CS 68: BIOINFORMATICS

Prof. Sara Mathieson
Swarthmore College
Spring 2018

Outline: Mar 23

- The Coalescent
- Measures of sequence diversity
- Midterm feedback

Notes:

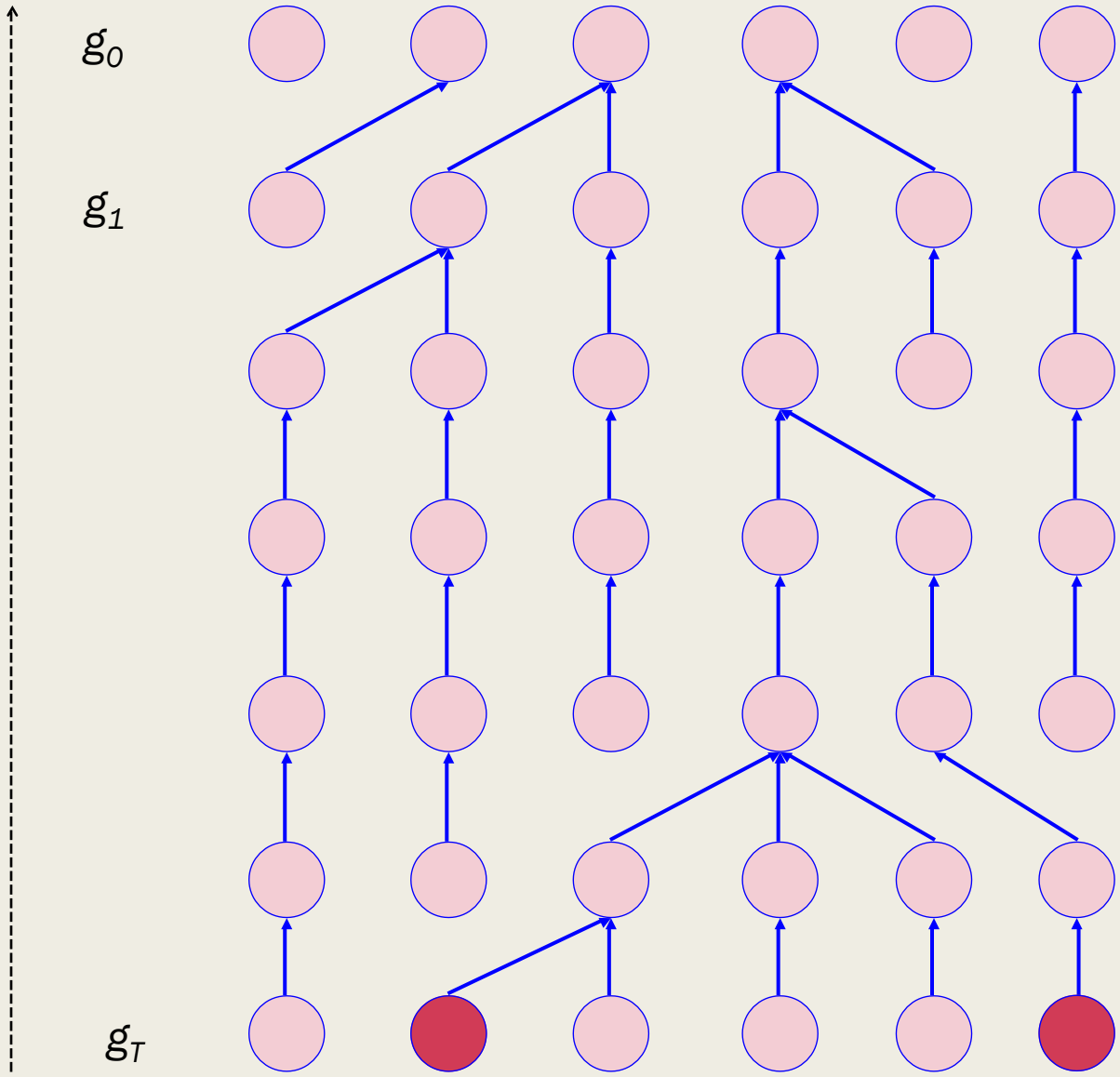
- Lab 6 due Wednesday
- I will be around after class and most of this afternoon

The Coalescent

Back to the Wright-Fisher Model

Generations back in time

Question: how long will it take two randomly chosen individuals to coalesce (i.e. find a common ancestor?)



