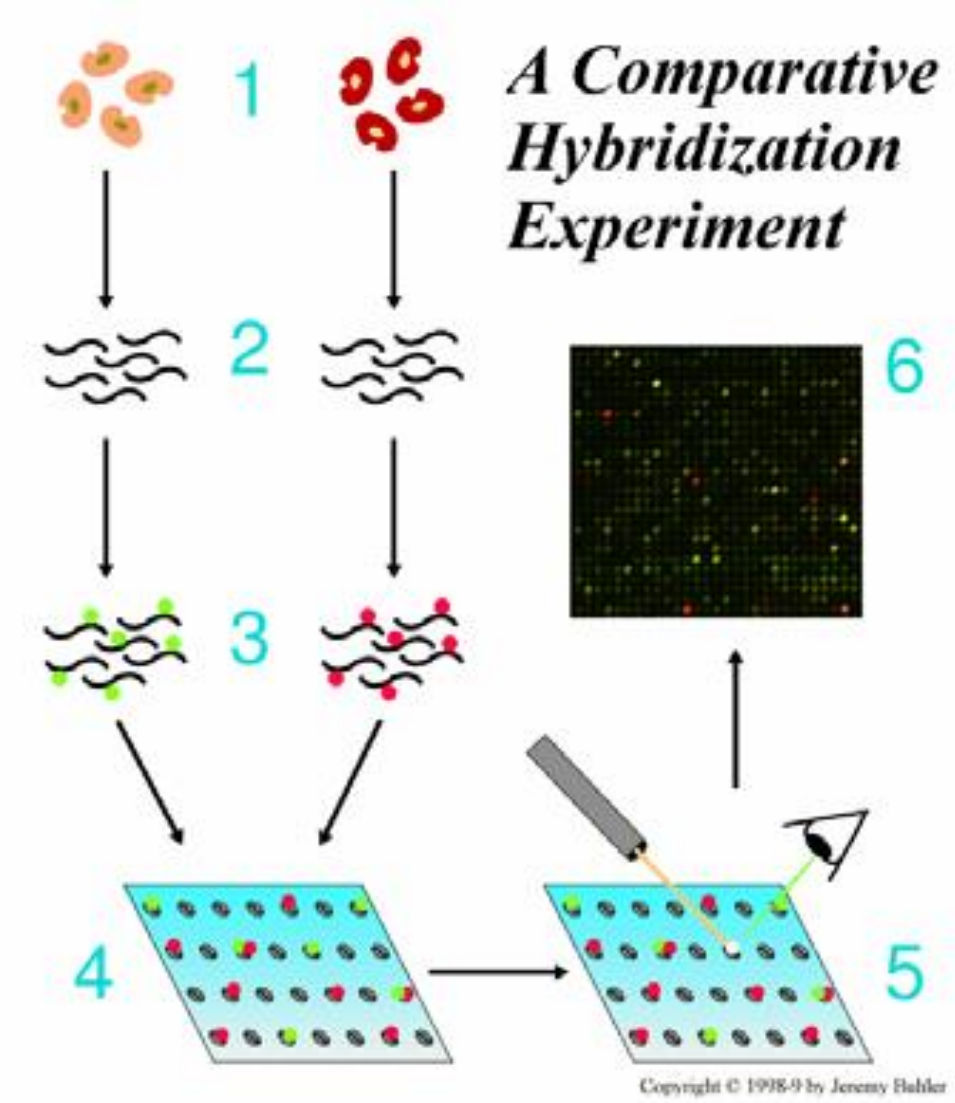


Figure 3: A two-channel experiment (figure by Jeremy Buhler)

Resulting data from M experiments can be organized into a matrix, as follows:

	exp1	exp2	exp3	exp4	… expM
gene1	1.3	0.4	3.2	1.2	2.2
gene2	2.5	2.2	1.1	3.1	2
gene3	.7	0.0	0.2	0.1	4
…					
…					
geneN	1.1	1.4	2.6	2.4	1.7

Activity: Microarray Hybridization

Suppose the following 20-mer target single-stranded cDNA is washed over the DNA microarray.

5′ GUACACCGUUAACGAUGAAU 3′

Suppose the following DNA 20-mers are on the microarray. (Remember 3′ and 5′ ends)

5′ ATTCATCGTTGGCGGTGTAC 3′

How well will the target hybridize to this DNA probe? _____

5′ TATCATTGATGGGCCTGTCC 3′

How well will the target hybridize to this DNA probe? _____

5′ TAATCCCCAACCCGCACATG 3′

How well will the target hybridize to this DNA probe? _____

RNA-seq

Similar to DNA microarrays in determining gene expression levels. It is the more common method for gene expression experiments now.

Instead of using probes and targets, the mRNA is extracted from the cell and sequenced. These mRNA fragments are then converted to cDNA for stability. The fragments are sequenced (read the bases at each position). Then, the fragments are compared to libraries to determine their association to genes. Counts are produced for each gene across each sample. Then, one can look at the fold change of expression of a gene between samples.

