Exact Matching & CS Fundamentals

Michael Schatz

Bioinformatics Lecture 1
Quantitative Biology 2010

Computer Science & Computational Biology

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

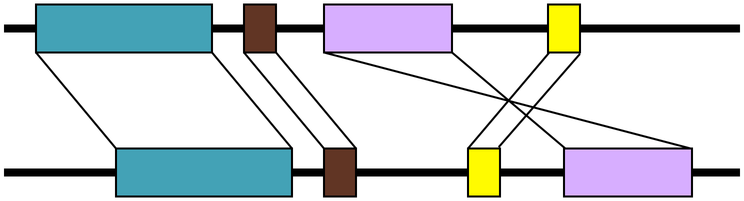
- Computer Science = Science of Computation
  - Solving problems, designing & building systems
  - Thinking recursively about data, across levels of abstraction
  - Reasoning that your methods are fast & correct

- Computer Science >> Computer Programming
  - Computers are very, very dumb, but we can instruct them
    - Build complex systems out of simple components
    - They will perfectly & repeatedly execute instructions forever

- CompBio = Thinking Computationally about Biology
  - Processing: Make more powerful instruments, analyze results
  - Designing & Understanding: protocols, procedures, systems

Sequence Alignment Applications

- A very common problem in computational biology is to find occurrences of one sequence in another sequence
  - Genome Assembly
  - Gene Finding
  - Comparative Genomics
  - Functional analysis of proteins
  - Motif discovery
  - SNP analysis
  - Phylogenetic analysis
  - Primer Design
  - Personal Genomics
  - …



Exact Matching Overview

Where is GATTACA in the human genome?

Brute Force
(3 GB)

```
BANANA
BAN
 ANA
  NAN
   ANA
```

Naive

Slow & Easy

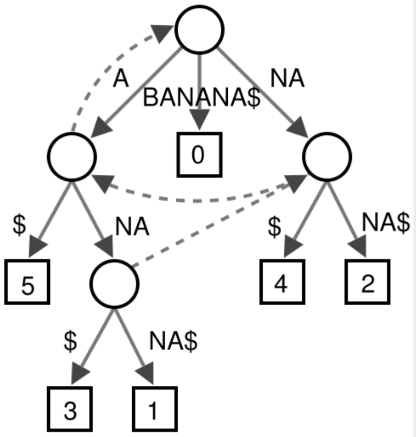
Suffix Array
(>15 GB)

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Vmatch, PacBio Aligner

Binary Search

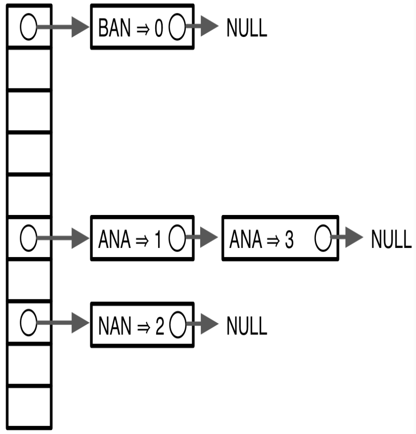
Suffix Tree
(>51 GB)



MUMmer, MUMmerGPU

Tree Searching

Hash Table
(>15 GB)



BLAST, MAQ, ZOOM, RMAP, CloudBurst

Seed-and-extend

Searching for GATTACA

- Where is GATTACA in the human genome?

- Strategy 1: 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 1: 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 1: 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

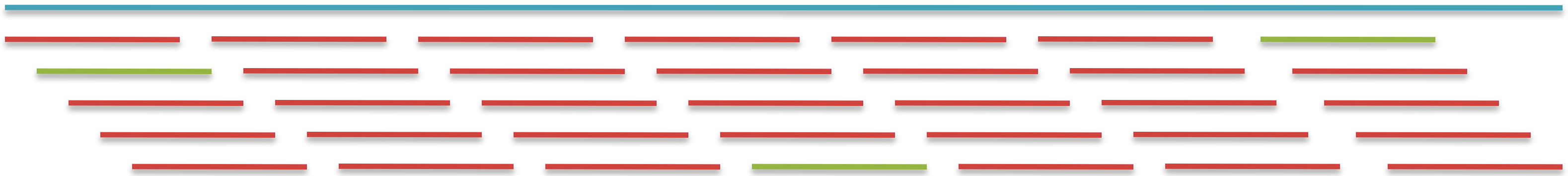
- Where is GATTACA in the human genome?

- Strategy 1: 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?

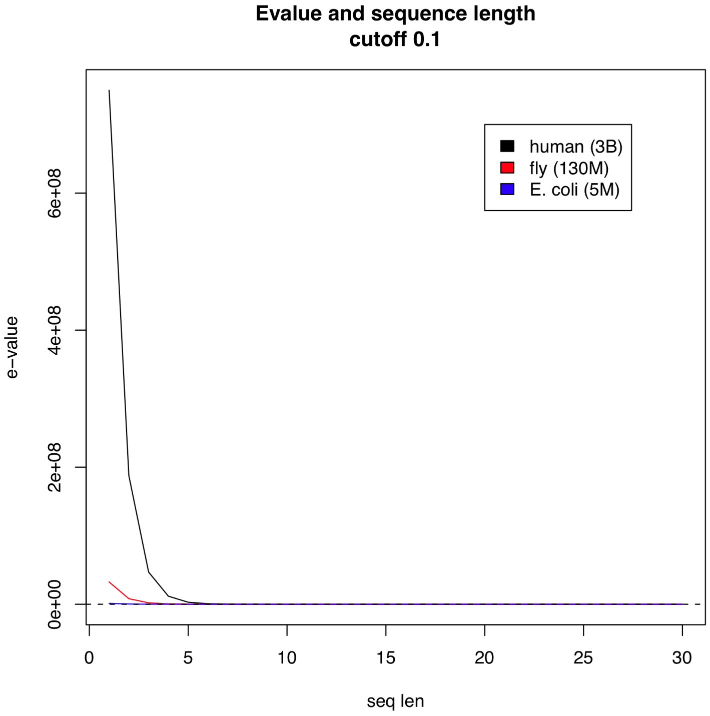
- Analysis
  - Simple, easy to understand
  - Genome length = n [3B]
  - Query length = m [7]
  - Comparisons: (n-m+1) * m [21B]

