

Sequence Alignment & Computational Thinking

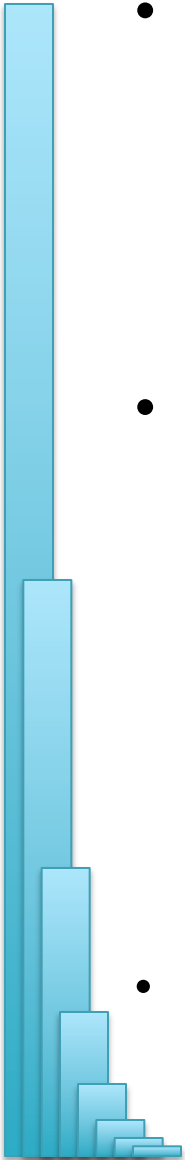
Michael Schatz

Bioinformatics Lecture 2
Undergraduate Research Program 2011



Recap

- Sequence assays used for many important and interesting ways
 - Variation Discovery: How do the reads map to the reference?
 - Expression Analysis: How many reads map to the reference?
 - Binding Analysis: Where do the reads map to the reference?
 - Plus many others!
- There is a galaxy of tools available to help analyze these data
 - Galaxy: point and click analysis
 - Bowtie: Read Mapping
 - Tophat: Spliced read mapping
 - Cufflinks: Transcript Assembly
 - Cuffdiff: Differential Expression
 - MACS: Peak Analysis
 - ...
- Questions for today
 - How does Bowtie map a billion reads to the human genome?
 - How can you think computationally about a problem?



Computational Biology

"Computer science is no more about computers than astronomy is about telescopes."
Edsger Dijkstra

- Computer Science = Science of Computation
 - Solving problems, designing & building systems
 - Computers are very, very dumb, but we can instruct them
 - Build complex systems out of simple components
 - They will perfectly execute instructions forever
- CompBio = Thinking Computationally about Biology
 - Processing: Make more powerful instruments, analyze results
 - Designing & Understanding: protocols, procedures, systems
- Sequence Alignment
 1. Brute Force
 2. Suffix Arrays
 3. Inexact Alignment
 4. Bowtie
- Computational Thinking
 1. Algorithm
 2. Data structure
 3. Computational Analysis
 4. Computational Modeling

Searching for GATTACA

- Where is GATTACA in the human genome?
- Strategy I: Brute Force

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

No match at offset 1

Searching for GATTACA

- Where is GATTACA in the human genome?
- Strategy I: Brute Force

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

Match at offset 2

Searching for GATTACA

- Where is GATTACA in the human genome?
- Strategy I: Brute Force

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

No match at offset 3...

Searching for GATTACA

- Where is GATTACA in the human genome?
- Strategy I: Brute Force

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

No match at offset 9 <- Checking each possible position takes time

Brute Force Analysis



- Brute Force:
 - At every possible offset in the genome:
 - Do all of the characters of the query match?
- Analysis
 - Simple, easy to understand
 - Genome length = n [3B]
 - Query length = m [7]
 - Comparisons: $(n-m+1) * m$ [21B]
- Overall runtime: $O(nm)$
 - [How long would it take if we double the genome size, read length?]
 - [How long would it take if we double both?]

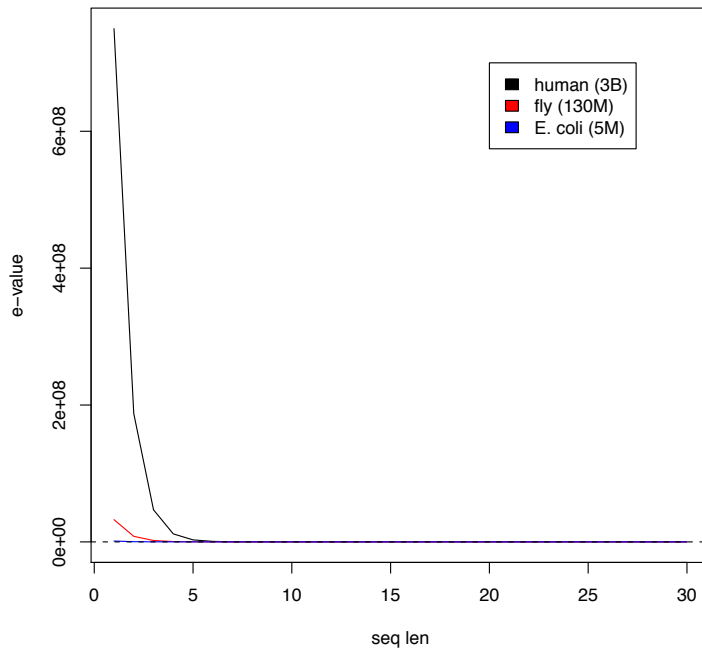
Expected Occurrences

The expected number of occurrences (e-value) of a given sequence in a genome depends on the length of the genome and inversely on the length of the sequence