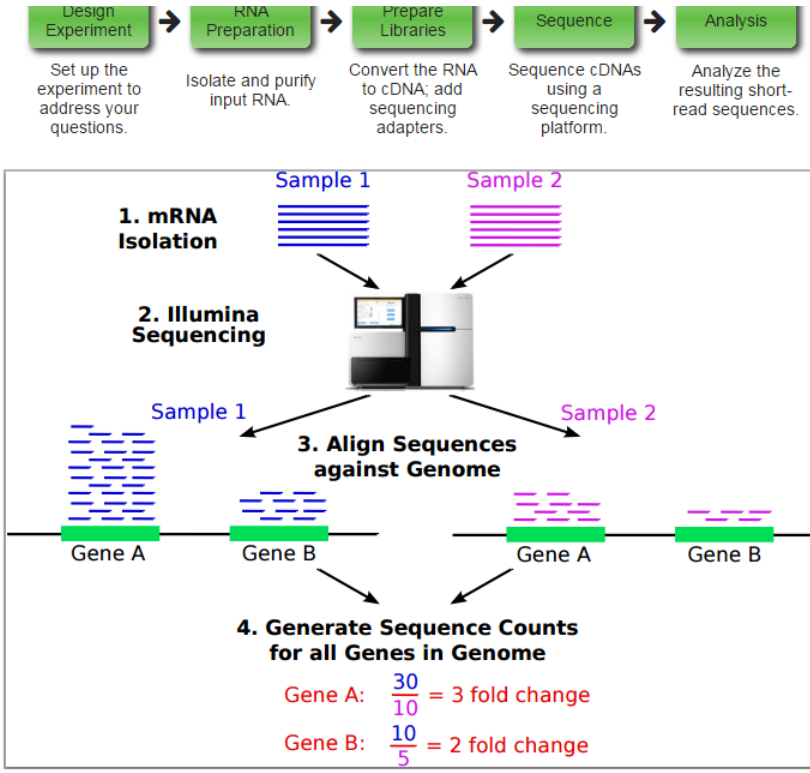


Figure from Figure 1: Overview of RNA-seq technology,
http://biocluster.ucr.edu/~rkaundal/workshops/R_mar2016/RNAseq.html#overview-of-rna-seq

Matrix has one gene per row.
Matrix has one sample per column.

The numbers are counts of that gene in the sample, which gives an indication of expression level. Note that the data is still subject to noise from each stage of the RNA-seq pipeline. Usually, you want to normalize the counts in some way so the expression levels can be comparable across samples.

GeneID	Sample_A	Sample_B	Sample_C	Sample_D	Sample_E	Sample_F	Sample_G	Sample_H	Sample_I
ENSG00000223972	23	11	31	9	11	13	17	17	22
ENSG00000227232	1000	828	1078	758	728	897	1075	793	1089
ENSG00000243485	8	6	2	2	3	4	2	5	6
ENSG00000237613	1	1	0	2	1	4	5	1	2
ENSG00000238009	107	69	85	66	87	64	89	55	81
ENSG00000233750	16	5	23	10	4	21	14	21	20
ENSG00000237683	1259	1025	1375	990	997	1109	1141	693	973
ENSG00000268903	3652	3422	2725	3274	3384	2154	2798	5761	6089
ENSG00000239906	25430	21022	13947	45938	47405	28038	8557	17889	16544
ENSG00000241860	194936	184076	162085	172115	164332	118233	146396	221478	262352
ENSG00000222623	49492	44102	41514	43487	43009	32654	40010	53883	65989
ENSG00000241599	4	10	3	6	5	2	5	9	6
ENSG00000228463	34074	32072	24434	41568	41246	27624	19095	39606	38636
ENSG00000237094	48499	45757	32395	77500	84031	57687	19371	32145	36202
ENSG00000250575	1	0	0	0	1	0	2	0	0
ENSG00000233653	0	1	3	1	0	2	0	0	0
ENSG00000235249	549	434	605	427	427	523	425	333	448
ENSG00000256186	599	591	842	683	724	843	700	391	478
ENSG00000236601	1	1	0	0	0	2	0	0	0
ENSG00000236743	91	57	85	59	58	70	82	57	70
ENSG00000236679	7	2	8	3	2	1	0	1	0
ENSG00000231709	266	213	297	191	210	300	299	174	274
ENSG00000235146	336	267	399	333	390	371	329	196	300
ENSG00000239664	25	14	30	30	29	23	16	13	12
ENSG00000230021	6	11	14	7	5	6	8	6	6
ENSG00000223659	4	7	10	5	12	12	7	4	7
ENSG00000225972	1	2	0	1	4	0	4	0	1

K-means Clustering (Lloyd's Heuristic)

Formal problem: Given N items with known distances between items and K clusters, assign the N items into one of K clusters such that the total distance from each item to its cluster center is minimized.

(Finding the optimal set of clusters takes exponential time, so k-means is a randomized, greedy heuristic to solve this problem more quickly.)

K-means clustering heuristic (Lloyd):
Input: n items, k = number of clusters to create

For each item, randomly assign it to one of the k clusters.
Calculate cluster centers for each cluster.
While there are new cluster assignments:
    For every item i:
        Assign i to the cluster with closest center.
    Re-compute cluster centers for each cluster.