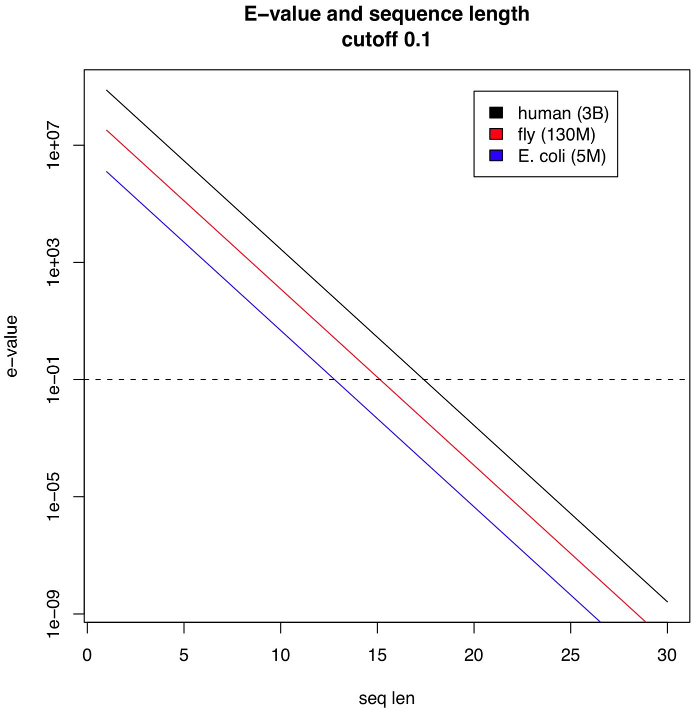
- Overall runtime: O(nm)
  - If we double genome or query size, takes twice as long
  - If we double both, takes 4 times as long

Expected Occurrences

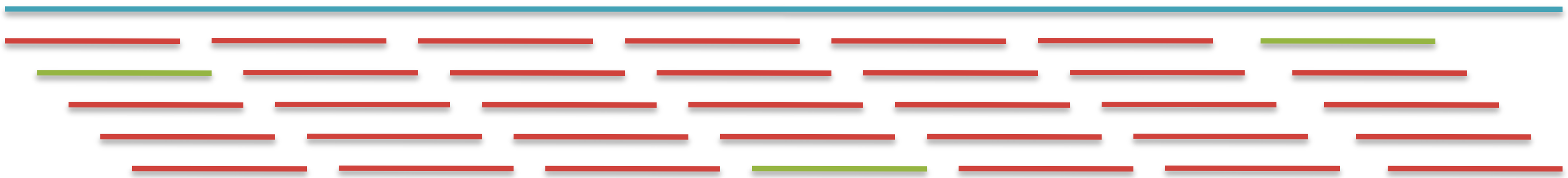
The expected number of occurrences (e-value) of a given sequence in a genome depends on the length of the genome and inversely on the length of the sequence

- 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-m+1)/(4^m) [183,105 expected occurrences]





Brute Force in Matlab



```
query  = 'GATTACA';
genome = 'TGATTACAGATTACC';

nummatches=0;

% At every possible offset
for offset=1:length(genome)-length(query)+1
    % Do all of the characters match?
    if (genome(offset:offset+length(query)-1) == query)
          disp(['Match at offset ', num2str(offset)])
          nummatches = nummatches+1;
    else
          %Uncomment to see every non-match
          %disp(['No match at offset ', num2str(offset)])
    end
end

disp(['Found ', num2str(nummatches),' matches of ', query, ' in genome of length ',
    num2str(length(genome))])


disp(['Expected number of occurrences: ', num2str((length(genome)-length(query)+1)/
    (4^length(query)))])
```

Brute Force Reflections

Why check every position?

- GATTACA can't 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	

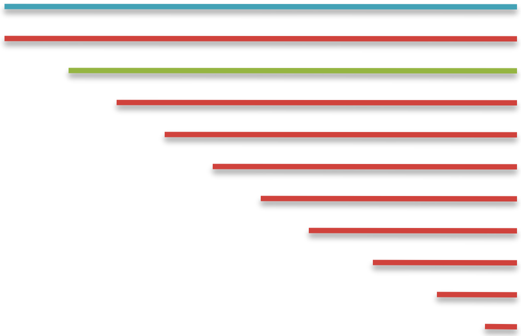
- Improve runtime to O(n + m) [3B + 7]
  - If we double both, it just takes twice as long
  - Knuth-Morris-Pratt, 1977
  - Boyer-Moyer, 1977, 1991

- 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

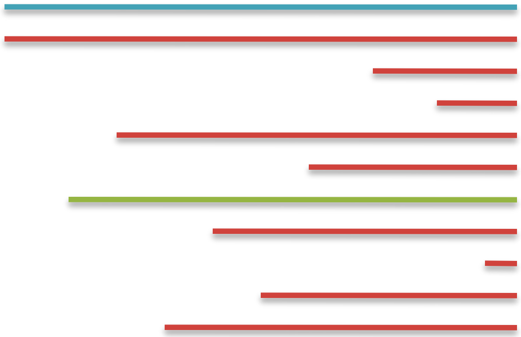
2. Suffix Arrays

- 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
Lo →	9	GATTACAGATTACC…	2
	10	GATTACC…	9
	11	TACAGATTACC…	5
	12	TACC…	12
	13	TGATTACAGATTACC…	1
	14	TTACAGATTACC…	4
Hi →	15	TTACC…	11

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

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; Mid = (9+15)/2 = 12
  - Middle = Suffix[12] = TACC
    => Lower: Hi = Mid - 1

  - Lo = 9; Hi = 11;