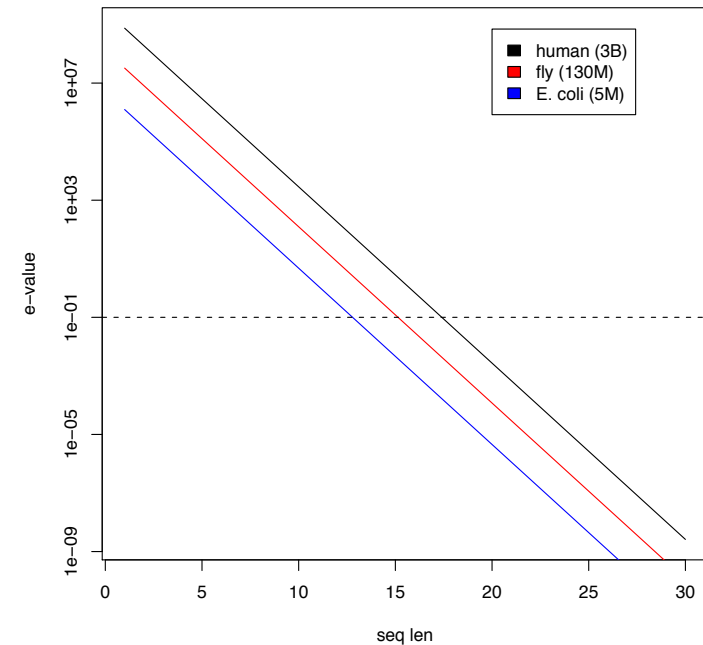
- 1 in 4 bases are G, 1 in 16 positions are GA, 1 in 64 positions are GAT, ...
- 1 in 16,384 should be GATTACA
- $E = n / (4^m)$

[183,105 expected occurrences]
[How long do the reads need to be for a significant match?]

Value and sequence length
cutoff 0.1



E-value and sequence length
cutoff 0.1



Brute Force Reflections

Why check every position?

- GATTACA can't possibly start at position 15

[WHY?]

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

- Improve runtime to $O(n + m)$

[3B + 7]

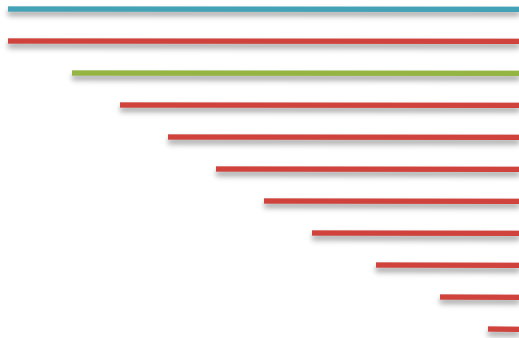
- If we double both, it just takes twice as long
- Knuth-Morris-Pratt, 1977
- Boyer-Moyer, 1977, 1991

- For one-off scans, this is the best we can do (optimal performance)

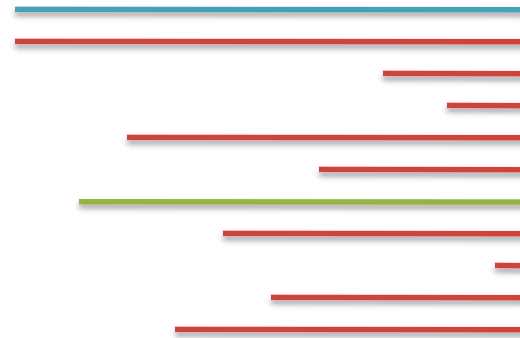
- We have to read every character of the genome, and every character of the query
- For short queries, runtime is dominated by the length of the genome

Suffix Arrays: Searching the Phone Book

- What if we need to check many queries?
 - We don't need to check every page of the phone book to find 'Schatz'
 - Sorting alphabetically lets us immediately skip 96% (25/26) of the book *without any loss in accuracy*
- Sorting the genome: Suffix Array (Manber & Myers, 1991)
 - Sort every suffix of the genome



Split into n suffixes



Sort suffixes alphabetically

[Challenge Question: How else could we split the genome?]

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - Lo = 1; Hi = 15;

Lo
→

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$

Lo
→

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$
=> Higher: $Lo = Mid + 1$

Lo
→

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$
=> Higher: $Lo = Mid + 1$
 - $Lo = 9; Hi = 15;$

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo
→

Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$
=> Higher: $Lo = Mid + 1$
 - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
 - $Middle = Suffix[12] = TACC$

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo
→

Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$
=> Higher: $Lo = Mid + 1$
 - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
 - $Middle = Suffix[12] = TACC$
=> Lower: $Hi = Mid - 1$
 - $Lo = 9; Hi = 11;$

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo
→

Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$
=> Higher: $Lo = Mid + 1$
 - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
 - $Middle = Suffix[12] = TACC$
=> Lower: $Hi = Mid - 1$
 - $Lo = 9; Hi = 11; Mid = (9+11)/2 = 10$
 - $Middle = Suffix[10] = GATTACC$

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo
→

Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$
 \Rightarrow Higher: $Lo = Mid + 1$
 - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
 - $Middle = Suffix[12] = TACC$
 \Rightarrow Lower: $Hi = Mid - 1$
 - $Lo = 9; Hi = 11; Mid = (9+11)/2 = 10$
 - $Middle = Suffix[10] = GATTACC$
 \Rightarrow Lower: $Hi = Mid - 1$
 - $Lo = 9; Hi = 9;$

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo
Hi
→

Searching the Index

- Strategy 2: Binary search
 - Compare to the middle, refine as higher or lower
- Searching for GATTACA
 - $Lo = 1; Hi = 15; Mid = (1+15)/2 = 8$
 - $Middle = Suffix[8] = CC$
=> Higher: $Lo = Mid + 1$
 - $Lo = 9; Hi = 15; Mid = (9+15)/2 = 12$
 - $Middle = Suffix[12] = TACC$
=> Lower: $Hi = Mid - 1$
 - $Lo = 9; Hi = 11; Mid = (9+11)/2 = 10$
 - $Middle = Suffix[10] = GATTACC$
=> Lower: $Hi = Mid - 1$
 - $Lo = 9; Hi = 9; Mid = (9+9)/2 = 9$
 - $Middle = Suffix[9] = GATTACA...$
=> Match at position 2!