To calculate the square of Euclidean distance from item c to item t:
$D(<c_1, c_2, \ldots, c_m>, <t_1, t_2, \ldots, t_m>) = \Sigma_i [(c_i - t_i)^2]$ // note that we do not need to take sqrt
// since all distances are positive and this
// reduces the number of computations

To find the center c from a set of items T:
center(T) = vector sum of items in T
| T |

Example k-means clustering:
Suppose our microarray data is the following with k = 2

	exp1	exp2
gene1	0	0
gene2	0	1
gene3	1	1
gene4	1	0
gene5	.5	.5
gene6	5	5
gene7	5	6
gene8	6	6
gene9	6	5
gene10	5.5	5.5

Step 1: Randomly assign each item to one of two clusters:

C_1 = {gene1, gene4, gene5, gene8}
C_2 = {gene2, gene3, gene6, gene7, gene9, gene10}

Step 2: Calculate cluster centers:

Center(C_1) = (<0, 0> + <1, 0> + <.5, .5> + <6, 6>) / 4
= <7.5, 6.5> / 4
= <1.875, 1.625>

Center(C_2) = (<0, 1> + <1, 1> + <5, 5> + <5, 6> + <6, 5> + <5.5, 5.5>) / 6
= <22.5, 23.5> / 6
= <3.75, 3.917>

While no new cluster assignments:

Calculate distance from each point to cluster center to assign point to cluster:

Item	**Sq Distance to center C_1**	**Sq Distance to center C_2**	**Cluster assignment**
<0, 0>	$1.875^2 + 1.625^2 = 6.15625$	$3.75^2 + 3.917^2 = 29.405$	C_1
<0, 1>	3.91	22.57	C_1
<1, 1>	1.16	16.07	C_1
<1, 0>	3.41	22.91	C_1
<.5, .5>	3.15	22.24	C_1
<5, 5>	21.16	2.74	C_2
<5, 6>	28.91	5.90	C_2
<6, 6>	36.16	9.40	C_2
<6, 5>	28.41	6.24	C_2
<5.5, 5.5>	28.16	5.57	C_2

Calculate cluster centers:
Center(C_1) = (<0, 0> + <0, 1> + <1, 1> + <1, 0> + <.5, .5>) / 5
= <.5, .5>

Center(C_2) = (<5, 5> + <5, 6> + <6, 6> + <6, 5> + <5.5, 5.5>) / 5
= <5.5, 5.5>

Calculate distance from each point to cluster center to assign point to cluster:

Item	**Sq Distance to center C_1**	**Sq Distance to center C_2**	**Cluster assignment**
<0, 0>	.5	60.5	C_1
<0, 1>	.5	50.5	C_1
<1, 1>	.5	40.5	C_1
<1, 0>	.5	50.5	C_1
<.5, .5>	0	50.0	C_1
<5, 5>	40.5	.5	C_2
<5, 6>	50.5	.5	C_2
<6, 6>	60.5	.5	C_2
<6, 5>	50.5	.5	C_2
<5.5, 5.5>	50.0	0	C_2

No new cluster assignments, so STOP. The final assignment is in the above table.

QT Clustering

Problem: Given N items, find the best clusters.

Note: In k-means clustering, the value of k is given as input (so the # of clusters to find must be known before the clustering takes place).

QT (Quality Threshold) Clustering is a clustering solution that instead requires as input the maximum diameter (distance between furthest items) for any cluster and finds clusters that do not exceed the maximum diameter.

QT Clustering Algorithm:
Input: N items, maximum diameter for clusters

S = all N items
While S has items:
    For each item i in set S of items to cluster:
        Build the largest cluster with item i by choosing the nearest item, the next nearest, etc
            until cluster diameter is exceeded
    Save the cluster with the most items as a true cluster and remove the items from S

---

example:
Assume the dataset is the following:

	exp1	exp2
g1	3	4
g2	3	5
g3	6	7
g4	0	2
g5	0	1
g6	2	2

Assume the maximum diameter is 3

S = {g1, g2, g3, g4, g5, g6}
For each item, find the largest cluster:
g1: {(3, 4), (3, 5)} //note: (2,2) is too far from (3,5) to be included
g2: {(3, 5), (3, 4)}
g3: {(6, 7)}
g4: {(0, 2), (0, 1), (2, 2)}
g5: {(0, 1), (0, 2), (2, 2)}
g6: {(2, 2), (0, 2), (0, 1)}

Choose cluster with most items: {(0, 2), (0, 1), (2, 2)} and remove these from S.
S = {g1, g2, g3}

For each item, find the largest cluster:
g1: {(3, 4), (3, 5)}
g2: {(3, 5), (3, 4)}
g3: {(6, 7)}

Choose cluster {(3, 4), (3, 5)} and remove these from S.
S = {g3}

Choose cluster {(6, 7)}.
Final Clusters: {{(0, 2), (0, 1), (2, 2)}, {(3, 4), (3, 5)}, {(6, 7)}}

Hierarchical Clustering

Hierarchical clustering uses the process that the UPGMA uses for creating trees. Instead of using alignment scores as distances, we can use any items for which we have pairwise distances (such as microarray data).