Lo →

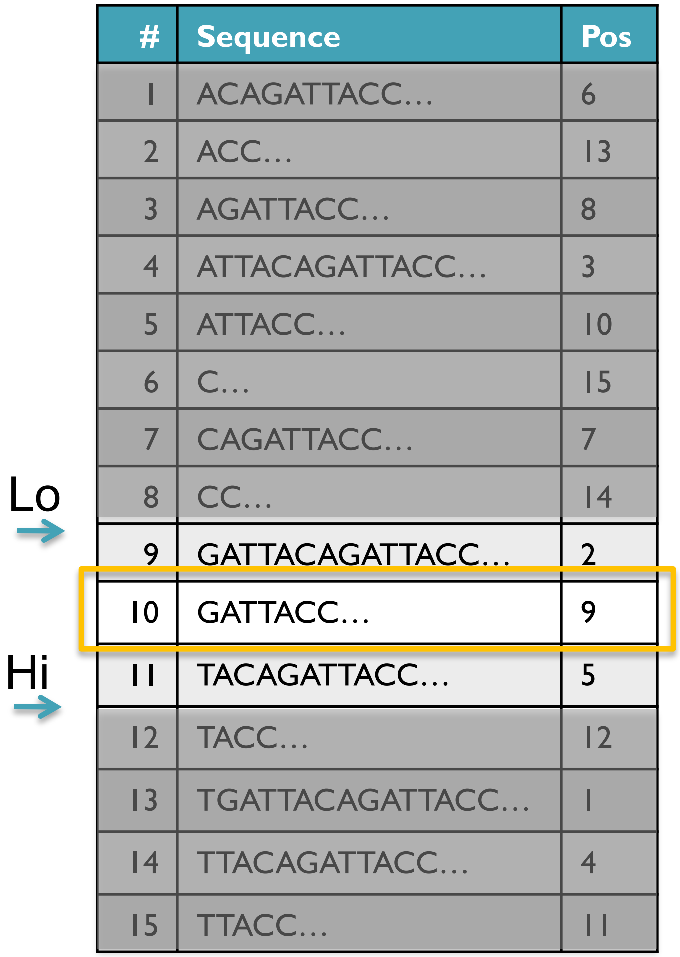
Hi →

#	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

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

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;

Lo

Hi

#	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

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

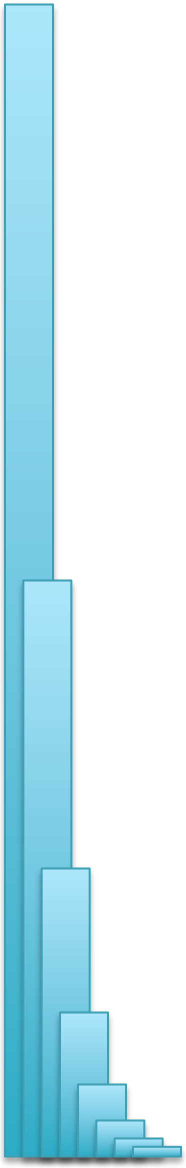
  - Lo = 9; Hi = 11; Mid = (9+11)/2 = 10
  - Middle = Suffix[10] = GATTACC
    => Lower: Hi = Mid - 1

  - 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

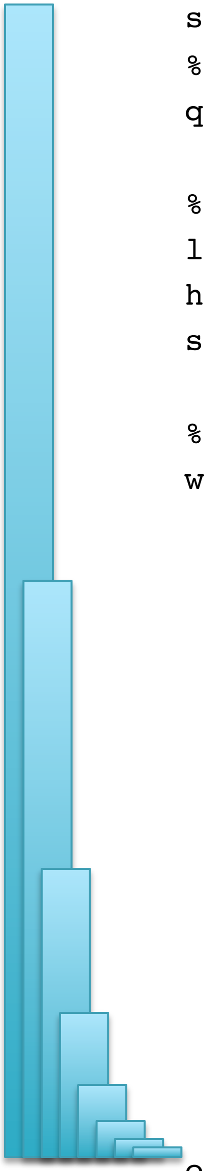
```
Initialize search range to entire list
    mid = (hi+lo)/2; middle = suffix[mid]
    if query matches middle: done
    else if query < middle: pick low range
    else if query > middle: pick hi range
Repeat until done or empty range                    [WHEN?]
```

- 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 nucleotides?]

Binary Search in Matlab

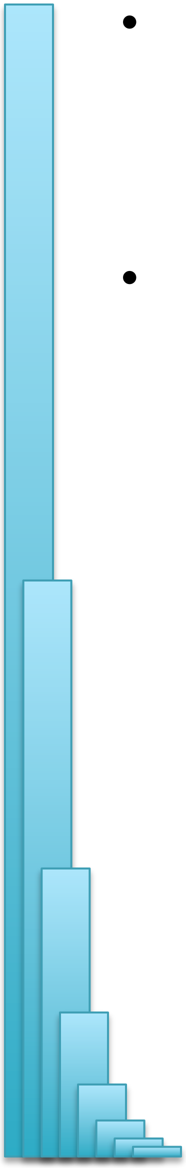


```
%% create our sorted list of 100 numbers
seq=1:100;
%% seq=sort(floor(rand(100)*100));
query=33;

%% initialize search range
lo=1;
hi=length(seq);
steps=0;

%% search
while (lo<=hi)
  steps = steps+1;
  mid=floor((lo+hi)/2);
  middle=seq(mid);
  disp(['Step ', num2str(steps), ' checking seq[', num2str(mid), ']=', num2str(middle)])
  if (query == middle)
   disp(['Found at ', num2str(mid), ' in ', num2str(steps), ' steps'])
   break
  elseif (query < middle)
    disp(['less than ', num2str(middle)])
    hi=mid-1;
  else
    disp(['greater than ', num2str(middle)])
    lo=mid+1;
  end
end
```

Suffix Array Construction



- 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

- How do we store the suffix array?
  - Explicitly storing all n strings is not feasible

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

For human genome S = 9 billion billion characters

  - Instead use implicit representation
    - Keep 1 copy of the genome, and a list of sorted offsets
    - Storing 3 billion offsets requires a big server (12GB)
      - Build a separate index for each chromosome