#	Sequence	Pos
1	ACAGATTACC...	6
2	ACC...	13
3	AGATTACC...	8
4	ATTACAGATTACC...	3
5	ATTACC...	10
6	C...	15
7	CAGATTACC...	7
8	CC...	14
9	GATTACAGATTACC...	2
10	GATTACC...	9
11	TACAGATTACC...	5
12	TACC...	12
13	TGATTACAGATTACC...	1
14	TTACAGATTACC...	4
15	TTACC...	11

Lo
Hi
→

Binary Search Analysis

- Binary Search

Initialize search range to entire list

$\text{mid} = (\text{hi} + \text{lo}) / 2$; $\text{middle} = \text{suffix}[\text{mid}]$

if query matches middle: done

else if query < middle: pick low range

else if query > middle: pick hi range

Repeat until done or empty range

[WHEN?]

- Analysis

- More complicated method

- How many times do we repeat?

- How many times can it cut the range in half?

- Find smallest x such that: $n / (2^x) \leq 1$; $x = \lg_2(n)$

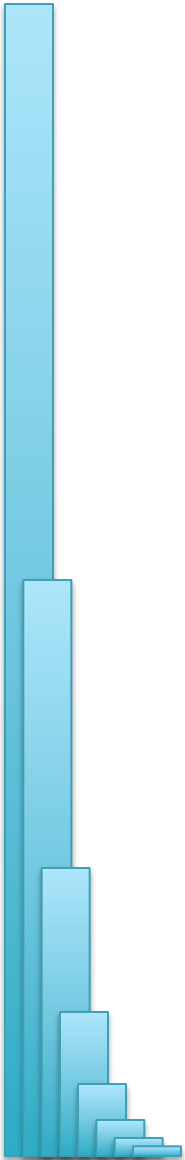
[32]

- Total Runtime: $O(m \lg n)$

- More complicated, but **much** faster!

- Looking up a query loops 32 times instead of 3B

[How long does it take to search 6B or 24B nucleotides?]



Suffix Array Construction

- How can we store the suffix array?
[How many characters are in all suffixes combined?]

$$S = 1 + 2 + 3 + \dots + n = \sum_{i=1}^n i = \frac{n(n+1)}{2} = O(n^2)$$

- Hopeless to explicitly store 4.5 billion billion characters
- Instead use implicit representation
 - Keep 1 copy of the genome, and a list of sorted offsets
 - Storing 3 billion offsets fits on a server (12GB)
- Searching the array is very fast, but it takes time to construct
 - This time will be amortized over many, many searches
 - Run it once "overnight" and save it away for all future queries

Pos
6
13
8
3
10
15
7
14
2
9
5
12
1
4
11

TGATTACAGATTACC

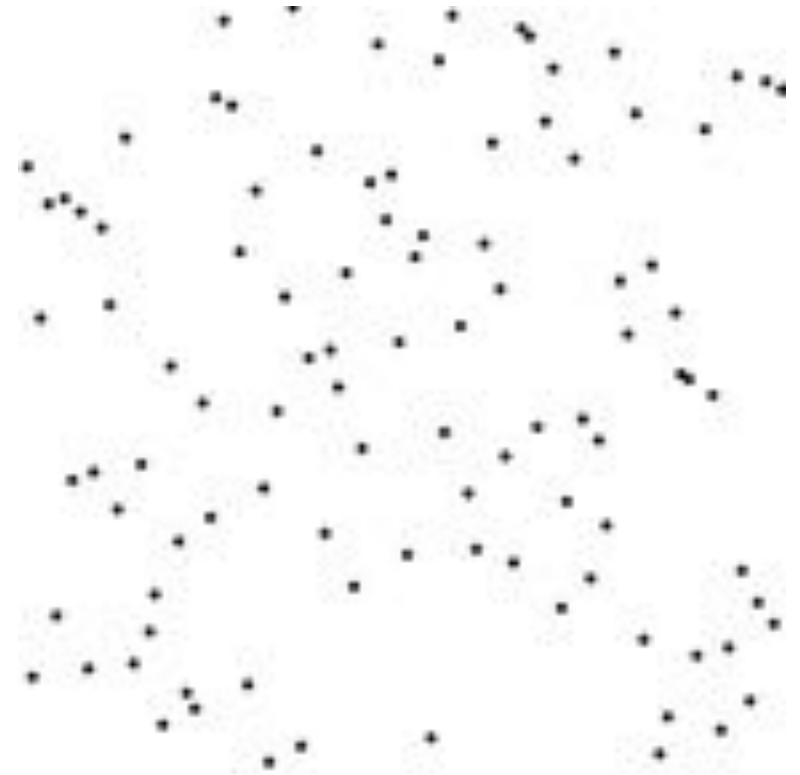
Sorting

Quickly sort these numbers into ascending order:

14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

[How do you do it?]