Input: distance matrix D of size n x n (for n items to cluster)
Output: a tree where each leaf is an item (so the reader can then choose appropriate cluster membership and number of clusters by looking at the tree)

Algorithm:

1. Form n clusters, each with item as its own cluster.
2. Construct a tree T where each item is a leaf node.
3. While there is more than one cluster:
    a. Find the two closest clusters C_i and C_j
    b. Merge C_i and C_j to C so it has all elements from each cluster
    c. Compute distance from C to other clusters
    d. Add new vertex in T to link clusters C_l and C_j [length of vertex to leaf is half the distance from C_1 to C_2]
    e. Remove rows and columns associated with C_i and C_j
    f. Add new row and column for C

The way distance is calculated can vary. Among many options are the following:

Option 1: use minimum distance between items in clusters

Option 2: use average distance between all points in each cluster (UPGMA)

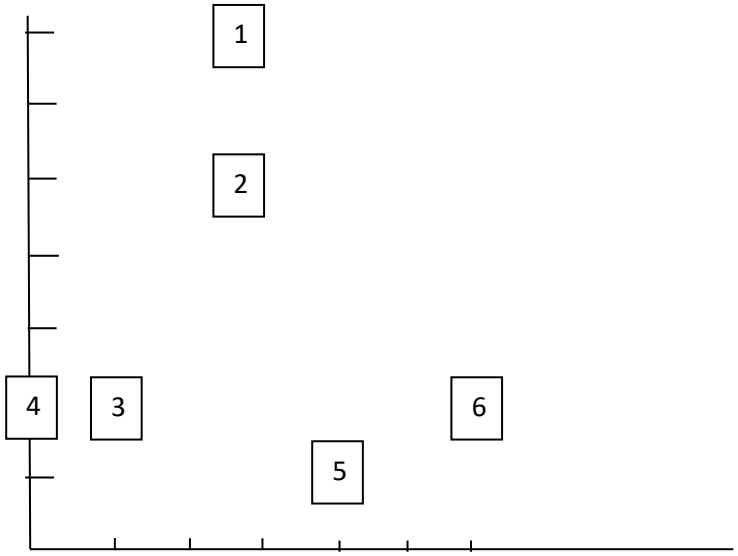
Option 3: can use distance that tries to separate clusters (similar to neighbor joining)

$$\frac{d(C^*, C_1) + d(C^*, C_2) - d(C_1, C_2)}{2}$$ // when merging C_1 and C_2, C^* is another cluster

EXAMPLE:

Suppose we have 6 genes:

	exp1	exp2
G1	3	7
G2	3	5
G3	1	2
G4	0	2
G5	4	1
G6	6	2



Need to calculate the distance matrix – using Euclidean (input to hierarchical clustering):

G1	G2	G3	G4	G5	G6	
	2	5.39	5.83	6.08	5.83	G1
		3.61	4.24	3.61	4.25	G2
			1	3.16	5	G3
				4.12	6	G4
					2.24	G5
						G6

Use metric: closest distance between closest elements in each cluster metric

Merge closest: G3 and G4

Update distances

G1	G2	G5	G6	{G3, G4}	
	2	6.08	5.83	5.39	G1
		3.61	4.25	3.61	G2
			2.24	3.16	G5
				5	G6
					{G3, G4}

Merge closest: G1 and G2

Update distances

G5	G6	{G3, G4}	{G1, G2}	
	2.24	3.16	3.61	G5
		5	4.25	G6
			3.61	{G3, G4}
				{G1, G2}

Merge closest: G5 and G6

Update distances

{G3, G4}	{G1, G2}	{G5, G6}	
	3.61	3.16	{G3, G4}
		3.61	{G1, G2}
			{G5, G6}

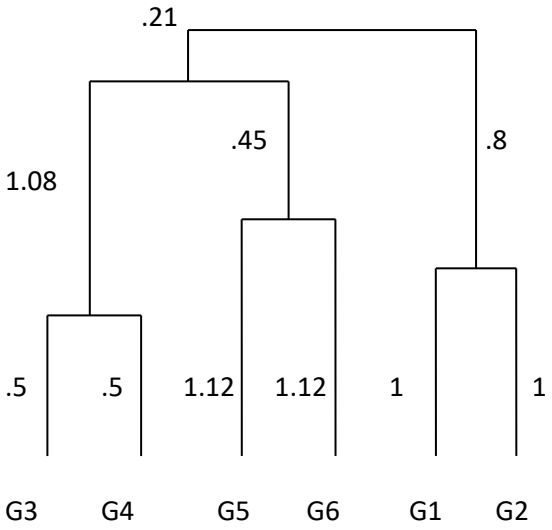
Merge closest: {G3, G4} with {G5, G6}

Update distances

{G1, G2}	{G3, G4, G5, G6}	
	3.61	{G1, G2}
		{G3, G4, G5, G6}

Merge closest: {G1, G2} with {G3, G4, G5, G6}

Overall hierarchical tree:



Note: we could have used other metrics to determine distance between clusters. Now, you can draw a line cutting through the tree to create 1 cluster, 2 clusters, 3 clusters, 4 clusters, 5 clusters, or 6 clusters.

Note: some software programs will make the height the actual distance instead of half the distance.