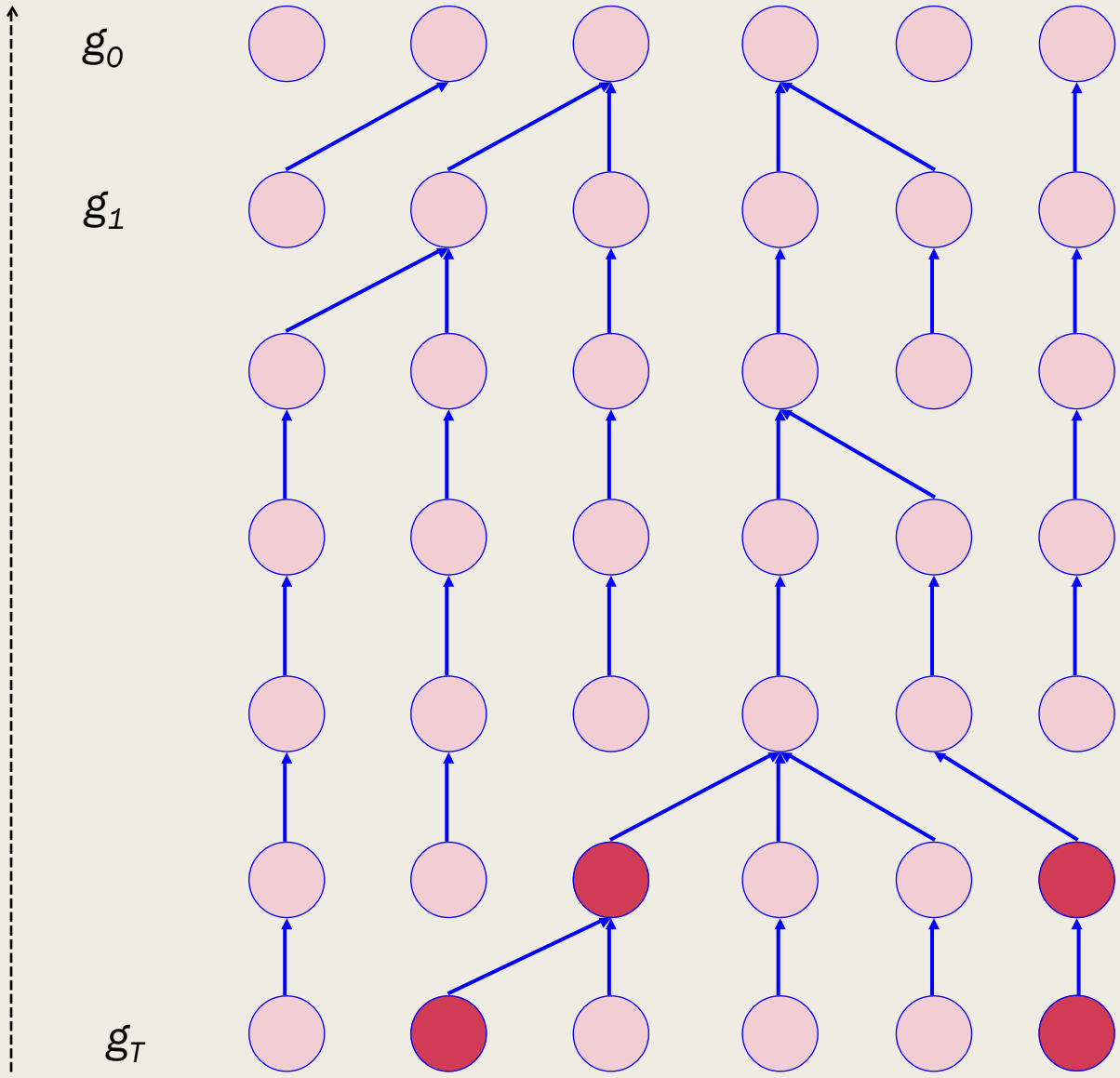
Constant population size: 2N

Back to the Wright-Fisher Model

Generations back in time

Question: how long will it take two randomly chosen individuals to coalesce (i.e. find a common ancestor?)

Probability they don't choose the same parent in the previous generation: 1-1/(2N)