6, 13, 14, 29, 31, 39, 64, 78, 50, 63, 61, 19
6, 13, 14, 29, 31, 39, 64, 78, 50, 63, 61, 19
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 64, 78, 50, 63, 61
6, 13, 14, 19, 29, 31, 39, 50, 64, 78, 63, 61
6, 13, 14, 19, 29, 31, 39, 50, 61, 64, 78, 63
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78
6, 13, 14, 19, 29, 31, 39, 50, 61, 63, 64, 78



http://en.wikipedia.org/wiki/Selection_sort

Selection Sort Analysis

- Selection Sort (Input: list of n numbers)

```
for pos = 1 to n
```

```
    // find the smallest element in [pos, n]
```

```
    smallest = pos
```

```
    for check = pos+1 to n
```

```
        if (list[check] < list[smallest]): smallest = check
```

```
    // move the smallest element to the front
```

```
    tmp = list[smallest]
```

```
    list[pos] = list[smallest]
```

```
    list[smallest] = tmp
```

- Analysis

$$T = n + (n - 1) + (n - 2) + \cdots + 3 + 2 + 1 = \sum_{i=1}^n i = \frac{n(n + 1)}{2} = O(n^2)$$

- Outer loop: pos = 1 to n

- Inner loop: check = pos to n

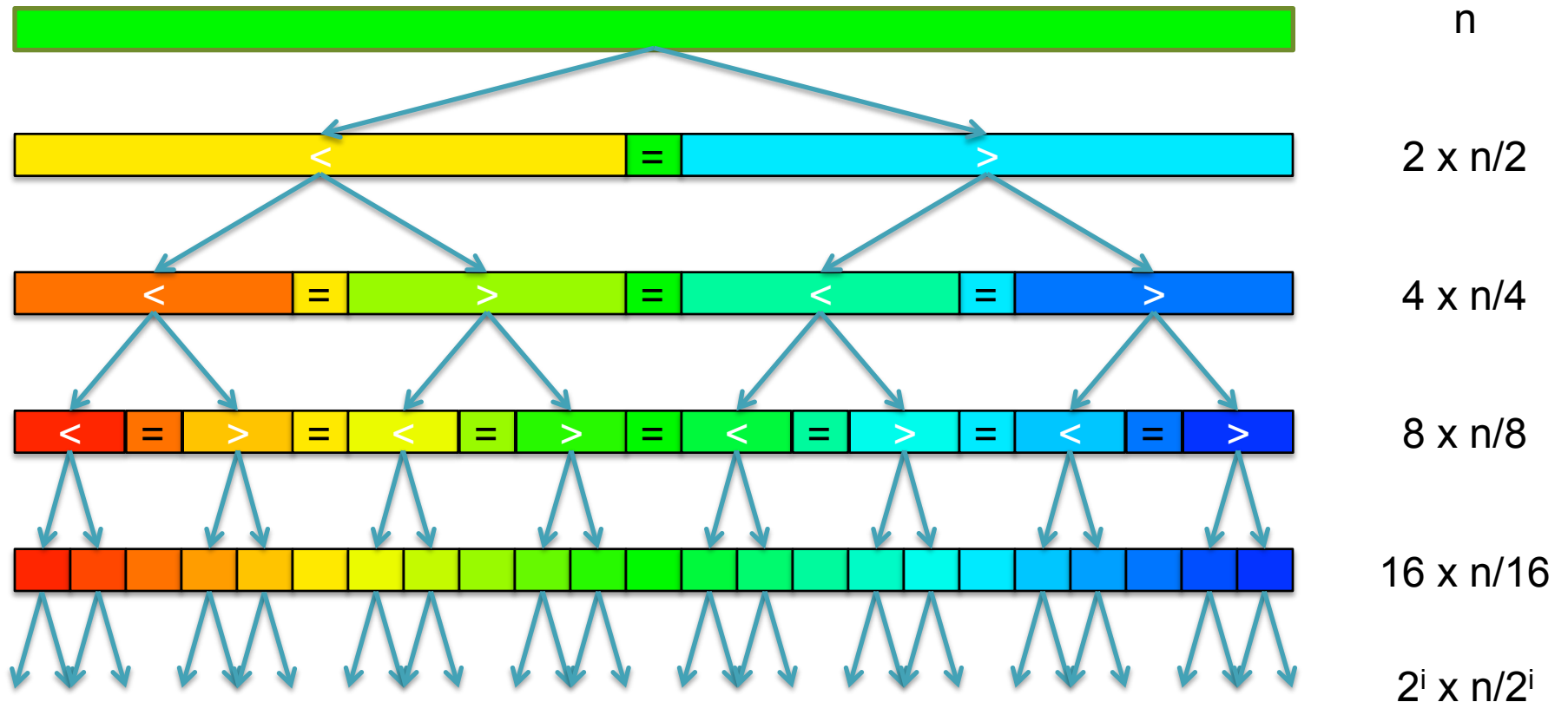
- Running time: Outer * Inner = $O(n^2)$

[4.5 Billion Billion]

[Challenge Questions: Why is this slow? / Can we sort any faster?]

Divide and Conquer

- Selection sort is slow because it rescans the entire list for each element
 - How can we split up the unsorted list into independent ranges?
 - Hint 1: Binary search splits up the problem into 2 independent ranges (hi/lo)
 - Hint 2: Assume we know the median value of a list



[How many times can we split a list in half?]