Practice:

Use K-means (K = 2) to cluster the same dataset. Need to randomly assign genes into one of two clusters to seed the algorithm. Use a coin flip for the randomness.

	exp1	exp2
G1	3	7
G2	3	5
G3	1	2
G4	0	2
G5	4	1
G6	6	2

Practice:

Use QT (diameter = 3.5) to cluster the same dataset.

	exp1	exp2
G1	3	7
G2	3	5
G3	1	2
G4	0	2
G5	4	1
G6	6	2

Expectation-Maximization to find motifs

Assume we have five sequences for which we believe there is a common 16-mer motif:

```
> Escherichia coli
TTGATTCCCTGAATGCCCGCTTAGTGTAACACTACTGTAACCGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCACAAAGGTTAGCAATAACTGCTGGGAAAATTCCG
AGTTAGTCGTTATATTCTAT
> Haemophilus influenzae
ATCTAACGGTACGGATTCTCCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCGGCATAAGTAGTAAGTGCCGTGCCTCCGCCATT
AGTTACACTAGTGGGACACC
> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAACGTTACACGAGTGTAACCGCCGAA
CCATGTTGACACGAATTCTG
> Salmonella typhi
GGTCGGCTTAGACTAGTGTGACCAAAAAGCTTTTGCTGAAGTTTCAGGGTAAGAAGAACC
AGCTCCTAGTAAAAAGACTATTGTGACTGAAAAGCGCGTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATGCCGGCGCCGC
TGCTGGAACGCCTCGACCCCAGGGCTACACTAGTTTAACCGGAACGCCGCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC
```

How do we find the motif? If we had a profile, we could easily find the best scoring 16-mer in each sequence. But we also need the profile.

Solution: Expectation-Maximization (EM)

Input: Set of sequences S thought to have a common motif and L, the length of the motif
Output: The motif profile and the instances of each sequence that best fit that motif profile.

Algorithm:

1. Create motif instances by randomly guessing the possible locations of the L-mer motif for each item in S
2. Repeat until convergence:
    a. Calculate a motif model given motif instances
    b. For each item in S, calculate new locations for the highest-scoring motif instances given the model generated in (a)

Example:

Let's run the algorithm on the sequences above.

Step 1: Randomly guess positions of 16-mers in each sequence (these are bold and separated by spaces)

```
> Escherichia coli
TTGATTCCCTGAATGCCCGCTTAGTGTAACACTACTGTAACCGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCACAAAGGTTAGCAATAACTGCTGGGAA AATTCCG
AGTTAGTCG TTATATTCTAT
> Haemophilus influenzae
A TCTAACGGTACGGATT CTCCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCGGCATAAGTAGTAAGTGCCGTGCCTCCGCCATT
AGTTACACTAGTGGGACACC
> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAACGTTACACGAGTGTAACCGCCGAA
CC ATGTTGACACGAATTC TG
> Salmonella typhi
GGTCGGCTTAGACTAGTGTGACCAAAAAGCTTTTGCTGAAGTTTCAGGGTAAGAAGAACC
AGCTCCTAGTAAAAAGACTAT TGTGACTGAAAAGCGC GTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATG CCGGCGCCGC
TGCTGG AACGCCTCGACCCCAGGGCTACACTAGTTTAACCGGAACGCCGCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC
```

Step 2: (Repeat until convergence)

a. Calculate motif profile based on those substrings

```
A .40 .20 0.0 .20 .40 0.0 .20 .20 .40 .40 .20 .60 .20 .20 0.0 0.0
C .20 .40 0.0 0.0 .40 .60 .20 .40 0.0 .40 .20 0.0 .20 0.0 .20 .40
G 0.0 .20 .40 .40 0.0 .40 .40 .40 .40 0.0 .20 .40 .60 .20 .40 .40
T .40 .20 .60 .40 .20 0.0 .20 0.0 .20 .20 .40 0.0 0.0 .60 .40 .20
```

b. Find highest scoring instance given this motif profile in each sequence

```
> Escherichia coli
TTGATTCCCTGAATGCCCGCTTAGTGTAACACTACTGTAACCGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCAC AAAGGTTAGCAATAAC TGCTGGGAAAATTCCG
AGTTAGTCGTTATATTCTAT
> Haemophilus influenzae
ATCTAACGGTACGGATTCTCCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCG GCATAAGTAGTAAGTG CCGTGCCTCCGCCATT
AGTTACACTAGTGGGACACC
```

> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAACGTTACACG **AGTGTAACCGCCGAA**
C CATGTTGACACGAATTCTG
> Salmonella typhi
GGTCGGCTTAGACTAGTGTGACCAAAAAGCTTTTGCTGAA **GTTTCAGGGTAAGAAG** AACC
AGCTCCTAGTAAAAAGACTATTGTGACTGAAAAGCGCGTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATGCCGGCGCCGC
TGCTGGAACGCCTCGACCCCAGGGCTACACTA **GTTTAACCGGAACGCC** GCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC

a. Calculate motif profile(you complete this)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A																
C																
G																
T																

b. Find highest scoring instance given this motif profile in each sequence

> Escherichia coli
TTGATTCCCTGAATGCCCGCTTAGT **GTAACACTACTGTAAC** CGGCATTTTCTGCTTTTCC
TGCCGATATTTTTTCTTATCTACCTCACAAAGGTTAGCAATAACTGCTGGGAAAATTCCG
AGTTAGTCGTTATATTCTAT
> Haemophilus influenzae
ATCTAACGGTACGGATTCTCCAAAGGCCTATGGAATCTTGTAGAATATGAAACGTTCTAA
TAAATCATAAAGTTGGAGCAAACGCTCGGCATAAGTAGTAAGTGCCGTGCCTCCGCCATT
A **GTTACACTAGTGGGAC** ACC
> Vibrio cholerae
ATTTGTGGCGGTTTTCAAATGCTTGGAGAATGGGTACATGATCCGCTTGGCATTGAAGGT
GAGGCTGGCAGCAGCGAAGGTCTGGGGCTGTTTGAAC **GTTACACGAGTGTAAC** CGCCGAA
CCATGTTGACACGAATTCTG
> Salmonella typhi
GGTCGG **CTTAGACTAGTGTGAC** CAAAAAGCTTTTGCTGAAGTTTCAGGGTAAGAAGAACC
AGCTCCTAGTAAAAAGACTATTGTGACTGAAAAGCGCGTCAGCGCAAAGCCGACCGCACA
AAACGCACAAGGAGTTACAG
> Pseudomonas aeruginosa
ACGCGGCCAGGGTCTTCTCCTGCGAGATCATGCGCGGCGCGCCGCGCATGCCGGCGCCGC
TGCTGGAACGCCTCGACCCCAGG **GCTACACTAGTTTAAC** CGGAACGCCGCCAGTGGATCG
GCCTGCCCCAGCTATTGCTC