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

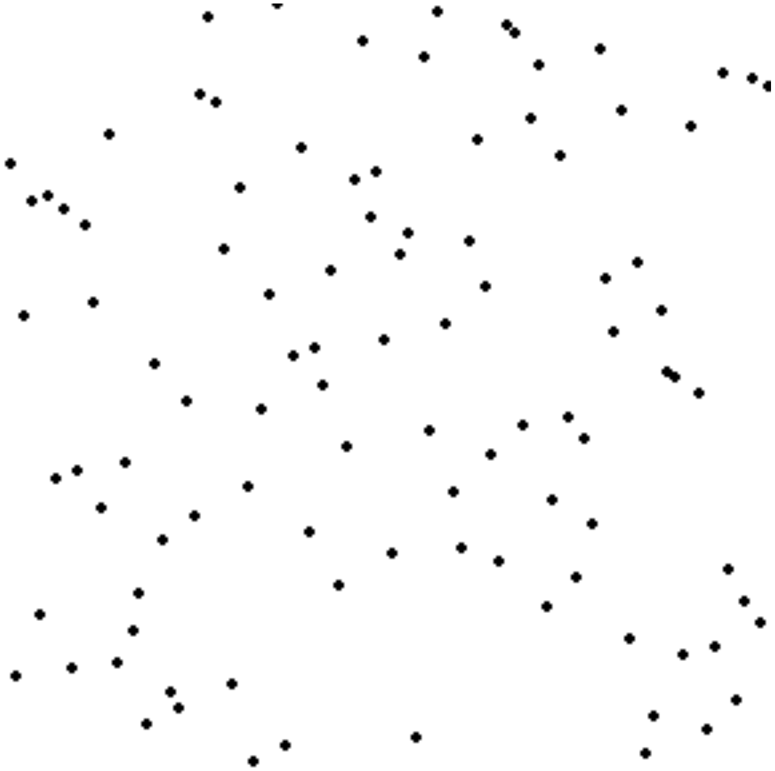
TGATTACAGATTACC

Sorting

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
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

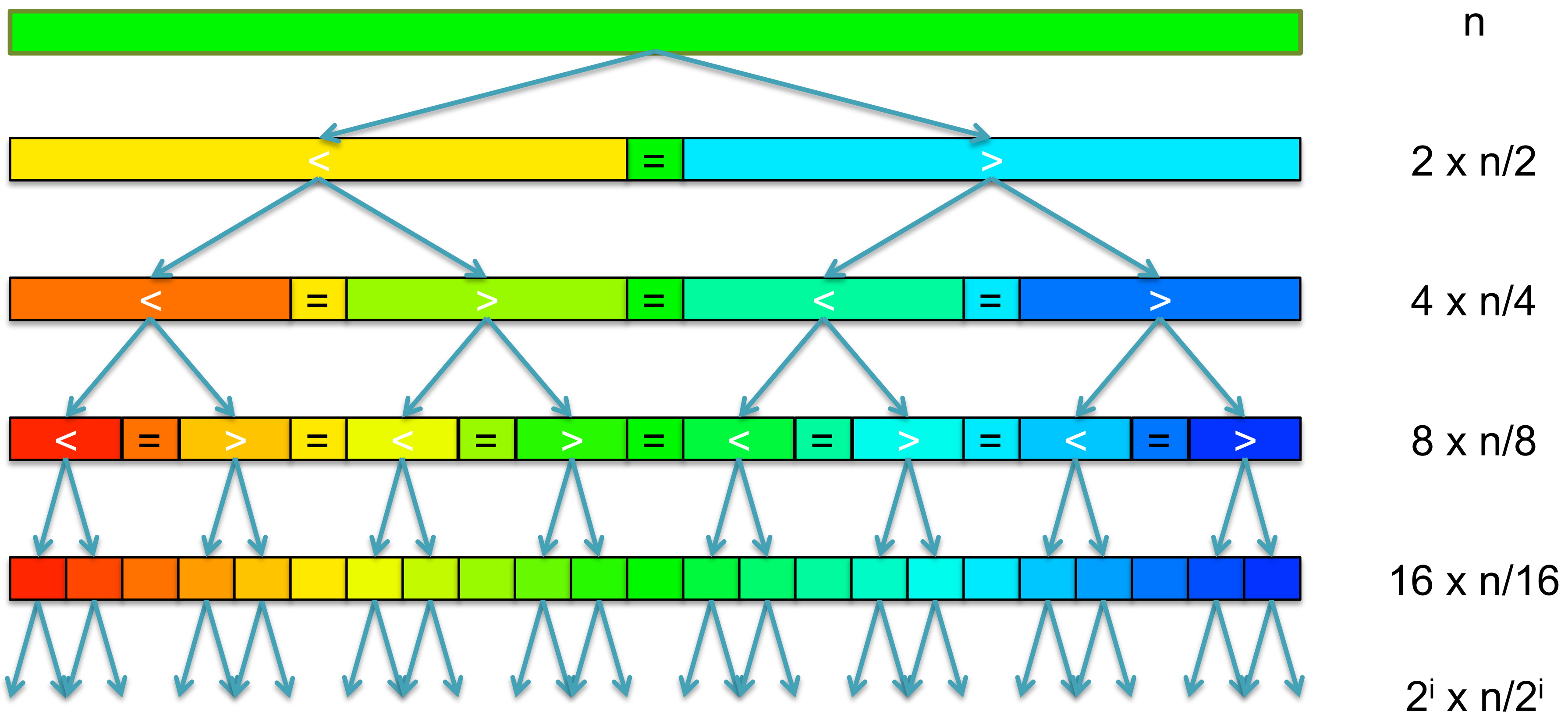
$$T = n + (n-1) + (n-2) + \cdots + 3 + 2 + 1 = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

  - Outer loop: pos = 1 to n
  - Inner loop: check = pos to n
  - Running time: Outer * Inner = $O(n^2)$ [9 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?]