QuickSort Analysis

- QuickSort(Input: list of n numbers)

```
// see if we can quit
```

```
if (length(list)) <= 1: return list
```

```
// split list into lo & hi
```

```
pivot = median(list)
```

```
lo = {}; hi = {};
```

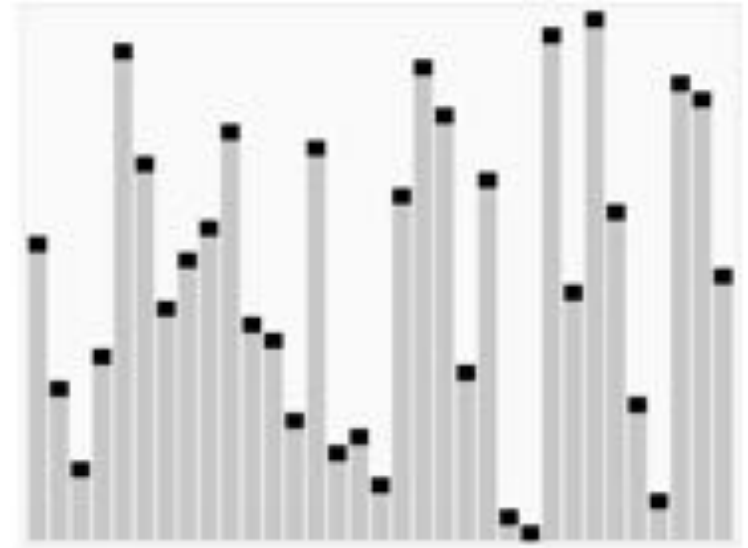
```
for (i = 1 to length(list))
```

```
    if (list[i] < pivot): append(lo, list[i])
```

```
    else:                append(hi, list[i])
```

```
// recurse on sublists
```

```
return (append(QuickSort(lo), QuickSort(hi)))
```



<http://en.wikipedia.org/wiki/Quicksort>

- Analysis (Assume we can find the median in $O(n)$)

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ O(n) + 2T(n/2) & \text{else} \end{cases}$$

$$T(n) = n + 2\left(\frac{n}{2}\right) + 4\left(\frac{n}{4}\right) + \cdots + n\left(\frac{n}{n}\right) = \sum_{i=0}^{\lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{\lg(n)} n = O(n \lg n) \quad [\sim 94B]$$

QuickSort Analysis

- QuickSort(Input: list of n numbers)

```
// see if we can quit
```

```
if (length(list)) <= 1: return list
```

```
// split list into lo & hi
```

```
pivot = median(list)
```

```
lo = {}; hi = {};
```

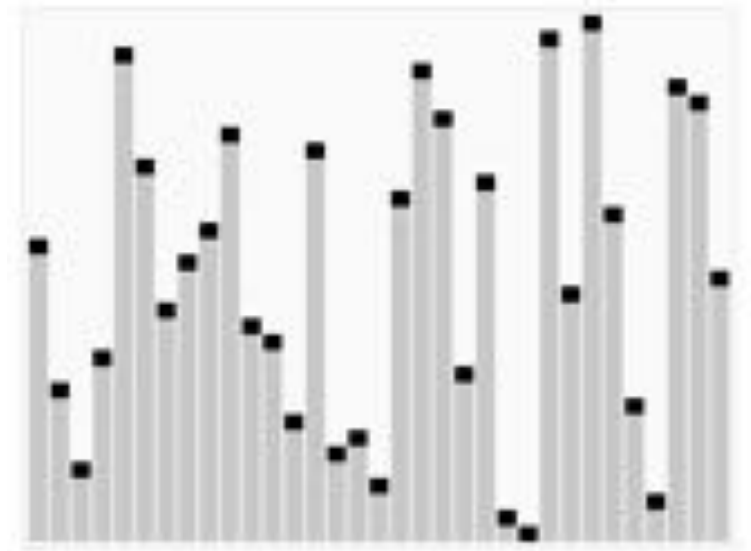
```
for (i = 1 to length(list))
```

```
    if (list[i] < pivot): append(lo, list[i])
```

```
    else:                append(hi, list[i])
```

```
// recurse on sublists
```

```
return (append(QuickSort(lo), QuickSort(hi)))
```



<http://en.wikipedia.org/wiki/Quicksort>

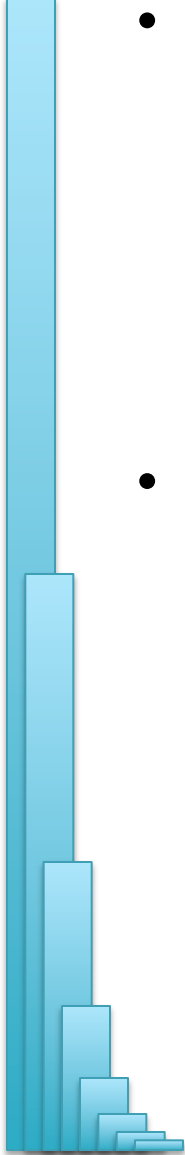
- Analysis (Assume we can find the median in $O(n)$)

$$T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ O(n) + 2T(n/2) & \text{else} \end{cases}$$

$$T(n) = n + 2\left(\frac{n}{2}\right) + 4\left(\frac{n}{4}\right) + \cdots + n\left(\frac{n}{n}\right) = \sum_{i=0}^{\lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{\lg(n)} n = O(n \lg n) \quad [\sim 94B]$$