a. Calculate motif profile
   … keep going until convergence (no different instances are found)

Information Content

Suppose we find one or more motif models for a set of sequences. How do we know which model is better? We can use a metric called **information content** (also known as relative entropy or just entropy). Suppose we have the following motif model M, where each entry is a probability (frequency):

	1	2	3	4
A	A1	A2	A3	A4
C	C1	C2	C3	C4
G	G1	G2	G3	G4
T	T1	T2	T3	T4

Let Q_N be the background distribution (frequency) of nucleotide N.
The information content I of position 1 of model M is:
$I_1 = \sum [M_{N,1} \log_2 (M_{N,1} / Q_N)]$ for N ϵ {A, C, G, T}

Expanded out, this is:
$I_1 = A1 * \log_2(A1/Q_A) + C1 * \log_2(C1/Q_c) + G1 * \log_2(G1/Q_G) + T1 * \log_2(T1/Q_T)$

I_2, I_3, and I_4 can be calculated similarly.

The **total information content** is simply the sum of each I_i value. In the model above, this is:
$I = I_1 + I_2 + I_3 + I_4$

Hertz and Stormo (Finding motif profiles and motifs)

Input: Set S of sequences with a likely common motif, L the motif length, and D the number of profiles to retain at each step of the algorithm

Algorithm:

1. Create a singleton profile from one of the L-mers in one of the sequences.
2. Until each profile has k members where the size of S is k:
    a. For each of the D profiles retained, add each possible L-mer from sequences not already represented in the set of profiles
    b. Compute the information content of each profile
    c. Retain the D profiles with the highest information content

Example input:

```
Sequences S:
Seq1: ACTGA
Seq2: TAGCG
Seq3: CTTGC
```

L = 4
D = 1

Step 1:
Randomly choose ACTG as the singleton profile.

Profile:

	1	2	3	4
A	1	0	0	0
C	0	1	0	0
G	0	0	0	1
T	0	0	1	0

Step 2:
a and b. Add each possible 4-mer from sequences not represented in the profile

Profile with set {ACTG, TAGC}: //try adding substring TAGC from Seq2

	1	2	3	4
A	.5	.5	0	0
C	0	.5	0	.5
G	0	0	.5	.5
T	.5	0	.5	0

Info content: (the 0's will make the log undefined, so we'll assume these are positive values close to 0)
(We'll assume a background distribution of 25% for each nucleotide.)
= 4

Profile with set {ACTG, AGCG}: // try adding substring AGCG from Seq2

	1	2	3	4
A	1	0	0	0
C	0	.5	.5	0
G	0	.5	0	1
T	0	0	.5	0

Info content: 6

Profile with set {ACTG, CTTG}: //try adding substring CTTG from Seq3

	1	2	3	4
A	.5	0	0	0
C	.5	.5	0	0
G	0	0	0	1
T	0	.5	1	0

Info content: 6

Profile with set {ACTG, TTGC}: //try adding substring TTGC from Seq3

	1	2	3	4
A	.5	0	0	0
C	0	.5	0	.5
G	0	0	.5	.5
T	.5	.5	.5	0

Info content: 4

Step c: Keep the top 1 profile (arbitrarily choose set {ACTG, AGCG})

Step 2:

a and b: Add each possible 4-mer from sequences not represented in the profile

Profile with {ACTG, AGCG, CTTG}: //note: we add the first 4-mer from seq3

	1	2	3	4
A	.67	0	0	0
C	.33	.33	.33	0
G	0	.33	0	1
T	0	.33	.67	0

Info content: 4.57

Profile with {ACTG, AGCG, TTGC} //note: we add the second 4-mer from seq3

	1	2	3	4
A	.67	0	0	0
C	0	.33	.33	.33
G	0	.33	.33	.67
T	.33	.33	.33	0

Info content: 2.96

Done, since the profile includes 3 instances, one from each of the three sequences.