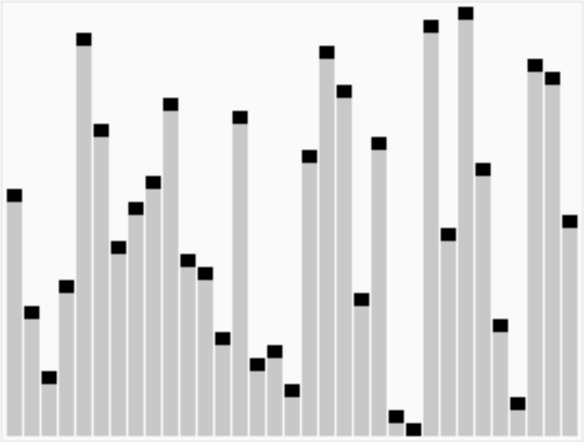
QuickSort Analysis

- 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

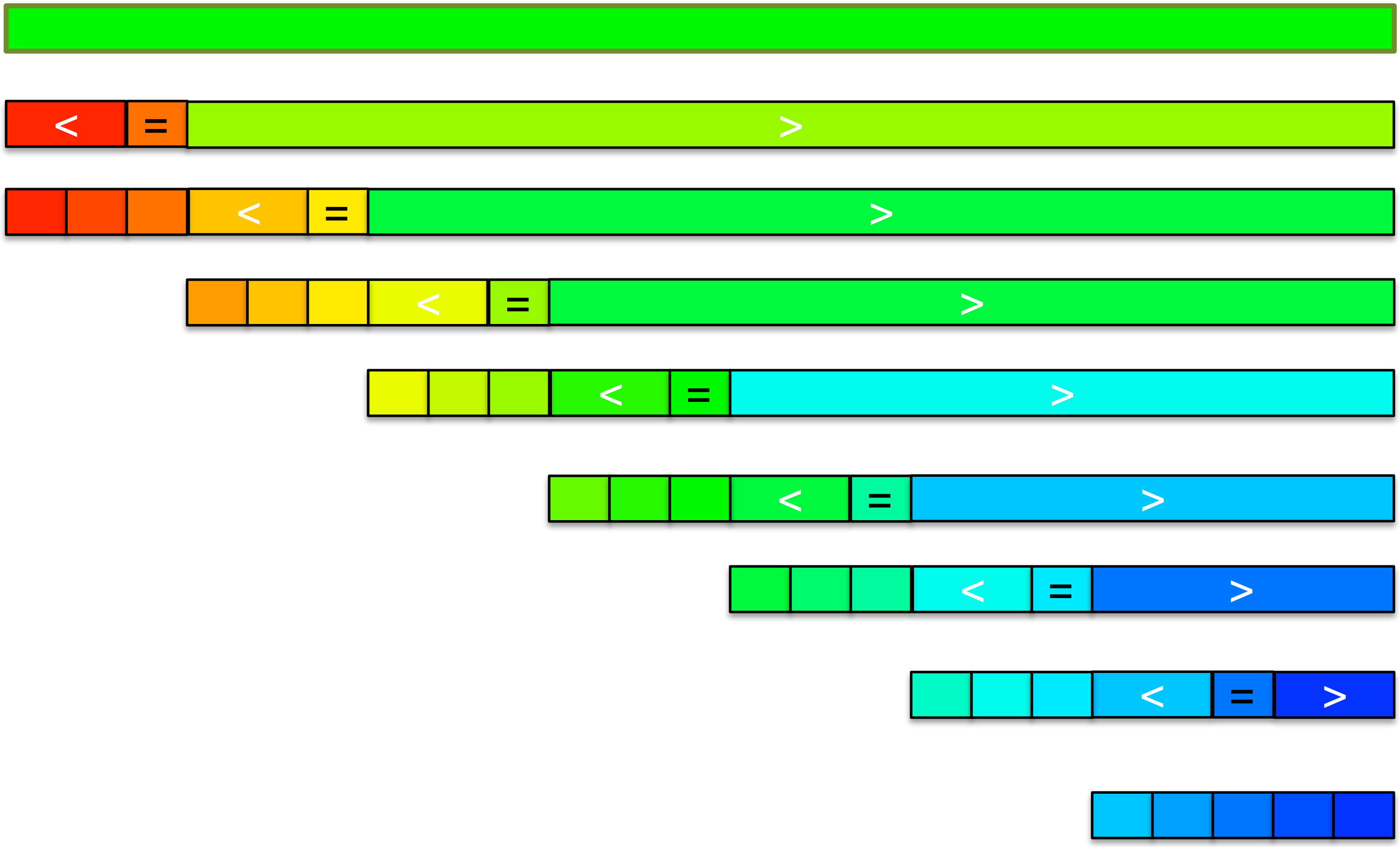
- Analysis (Assume we can find the median in O(n))

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

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

Picking the Median

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



n

$n/10 + 9n/10$

$\ldots + 81n/100$

$\ldots + 729n/1000$

$\ldots + 6561n/10000$

$\ldots + 59049n/100000$

$\ldots + 531441n/1000000$

$\ldots + 4782969n/10000000$

$\ldots + 9^i n/10^i$

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

Randomized Quicksort

- 90/10 split runtime analysis

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

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

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

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

Find smallest x s.t.