2 minute break

Quick recap

- 
- Sequence Alignment
 1. Brute Force Exact Matching – simple but slow
 2. Suffix Arrays – very fast matching
 3. Inexact Alignment - TODO
 4. Bowtie - TODO
 - Computational Thinking
 1. Algorithm – “Formal” recipe, precise definition of problem
 2. Data structure – Choices of how to represent data
 3. Computational Analysis – Time, space requirements
 4. Computational Modeling – Characterize expected results

In-exact alignment

- Where is GATTACA *approximately* in the human genome?
 - And how do we efficiently find them?
- It depends...
 - Define 'approximately'
 - Hamming Distance, Edit distance, or Sequence Similarity
 - Ungapped vs Gapped vs Affine Gaps
 - Global vs Local
 - All positions or the single 'best'?
 - Efficiency depends on the data characteristics & goals
 - Smith-Waterman: Exhaustive search for optimal alignments
 - BLAST: Hash-table based homology searches
 - Bowtie: BWT alignment for short read mapping

Searching for GATTACA

- Where is GATTACA *approximately* in the human genome?

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

Match Score: 1/7

Searching for GATTACA

- Where is GATTACA *approximately* in the human genome?

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

Match Score: 7/7

Searching for GATTACA

- Where is GATTACA *approximately* in the human genome?

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

Match Score: 1/7

Searching for GATTACA

- Where is GATTACA *approximately* in the human genome?

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

Match Score: 6/7 <- We may be very interested in these imperfect matches
Especially if there are no perfect end-to-end matches

Hamming Distance

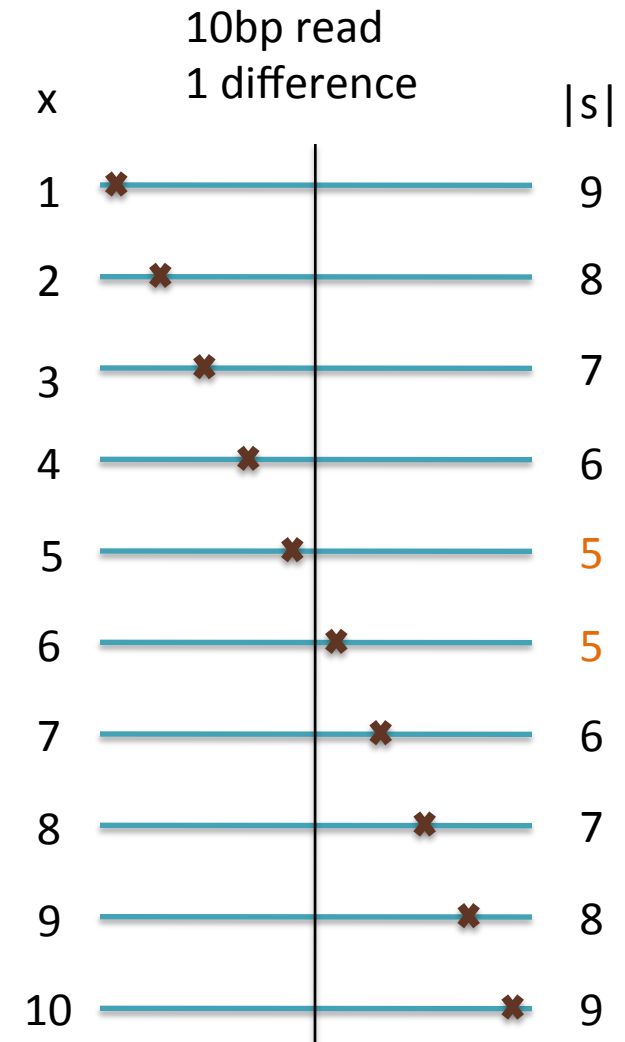


- How many characters are different between the 2 strings?
 - Minimum number of substitutions required to change transform A into B
- Traditionally defined for end-to-end comparisons
 - Here end-to-end (global) for query, partial (local) for reference
- Find all occurrences of GATTACA with Hamming Distance ≤ 1
- Find all occurrences with minimal Hamming Distance
 - [What is the running time of a brute force approach?]

Seed-and-Extend Alignment

Theorem: An alignment of a sequence of length m with at most k differences **must** contain an exact match at least $s = m / (k + 1)$ bp long
(Baeza-Yates and Perleberg, 1996)

- Proof: Pigeonhole principle
 - 1 pigeon can't fill 2 holes
- Seed-and-extend search
 - Use an index to rapidly find short exact alignments to seed longer in-exact alignments
 - BLAST, MUMmer, Bowtie, BWA, SOAP, ...
 - Specificity of the depends on seed length
 - Guaranteed sensitivity for k differences
 - Also finds some (but not all) lower quality alignments <- heuristic