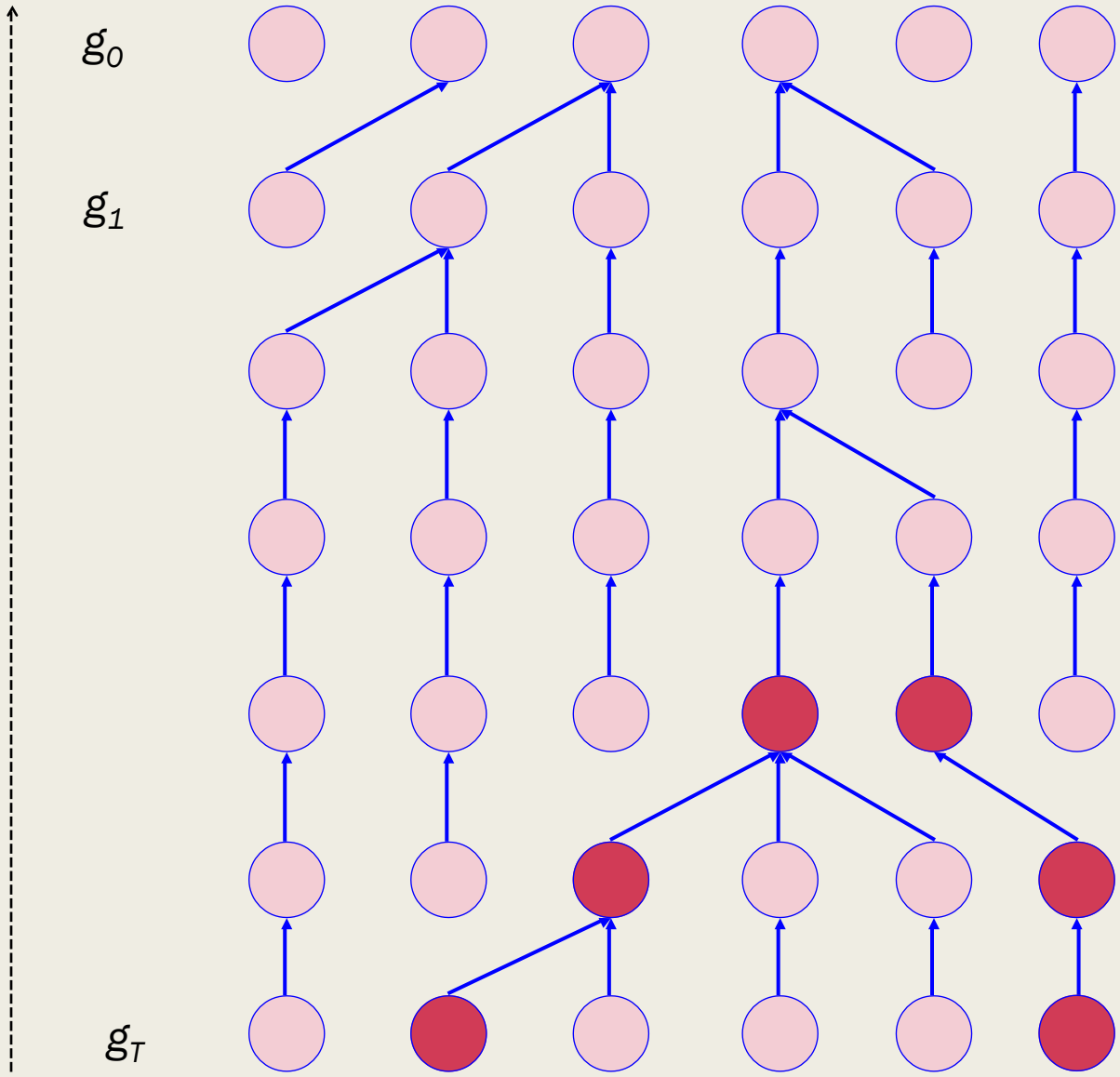
Constant population size: 2N

Back to the Wright-Fisher Model

Generations back in time

Question: how long will it take two randomly chosen individuals to coalesce (i.e. find a common ancestor?)

Probability they don't choose the same parent in the previous generation: 1-1/(2N)