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

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

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

- 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 element]

QuickSort in Matlab

```
sort(seq)
```

- The goal of software engineering is to build libraries of correct reusable functions that implement higher level ideas
  - Build complex software out of simple components
  - Software tends to be 90% plumbing, 10% research
  - You still need to know how they work
    - Matlab requires an explicit representation of the strings

Break

Sorting in Linear Time

- Can we sort faster than O(n lg n)?
  - No – Not if we have to compare elements to each other
  - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:
14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100

Sorting in Linear Time

- Can we sort faster than O(n lg n)?
  - No – Not if we have to compare elements to each other
  - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:
14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100

Sorting in Linear Time

- Can we sort faster than O(n lg n)?
    - No – Not if we have to compare elements to each other
    - Yes – But we have to 'cheat' and know the structure of the data

Sort these numbers into ascending order:
14, 29, 6, 31, 39, 64, 78, 50, 13, 63, 61, 19

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100

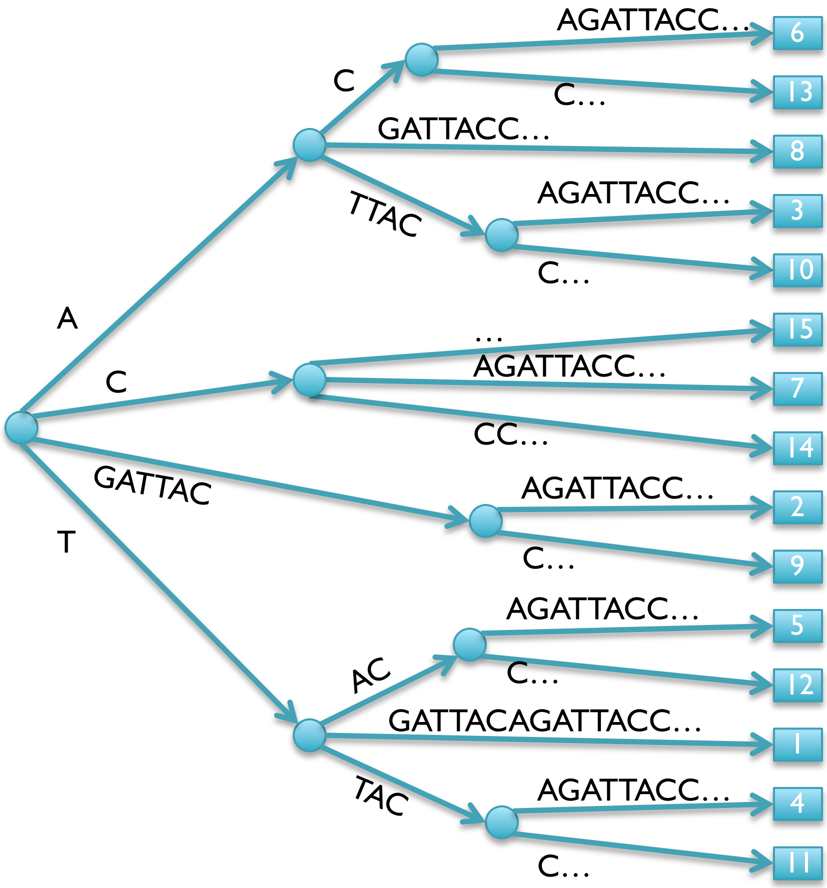
6,13,14,19,29,31,39,50,61,63,64,78

```
for(i = 1 to 100) { range[i] = 0; }
for(i = 1 to n) { range[list[i]] = 1; }
for(i = 1 to 100) { if (range[i] == 1){print i}}
```

[3B instead of 94B]

3. Suffix Trees

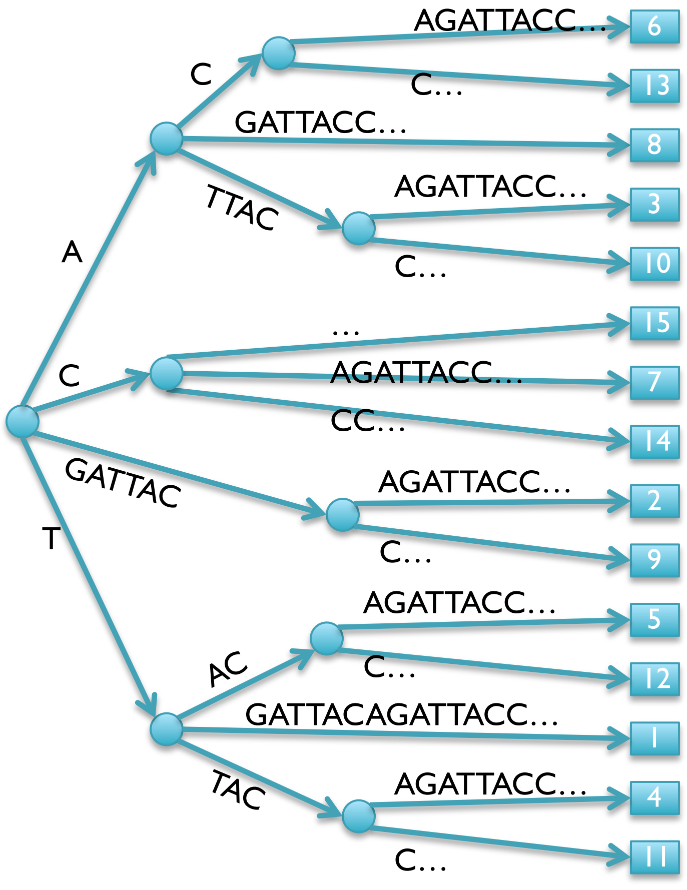
#	Sequence	Pos
1	ACAGATTACC…	6
2	ACC…	13
3	AGATTACC…	8
4	ATTACAGATTACC…	3
5	ATTACC…	10
6	C…	15
7	CAGATTACC…	7
8	CC…	14
9	GATTACAGATTACC…	2
10	GATTACC…	9
11	TACAGATTACC…	5
12	TACC…	12
13	TGATTACAGATTACC…	1
14	TTACAGATTACC…	4
15	TTACC…	11



- Suffix Tree = Tree of suffixes (indexes **all** substrings of a sequence)
    - 1 Leaf ($) for each suffix, path-label to leaf spells the suffix
    - Nodes have at least 2 and at most 5 children (A,C,G,T,$)