Thus, the algorithm found the best motif profile to be:

Profile with {ACTG, AGCG, CTTG}:

	1	2	3	4
A	.67	0	0	0
C	.33	.33	.33	0
G	0	.33	0	1
T	0	.33	.67	0

Info content: 4.57

Consensus sequence for this motif: ACTG or AGTG or ATTG

Markov Chains

Defn: Let S be a set of states (example: A, C, G, T). Let (X_0, X_1, …) be a sequence of random variables, each with sample space S. A k^{th} order Markov chain satisfies:

$Pr(X_t = x \mid X_0, X_1, X_2, \ldots, X_{t-1}) = Pr(X_t = x \mid X_{t-k}, X_{t-k+1}, \ldots, X_{t-1})$ for any t and any x in S. (the last k steps matter)

0^{th} order Markov chain: distribution of X_t is not dependent on previous state (just like frequencies in motif profiles)
1^{st} order Markov chain: distribution of X_t is dependent on just one previous state X_{t-1}
Etc…

Example:
Let's look at dinucleotide frequencies. Then the probability transition matrix M for the Markov chain is size 4 x 4 (4 states going to each of the other states).
$M_{r,s} = Pr(X_t = s \mid X_{t-1} = r)$

	A	C	G	T
A	.40	.30	.10	.20
C	.30	.25	.35	.10
G	.10	.30	.20	.40
T	.25	.25	.25	.25

row A, col A: means the probability that the next nucleotide is A given A
row A, col C: means the probability that the next nucleotide is C given A
etc…

So, if this is the Markov chain model M, then the probability of sequence $s_1s_2s_3…s_t$ being generated by this model is:
$Pr(s_1s_2s_3…s_t) = Pr(s_1)*M_{s1,\, s2}*M_{s2,s3}*M_{s3,s4}*…*M_{st-1,st}$

What is the probability that "ACTCGGA" is generated by this model M? Assume the probability of the first nucleotide is 25% for each of ACGT.
Pr(ACTCGGA) = .25 *.30 * .10 * .25 * .35 * .20 * .10

Course Notes:

CS423/BIO423 Review Sheet – Midterm Exam #1

Content: Exam 1 will cover topics below (or however far we get through the day before the exam). Material will be drawn from labs 1 - 3, prelabs, lectures, and posted moodle resources.

Procedure: The exam will start promptly at the beginning of class. Please arrive to class on time. You may use one sheet of 8/5" x 11" paper (both sides) during the exam. Other than your sheet of notes, the exam is closed-book, closed-calculator, closed-computer, and closed-notes. All computations will be simple enough for you to do by hand. Tammy will provide the RNA-> amino acid translation table for you during the exam.

Topics: This study guide is not a contract – in other words, the exam may not cover every topic listed below and there may be topics that we covered in class that are not explicitly listed. You may want to review the lecture notes, found on Moodle and review your lab assignments.

- Cellular and Molecular Biology (including lab 2)
    - Genomes
    - Mutations
    - DNA (bases: ACTG)
    - RNA (bases: ACUG)
    - Amino Acids (20 different ones) and Proteins
    - Central Dogma
        - Transcription (DNA -> RNA)
            - If there are introns, these are spliced out before translation
        - Translation (RNA -> proteins)
    - Regulatory regions, transcription start/stop sites
    - DNA replication; DNA polymerase
    - Gene expression
    - Eukaryotes (genes have introns/exons, DNA is double helix), ex: yeast
    - Prokaryotes (genes are all exons, circular DNA loop), ex: E.coli
    - Alternative splicing
- Algorithms
    - Types: recursive, exhaustive, branch and bound, divide and conquer, greedy
    - Example: Towers of Hanoi is recursive
    - Example: Merge sort is divide and conquer and recursive
    - Example: Linear search is exhaustive
    - Example: Binary search is branch and bound and recursive
- Algorithm analysis
    - Big O
    - Determine running time of algorithms
        - Examples: insertion sort ($O(n^2)$), merge sort ($O(n \lg n)$)
- Python (lab 1)
    - Variables, strings
    - Printing
    - Conditions (if, else, elif), loops (for, while), functions (def)

    - Files (input/output)
    - Random numbers
    - Booleans and comparison, logical operators (or, and, not)
    - Lists (like arrays)
- Motifs (lab 3)
    - DNA polymerase might bind to a motif, transcription start/stop sites, promoter signals, intron splice sites
    - Finding instances of known sites:
        - Use profiles (table indicating frequency of nucleotides at each position in motif)
        - Calculate likelihood ratios (Prob that it is a motif / Prob that it is background dist)
        - Log likelihood ratios (Take log of likelihood, so entries can be added in table)

Subject to change: refer to moodle for any changes to exam guide

CS423/BIO423 Review Sheet – Midterm Exam #2

Content: Exam 1 will cover topics from lecture and labs since exam 1 (or however far we get through the day before the exam). Material will be drawn from labs 4 - 6, prelabs, lectures, and posted moodle resources.

Procedure: The exam will start promptly at the beginning of class. Please arrive to class on time. You may use one sheet of 8/5" x 11" paper (both sides) and a regular calculator during the exam. Other than your sheet of notes, the exam is closed-book, closed-computer, and closed-notes. All computations will be simple enough for you to do with a stand-alone calculator. Tammy will provide the RNA-> amino acid translation table for you during the exam if needed.

Topics: This study guide is not a contract – in other words, the exam may not cover every topic listed below and there may be topics that we covered in class that are not explicitly listed. You may want to review the lecture notes, review Moodle, and review your lab assignments.