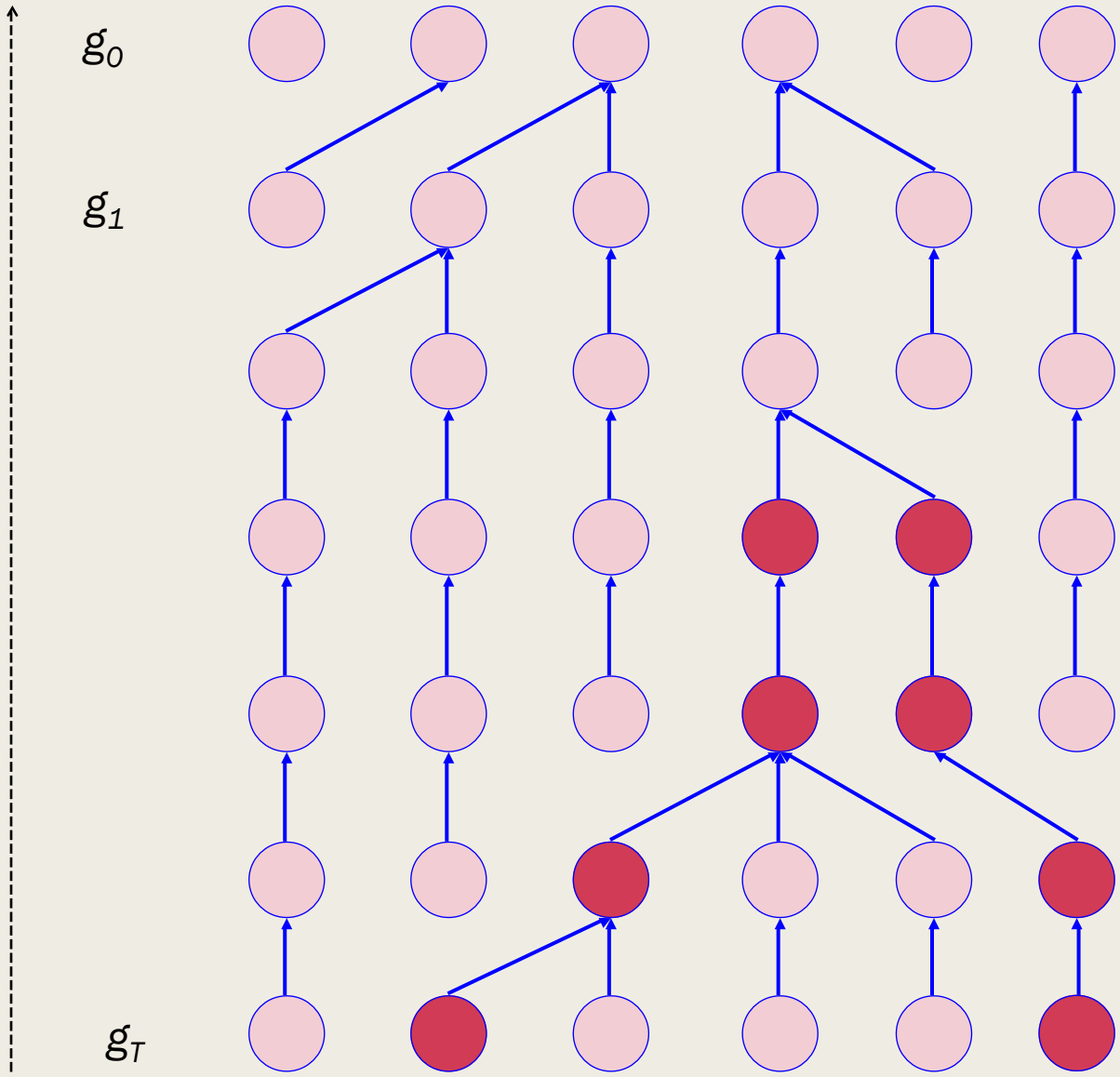
Constant population size: 2N

Back to the Wright-Fisher Model

Generations back in time

Question: how long will it take two randomly chosen individuals to coalesce (i.e. find a common ancestor?)

Probability they don't choose the same parent in the previous generation: 1-1/(2N)



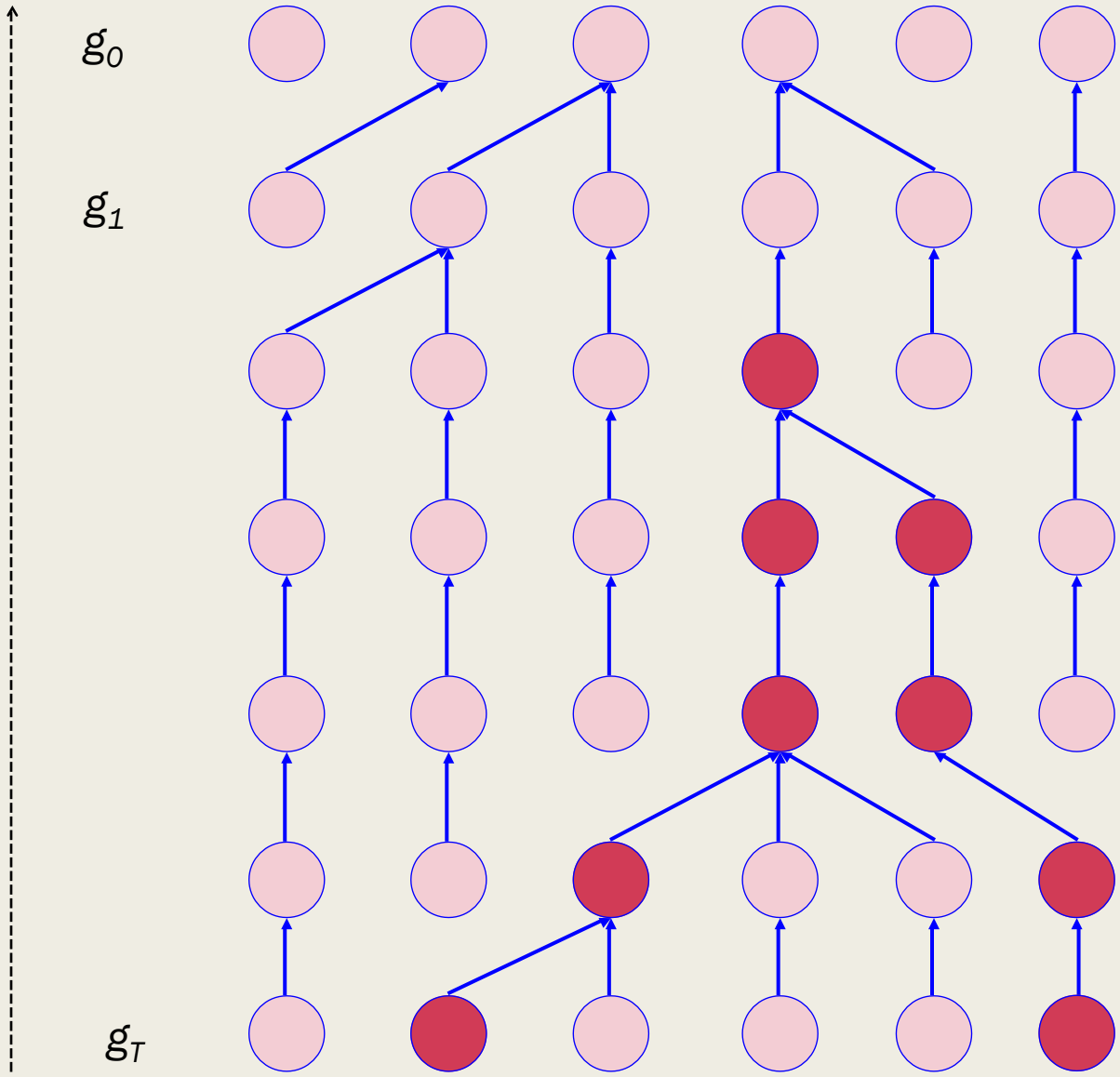
Constant population size: 2N

Back to the Wright-Fisher Model

Generations back in time

Question: how long will it take two randomly chosen individuals to coalesce (i.e. find a common ancestor?)