Suffix Trees Searching

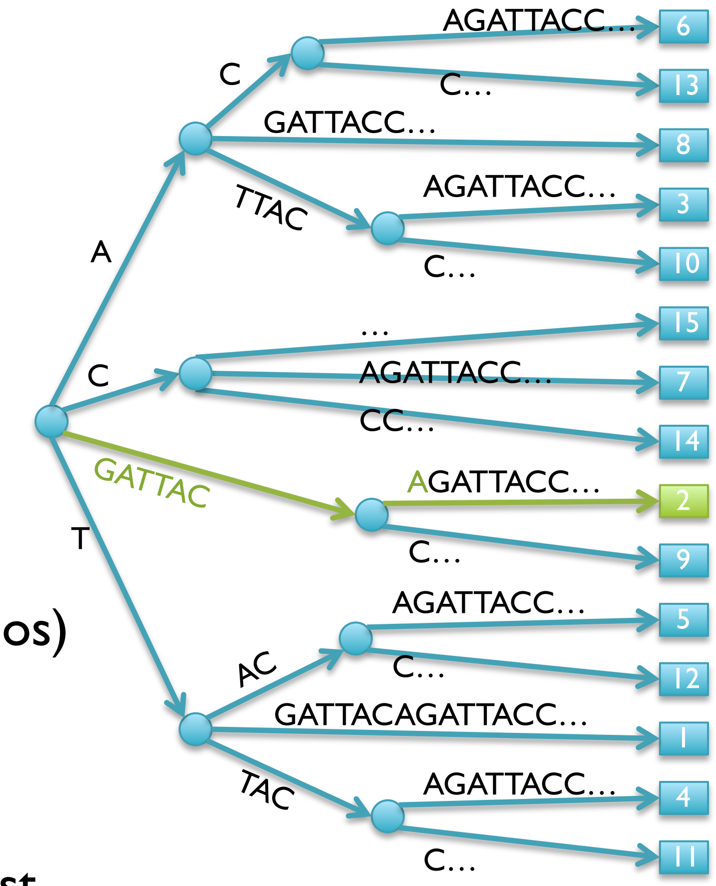
- Look up a query by "walking" along the edges of the tree
    - GATTACA



Suffix Trees Searching

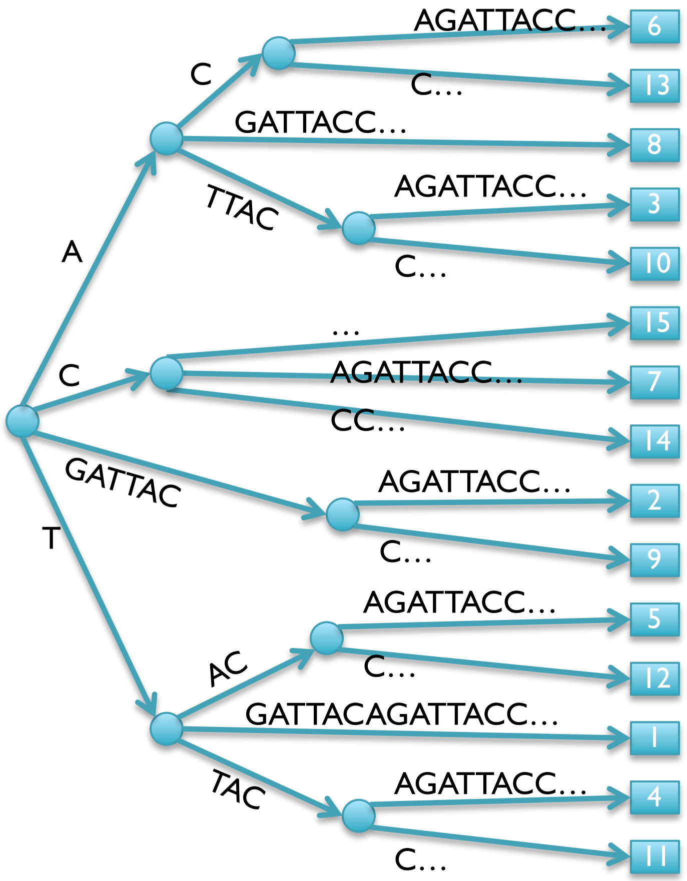
- Look up a query by "walking" along the edges of the tree
    - GATTACA
    - Matches at position 2

```
WalkTree
    cur = ST.Root; qrypos = 0;
    while (cur)
        // check for partial matches
        …
        // walk the tree
        edge = cur.getEdge(q[qrypos]); edgepos=0
        dist = matchstrings(edge, edgepos, qry, qrypos)
        if (qrypos+dist == length(qry))
            print "end-to-end match"
        else if  (dist == length(edge))
            cur=cur.getNode(edge[0]); qrypos+=dist
        else
            print "no match"
```



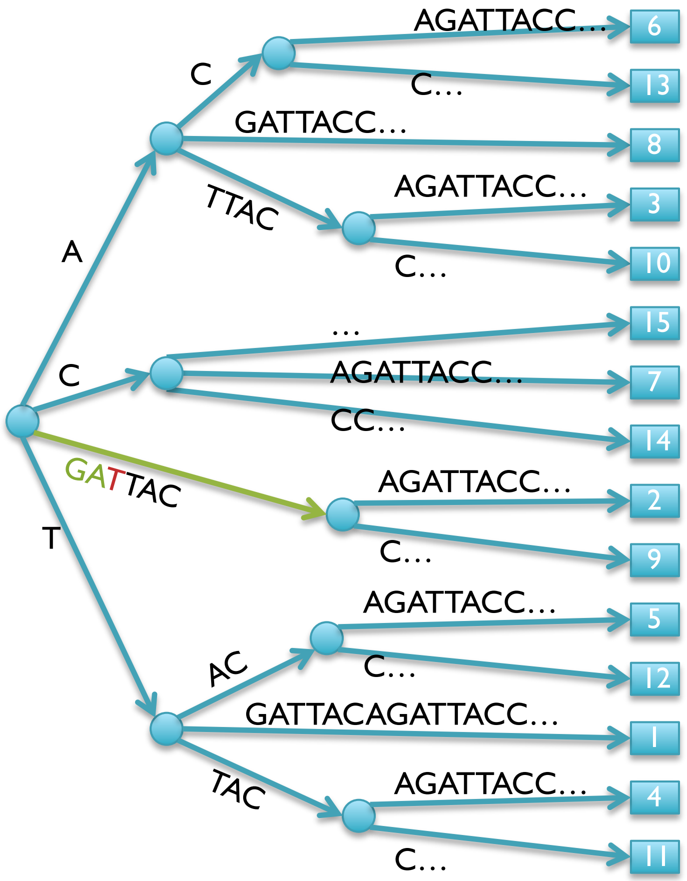
Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
    - GACTACA