- Mutations/Substitutions/Evolution (lab 4)
    - Molecular clock
    - Sequence similarity
        - Hamming distance
        - p distance
        - Jukes Cantor correction estimate (includes non-observed mutations)
    - PAM and BLOSUM matrices for amino acid substitution rates
- String alignment (lab 5)
    - Determine how similar two sequences are given deletions, insertions, and substitutions
    - Manhattan problem (example of dynamic programming)
    - optimal global alignment (dynamic programming) – find best alignment for two entire strings
    - optimal local alignment (dynamic programming) – find best alignment among any substring of each of two input strings
    - longest common subsequence (dynamic programming) – find longest common subsequence in two strings
    - Affine gap penalty (dynamic programming with four tables)
        - penalty for starting a gap, penalty for extending gap
    - All these run in time O(mn) where m and n are the lengths of the two input strings
- BLAST (Basic Local Alignment Search Tool) (lab 6)
    - For proteins: default word size is 3 amino acids
    - E-values
    - bit scores
    - types: blastp (protein to protein), blastn (DNA to DNA)
- Multiple sequence alignment
    - Aligning more than 2 sequences at once
    - One algorithm is extension of dynamic programming, but it gets too slow with several sequences

- Heuristics: build up multiple alignment by aligning two strings, getting its alignment and then adding another string, etc
- Clustal omega (tool for multiple sequence alignment and generating trees)

Subject to change: refer to moodle for any changes to exam guide

CS423/BIO423 Review Sheet – Midterm Exam #3

Content: Exam 3 will cover topics from lecture and labs since exam 2. Material will be drawn from labs 7 - 10, lectures, moodle resources, and the textbook.

Procedure: The exam will start promptly at the beginning of class. Please arrive to class on time. You may use one sheet of 8/5" x 11" paper (both sides) and a regular calculator during the exam. Other than your sheet of notes, the exam is closed-book, closed-computer, and closed-notes. All computations will be simple enough for you to do with a stand-alone calculator. Tammy will provide the RNA-> amino acid translation table for you during the exam if needed.

Topics: This study guide is not a contract – in other words, the exam may not cover every topic listed below and there may be topics that we covered in class that are not explicitly listed. You may want to review the lecture notes, found on Moodle and review your lab assignments.

- Phylogenetic Trees (also known as Evolutionary Trees, Guide Trees, lab 7)
    - Terminology
        - Clade (node and all descendants)
        - Rooted (tree has a root, shows ancestry)
        - Unrooted (tree has no root, just shows kinship among organisms)
        - Cladogram (tree is unscaled, branch lengths do not show distance)
        - Phylogram (tree is scaled, branch lengths show distance)
    - UPGMA – given distance matrix, creates rooted tree
        - Use mean distance as distance between clusters
        - Merge closest clusters at each step
        - If matrix is a distance matrix (where lower numbers are better), then branch lengths can be calculated as the distance between the clusters / 2 for the total length of the branch to the leaves
    - Neighbor Joining – given distance matrix, creates unrooted tree
        - Start with star topology
        - Minimizes intra-cluster distance and maximizes distance to other clusters
        - Join leaves by creating internal node in tree
    - Both UPGMA and Neighbor Joining are greedy algorithms and may not produce the optimal tree in terms of minimizing tree branch lengths. But both are fast and run in time $O(N^3)$.
    - The number of possible trees for N organisms grows really fast, so finding optimal tree is only realistic for $N < 8$.
    - Maximum Parsimony – given sequences and their alignment, creates unrooted tree
        - Generates all possible trees for all informative sites, actually uses the alignment to construct tree (not just distance scores or alignment scores)
        - Choose tree that has the fewest changes (best explains the differences among the sequences) over all sites
        - Note that this method uses the sequence info and not the distance info (as in UPGMA and Neighbor Joining)
- Gene Expression

- DNA Microarrays (may not be included)
    - Probes (complementary DNA) on array / Targets (mRNA) washed over array
    - Uses:
        - sick / healthy cells
        - compare species
        - disease detection
        - cell life cycle
        - related cell functional pathways
- RNA-seq (lab 8)
    - Comparative gene expression
    - Counts of expressed mRNA
- Clustering (lab 9)
    - k-means – given microrarray data, produce k clusters
        - randomly select initial cluster assignments, calculate cluster center and reassign
        - may not converge to same set of clusters
        - usually fast running time
    - QT – given microarray data, produce clusters with diameter at most d
        - consider maximum clusters for each item, choose largest cluster and remove items from chosen cluster for next iteration
        - always produces same answer
        - greedy, so may not produce optimal clusters
        - usually fast running time
    - Hierarchical clustering – given microarray data, produce a tree
        - Pre-process step includes taking microarray data and generating N x N distance matrix
        - Runs just like UPGMA and distance metric can be chosen (mean distance, closest points between clusters, furthest points between clusters, etc)
        - Greedy, so may not produce optimal tree
        - Up to experimenter to determine where to slice branches of tree to produce independent clusters
        - $O(N^3)$
    - Cluster 3.0 and Java TreeView (tools for clustering and viewing clusters)
- Motifs (lab 10)
    - Conserved short regions of DNA, generally thought to be involved in gene expression
    - Expectation Maximization
        - Given sequences, randomly sample sequences to create profile, then use profile to find best instances (until no changes are made)
    - Gibbs Motif Sampler (on-line tool for lab 10)
    - Hertz-Stormo Algorithm
        - Keep adding best l-mer from sequences not already represented in profile
- Markov chains
    - Example: dinucleotide frequencies (frequency of nucleotide at position N depends on nucleotide at position N-1)

Subject to change: refer to moodle for any changes to exam guide