Probability they do choose the same parent: 1/(2N)



Constant population size: 2N

$$P_c(g) = \underbrace{\left(1-\frac{1}{2N}\right)^{g-1}\frac{1}{2N}}_{\text{geometric distribution}}$$

coalesce

generations

→ discrete probability distribution

↳ difficult to work with.

Idea: rescale time to be continuous

let $2N \to \infty$ (pop size)

[n will be our sample size]

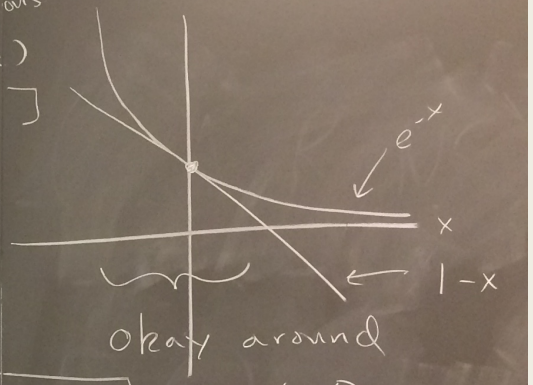
Recall $e^{-x} \approx 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$

(around $x=0$)

$$P_c(g) \approx e^{-\frac{1}{2N}g} \cdot \frac{1}{2N} \longrightarrow \boxed{P_{T_2}(t) = e^{-t}}$$

$\binom{2}{2}$

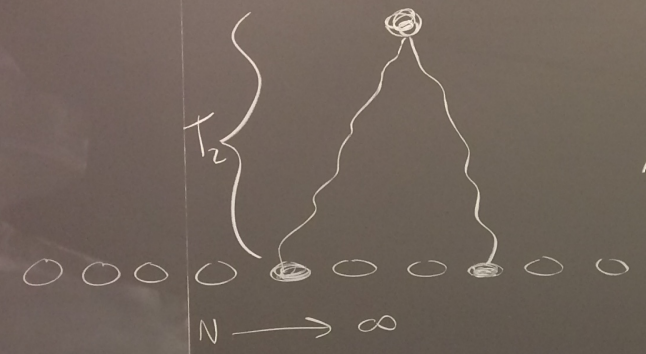
$\binom{2}{2}$



okay around $x=0$

$\int_0^\infty f =$

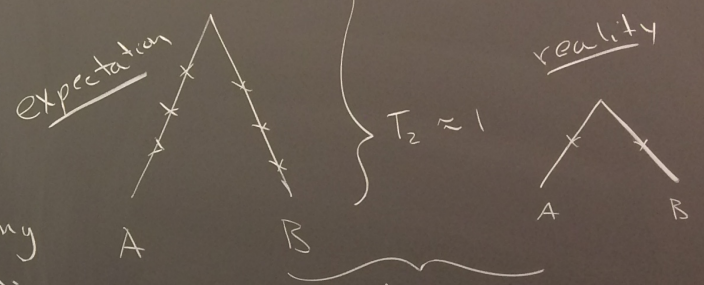
t coalescent units = $2N$ generations



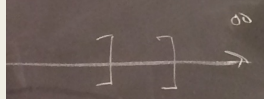
every $2N$ generation $= 1$ coalescent unit

$$E[T_2] = \int_0^\infty t e^{-t}\, dt = 1$$

$\Rightarrow$ expected time to coal. of 2 lineages is $\underline{\underline{2N}}$ generation.



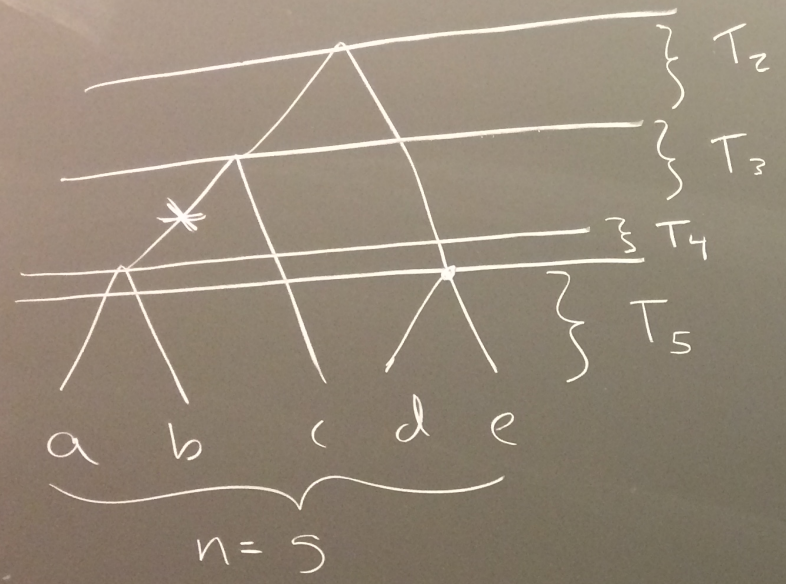
① N is wrong
② selection.