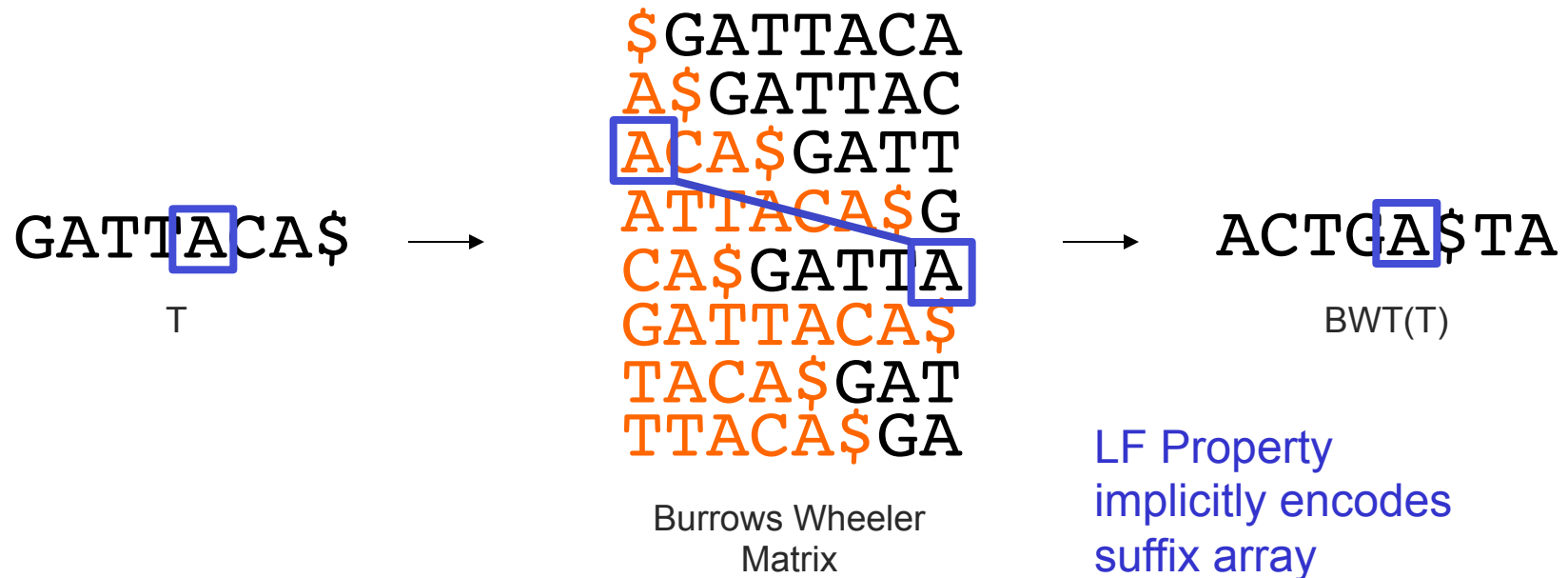




Bowtie: Ultrafast and memory efficient alignment of short DNA sequences to the human genome

Slides Courtesy of Ben Langmead
(langmead@umiacs.umd.edu)

Burrows-Wheeler Transform



- Suffix Array is tight, but much larger than genome
 - BWT is a reversible permutation of the genome based on the suffix array
 - Core index for Bowtie (Langmead et al., 2009) and most recent short read mapping applications

A block sorting lossless data compression algorithm.

Burrows M, Wheeler DJ (1994) *Digital Equipment Corporation, Palo Alto, CA* 1994, Technical Report 124

Bowtie algorithm

Reference



BWT(Reference)



Query:

AATGATACGGCGACCAACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)

Query:

AATGATACGGCGACCAACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)



Query:

AATGATACGGCGACCAACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)



Query:

AATGATACGGCGACCAACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)



Query:

AATGATACGGCGACCACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)



Query:

AATGATACGGCGACCAACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)

Query:

AATGATACGGCGACCAACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)



Query:

AATG **T** TACGGCGACCAACCGAGATCTA



Bowtie algorithm

Reference



BWT(Reference)



Query:

AATGTTACGGCGACCAACCGAGATCTA



BWT Short Read Mapping

- Seed-and-extend search of the BWT
 1. If we fail to reach the end, back-track and resume search
 2. The beginning of the read is used as high confidence seed
 3. BWT enables searching for good end-to-end matches entirely in RAM
 1. 100s of times faster than competing approaches
- Report the "best" n alignments
 1. Best = smallest hamming distance, possibly weighted by QV
 2. Some reads will have millions of equally good mapping positions
 3. If reads are paired, try to find mapping that satisfies both

Algorithms Summary

- Algorithms choreograph the dance of data inside the machine
 - Algorithms add provable precision to your method
 - A smarter algorithm can solve the same problem with much less work
- Techniques
 - Analysis: Characterize performance, correctness
 - Modeling: Characterize what you expect to see
 - Binary search: Fast lookup in any sorted list
 - Divide-and-conquer: Split a hard problem into an easier problem
 - Recursion: Solve a problem using a function of itself
 - Indexing: Focus on just the important parts
 - Seed-and-extend: Anchor the problem using a portion of it

Challenge Question

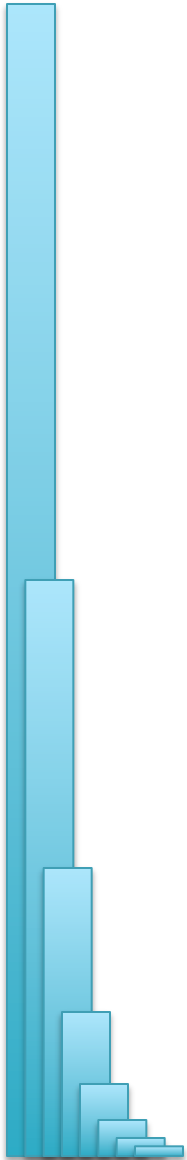
Using Bowtie (bowtie -v 0 -a --norc) or your own implementation of the brute force algorithm, scan the E. coli K12/MG1655 genome for GATTACA:

<http://schatzlab.cshl.edu/teaching/2011/Ecoli.fa>

<http://schatzlab.cshl.edu/teaching/2011/GATTACA.fq>

Compute the number of occurrences for each of the following queries, and the degree to which the empirical number of matches is consistent with the theoretical e-value. Point out any particularly significant deviations from the theoretical model.