$$\lim_{p \to 0} \int_{F \cap U_N} f_p$$

$$t e^{-t} \Big|_0^\infty$$

$$P_{T_i}(t) = \binom{i}{2} e^{-\binom{i}{2}t}, \quad E[T_i] = \frac{1}{\binom{i}{2}}$$



$$E[T_5] = \frac{1}{\binom{5}{2}} = \frac{1}{10}$$

The Coalescent

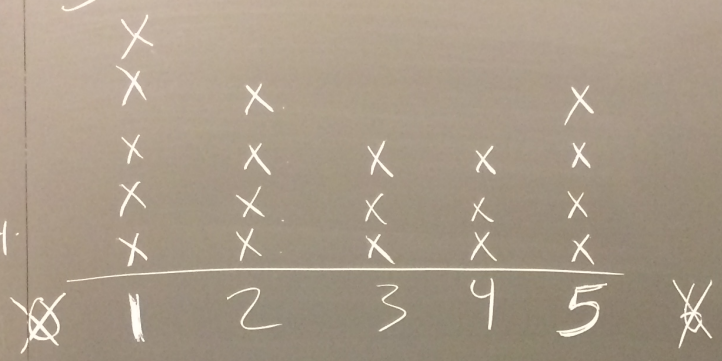
Back to measures of sequence diversity

S = # segregating sites

π = pairwise heterozygosity

SFS

$n=6$



$\xi_1 = 5, \xi_2 = 4, 3, 3, 4$

$O(n)$ dims

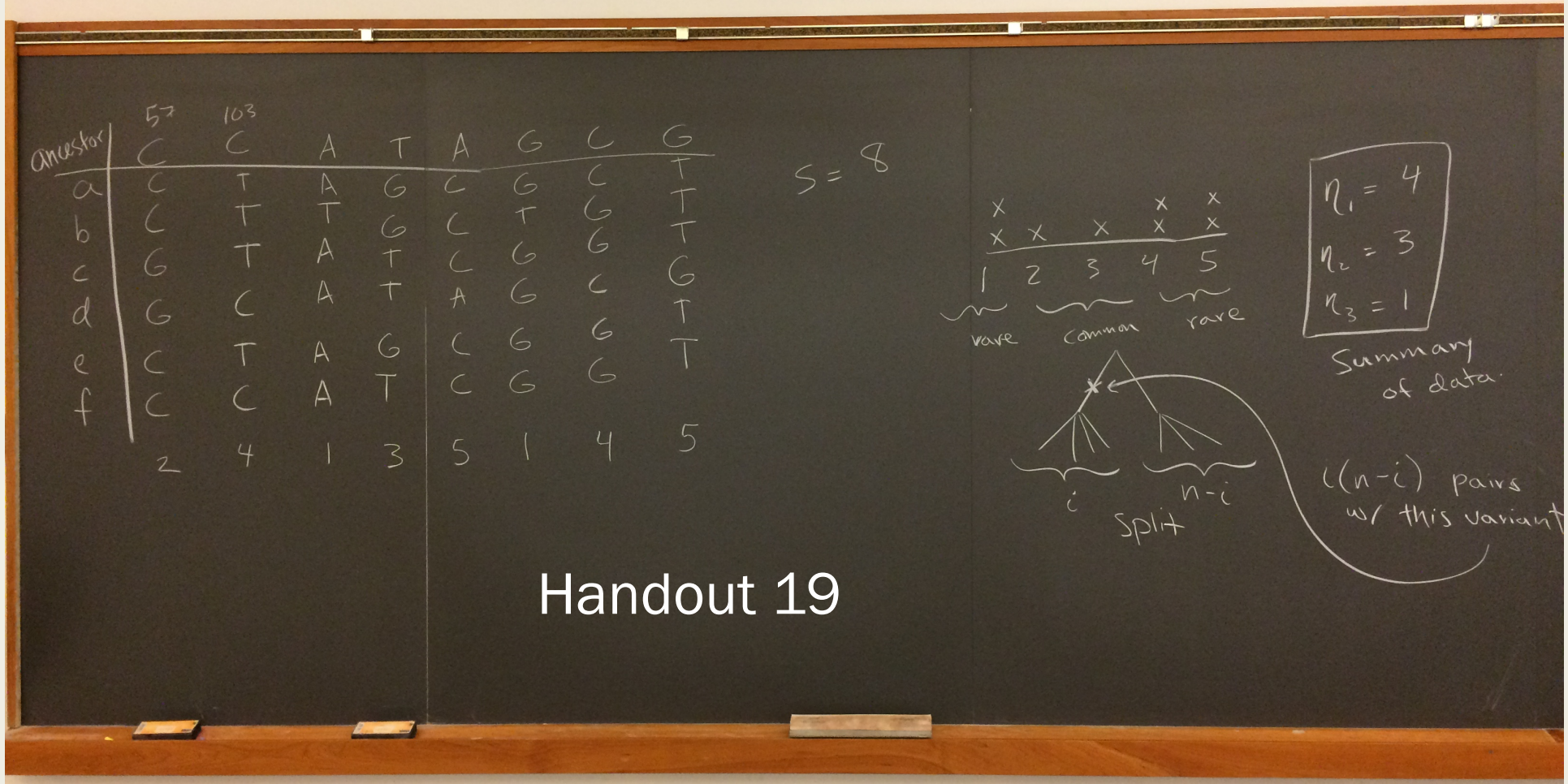
folded SFS

don't know ancestor

$\eta_1 = 9, \eta_2 = 7, \eta_3 = 3$

1 2 3

$\frac{n}{2}$ dims


57 103
ancestor C C A T A G C G
a C T A G C G C T
b C T T G C T G T
c G T A T C G G T
d G C A T A G C G
e C T A G C G G T
f C C A T C G G T
2 4 1 3 5 1 4 5
S = 8
1 2 3 4 5
rare common rare
n1 = 4
n2 = 3
n3 = 1
Summary of data.
i
n-i
Split
i(n-i) pairs w/ this variant

Handout 19

Midterm feedback

Midterm letter grades

A	85-100
B	65-84*
C	50-64
D	35-49

- Average: 72
- For +/- grades, every 5 points
- *80-84: A-/B+

Part 1: short answer

(a) Say I want to analyze many individuals from the same species. Order the following steps (i.e. 1 happens first, 2 second, etc).

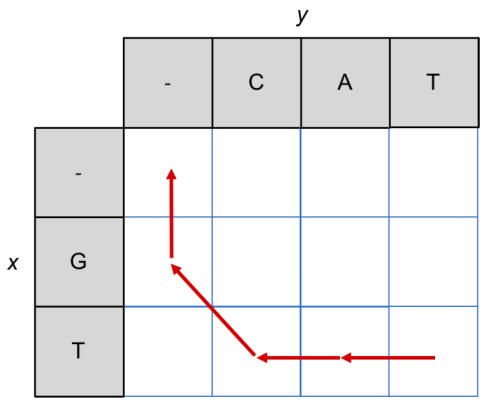
_2_ Sequence reads from several individuals and map them to a reference
_4_ Build an evolutionary tree for individuals in the population
_1_ Perform genome assembly on a set of reads from one individual
_3_ Perform variant calling to determine SNPs in the population

(b) In which of the scenarios below would pairwise sequence alignment via dynamic programming be a good choice? Check all that apply.

- ☑ Comparing the output of genome assembly (contigs) to a known reference
- ☐ Comparing many short reads to a long reference genome (Use BWT and FM-Index)
- ☑ Comparing two protein sequences with potentially similar functions
- ☑ Comparing two distantly related species in preparation for building a phylogenetic tree

Part 1

(c) What alignment between x and y does the following traceback represent?




```
x: GT--
y: -CAT
```

(d) Let $S(i, j)$ be the best global alignment up through x_i and y_j. If I want to disallow internal (i.e. not leading/trailing) gaps in x, which part of the recurrence relation below should I remove? (you can just cross it out below)

$$S(i,j) = \max \begin{cases} S(i-1,j) + g \\ S(i-1,j-1) + m(x_i, y_j) \\ \cancel{S(i,j-1) + g} \end{cases}$$

(e) Which of the following RNA transcripts (output of transcription), could never be valid? Cross them out below. Let our start codon be `AUG` and stop codons be `UAG, UAA, UGA`.

`CUGGACUGU`

`UCUAGCCAG`

~~`CCGGA`~~

~~`ACUUAAGAC`~~

Part 2: assembly

(a) For a read of length L, what is the maximum number of unique, overlapping k-mers?

$L-k+1$ (technically $\min\{L-k+1, 4^k\}$ for DNA, but for sufficiently large k this will almost never be a factor)

(b) For a genome of length G, what is the maximum number of unique, overlapping k-mers?

$G-k+1$ (same situation as part (a))

(c) Given n reads and your responses above, *on average*, what is the number of reads that share a k-mer? (i.e. what is the average length of the values in our dictionary?) Call this m.

$$m = \frac{n \cdot (L-k+1)}{G-k+1} = O\left(\frac{nL}{G}\right)$$

This is because there are $O(nL)$ total k-mers in all the reads, and $O(G)$ unique k-mers in the genome (i.e. the keys or "bins" of our dictionary).

(d) Assuming we can hash k-mers in $O(1)$, what is the big-O runtime of creating this dictionary?

$O(nL)$, the total number of k-mers in all the reads.

Part 2: assembly

(e) If two reads share a k-mer, does that mean there should be an edge between them in an *overlap* graph? Why or why not?

No. Regardless of the overlap threshold T, the shared k-mer could appear in the *middle* of one or both of the reads. (This is possible unless k is very high, i.e. $k > L - 2$.)

(f) Using this method, what is the total big-O runtime of creating an overlap graph (in terms of n, G, L, and m)? Assume that the runtime of detecting the maximal overlap between two reads is $O(L^2)$, using either dynamic programming or some other method. (There are ways to speed this up using the locations of the shared k-mer, but we'll ignore those for now).

Building the dictionary is $O(nL)$. We have on average m reads per k-mer, which is $O(m^2)$ *pairs* per k-mer. It takes $O(L^2)$ work to detect an overlap between each pair. Given $O(G)$, total k-mers, the complete runtime is:

$$O(nL) + O(Gm^2L^2)$$

(g) If $G = O(n)$ (not uncommon), how could you simplify your runtime from the previous part? How does this compare to the runtime of the more standard method for building an overlap graph discussed in class?

Simplifying the above expression, we get $O(nL^4)$. Our standard runtime is $O(n^2L^2)$. Given that n is in the millions or billions and L is 50-100, this is a dramatic improvement.

Part 3: alignment

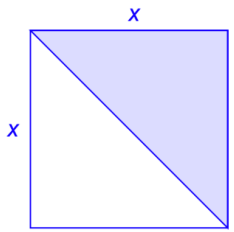
High level idea: finding repeats in X is a variation on local alignment of X with itself. We will use a square DP table with X on both the rows and the columns, but we will only fill in the upper right triangle of the matrix since it is symmetric (and we want to exclude the diagonal which represents aligning the same region of X with itself). Let $n = \text{len}(X)$.