Gattaca:	GATTACA
Gattaca ² :	GATTACAGATTACA
Gattaca ³ :	GATTACAGATTACAGATTACA
Start Codon:	ATG
Stop Codons:	TAG, TAA, TGA

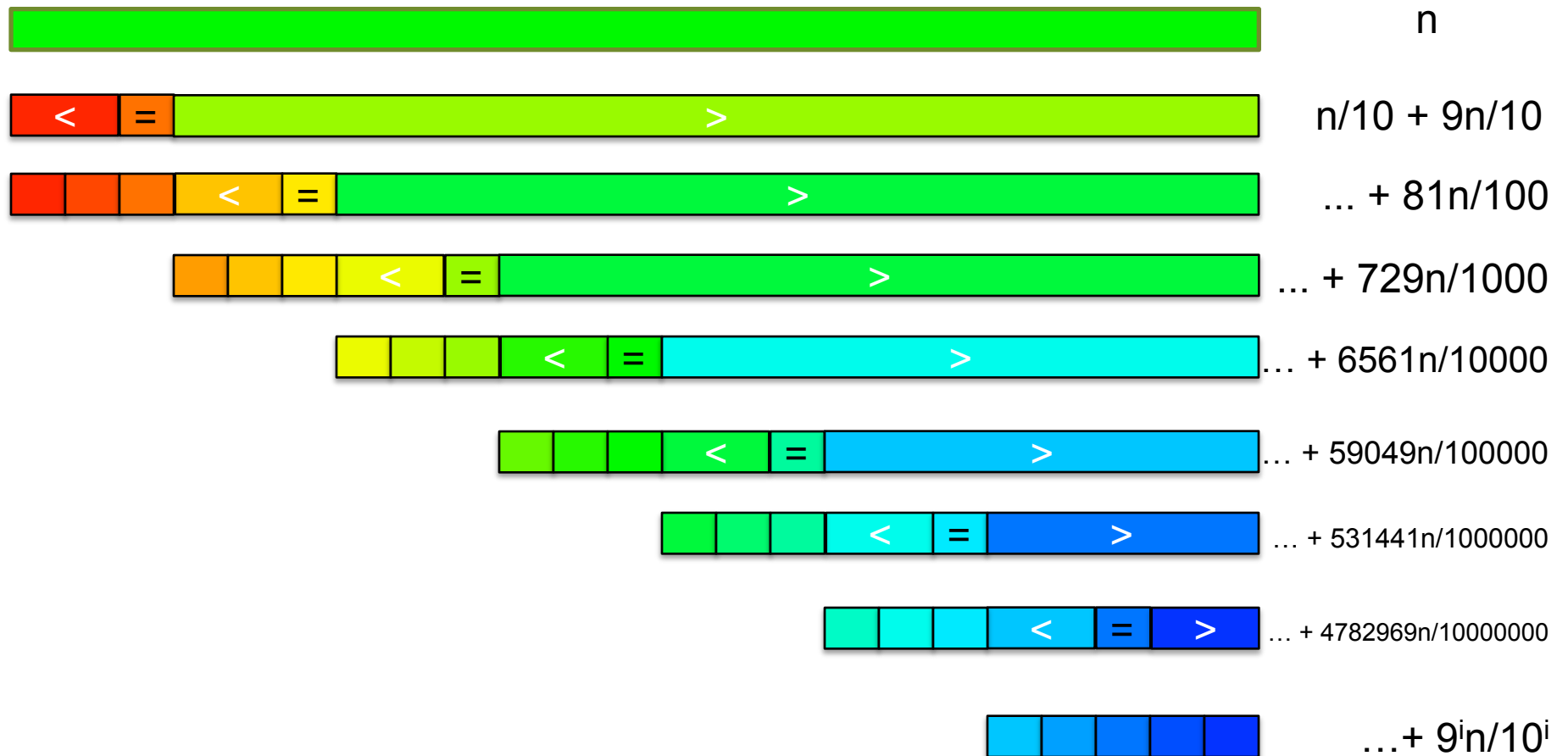


Thank You!

<http://schatzlab.cshl.edu>

Picking the Median

- What if we miss the median and do a 90/10 split instead?



[How many times can we cut 10% off a list?]

Randomized Quicksort

- 90/10 split runtime analysis

Find smallest x s.t.

$$T(n) = n + T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right)$$

$$(9/10)^x n \leq 1$$

$$T(n) = n + \frac{n}{10} + T\left(\frac{n}{100}\right) + T\left(\frac{9n}{100}\right) + \frac{9n}{10} + T\left(\frac{9n}{100}\right) + T\left(\frac{81n}{100}\right)$$

$$(10/9)^x \geq n$$

$$T(n) = n + n + T\left(\frac{n}{100}\right) + 2T\left(\frac{9n}{100}\right) + T\left(\frac{81n}{100}\right)$$

$$x \geq \log_{10/9} n$$

$$T(n) = \sum_{i=0}^{\log_{10/9}(n)} n = O(n \lg n)$$

- If we randomly pick a pivot, we will get at least a 90/10 split with very high probability
 - Everything is okay as long as we always slice off a fraction of the list

[Challenge Question: What happens if we slice 1