Initialization: $S(0, j) = 0$, for $j = 0, 1, \cdots, n$

Recursion: For all i, j such that $i < j$:

$$S(i,j) = \begin{cases} S(i-1, j-1) + 1 & \text{if } \; x_i = x_j \\ 0 & \text{otherwise} \end{cases}$$

Traceback: Find the max value in the table and traceback from all cells that are equal to the max until we hit a zero.



Part 3: alignment

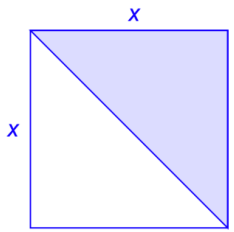
High level idea: finding repeats in X is a variation on local alignment of X with itself. We will use a square DP table with X on both the rows and the columns, but we will only fill in the upper right triangle of the matrix since it is symmetric (and we want to exclude the diagonal which represents aligning the same region of X with itself). Let $n = \text{len}(X)$.

Initialization: $S(0, j) = 0$, for $j = 0, 1, \cdots, n$

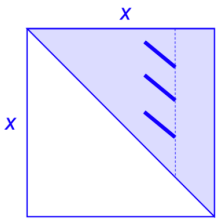
Recursion: For all i, j such that $i < j$:

$$S(i,j) = \begin{cases} S(i-1, j-1) + 1 & \text{if } \; x_i = x_j \\ 0 & \text{otherwise} \end{cases}$$

Traceback: Find the max value in the table and traceback from all cells that are equal to the max until we hit a zero.



Repeats that occur more than twice will end at the same column value in our DP table. If we record the end column when tracing back, we can cluster repeats based on this value to find the number of times each repeat occurs.



Part 4

c	M[c]
\$	1
a	2
b	11

i	occ(\$)	occ(a)	occ(b)	A
1	0	0	1	21
2	0	0	2	2
3	0	0	3	17
4	0	0	4	9
5	0	0	5	15
6	0	0	6	13
7	0	1	6	3
8	0	2	6	18
9	0	3	6	10
10	0	3	7	5
11	0	3	8	20
12	1	3	8	1
13	1	4	8	16
14	1	4	9	8
15	1	5	9	14
16	1	5	10	12
17	1	6	10	4
18	1	7	10	19
19	1	7	11	7
20	1	8	11	11
21	1	9	11	6

1.

$$\text{sp}(a) = M[a] = 2$$
$$\text{ep}(a) = M[b] - 1 = 10$$

2.

$$\begin{aligned}\text{sp}(ba) &= M[b] + \text{occ}(b, sp(a) - 1)\\ &= 11 + 1\\ &= 12\\ \text{ep}(ba) &= M[b] - \text{occ}(b, ep(a)) - 1\\ &= 11 + 7 - 1\\ &= 17\end{aligned}$$

3.

$$\begin{aligned}\text{sp}(aba) &= M[a] + \text{occ}(a, sp(ba) - 1)\\ &= 2 + 3\\ &= 5\\ \text{ep}(aba) &= M[b] - \text{occ}(a, ep(ba)) - 1\\ &= 2 + 6 - 1\\ &= 7\end{aligned}$$

Finally we can look up $A[5]$,$A[6]$,and $A[7]$ to find that $P = aba$ starts at indices 15, 13, and 3.

Part 4

Input: a single reference sequence S and a set of reads $R = \{r_1, r_2, \cdots, r_n\}$. Each read is of length L with (potentially) a difference from the reference at index $L/2$ (assume L is even). I would recommend using 0-based indexing for this problem, but if you prefer 1-based write that at the top. For example, say read $r = \text{GACT}$ of length $L = 4$. It could match to a region of the reference that is GATT (difference at index 2).

Output: the location(s) of each read in the original string. Note that it is possible for a read to match to the reference both exactly and in-exactly. You don't need to write code (or even pseudocode), but be clear about how you will obtain the indices of each read in the original string.

We cannot "skip over" the mismatch without branching the search.

Use the BWT and FM-Index to find the locations of the first half of the read, $r[0 : L/2]$. Let this set of start locations be A. Then find all the locations of the second half of the read after the mismatch, $r[L/2 + 1 :]$. Let this set of start locations be B. For each pair of locations in $a \in A$ and $b \in B$, if $b = a + L/2 + 1$, retain a as a start location of the original read r. No matter what the mismatch is (if there is one), this will return the locations where the mismatch is in the center.

Assuming each read maps to a small number of locations, this algorithm will still run in $O(L)$ for each read, so $O(nL)$ for n reads.

Now say we are in the same situation with at most one mismatch, but now we don't know where it is in each read. How could you use our same, unmodified pattern matching method to find out where the mismatch is? What is the runtime of your algorithm in terms of n, L, and G?

Does not depend on G!

(Note: there could be many ways of making this work, with various runtimes.) One idea is to run the above algorithm for every possible "split" of the read around a mismatch. There are $O(L)$ such splits and each will take $O(L)$ time, for an overall runtime of $O(nL^2)$ for all the reads. This is still much better than dynamic programming which depends on the length of the reference G.