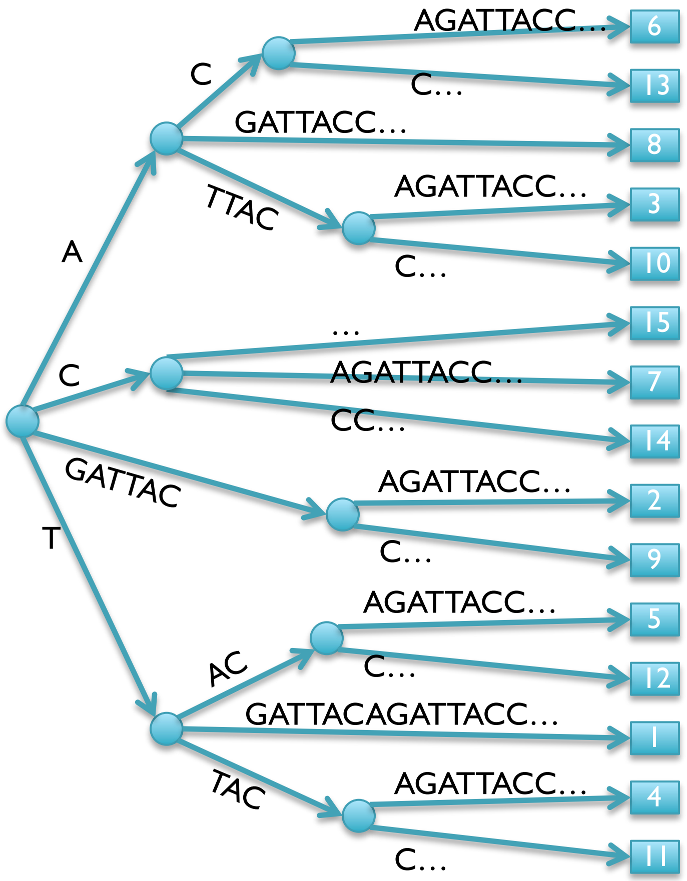
Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
    - GACTACA
    - Fell off tree – no match



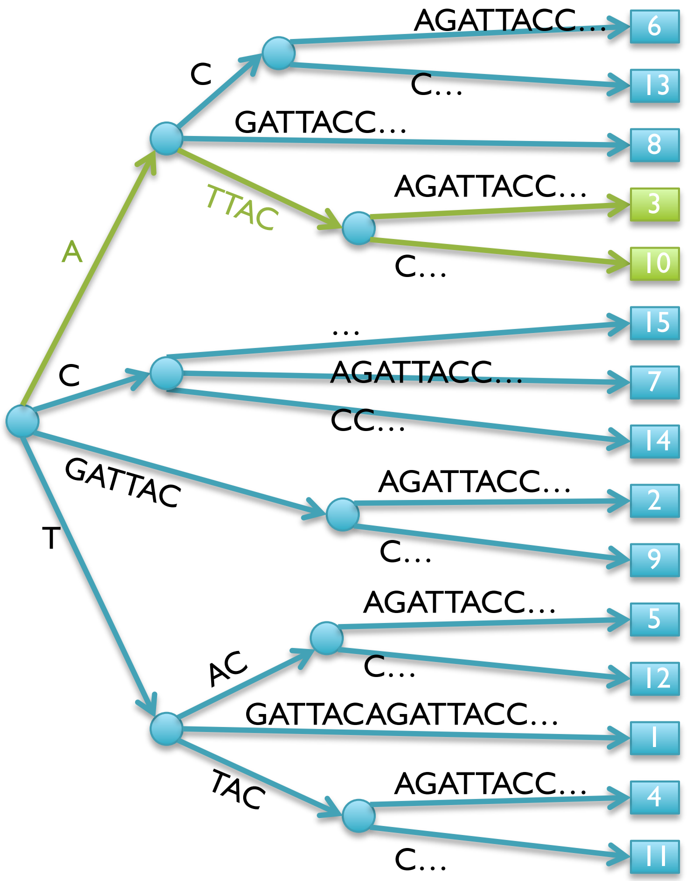
Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
    - ATTAC



Suffix Trees Searching

- Look up a query by "walking" along the edges of the tree
  - ATTAC
  - Matches at 3 and 10



- Query Lookup in 2 phases:
  1. Walk along edges to find matches
  2. Walk subtree to find positions

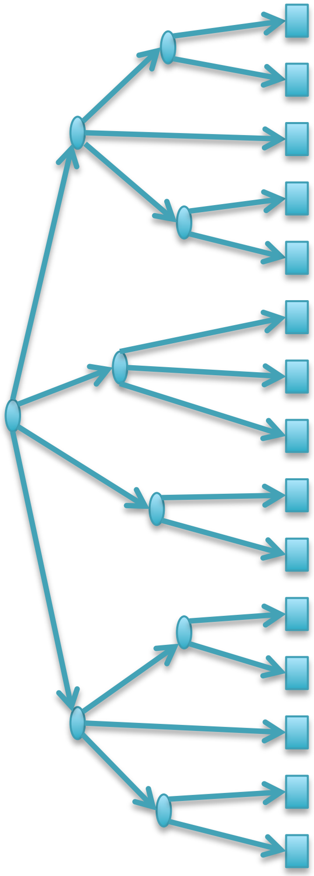
```
DepthFirstPrint(Node cur)
if cur.isLeaf
    print cur.pos
else
    foreach child in cur.children
        DepthFirstPrint(child)
```

[What is the running time of DFP
    => How many nodes does the tree have?]

Suffix Tree Properties & Applications

Properties

- Number of Nodes/Edges: O(n)
- Tree Size: O(n)
- Max Depth: O(n)
- Construction Time: O(n)
    - Uses suffix links to jump between nodes without rechecking
    - Tricky to implement, prove efficiency



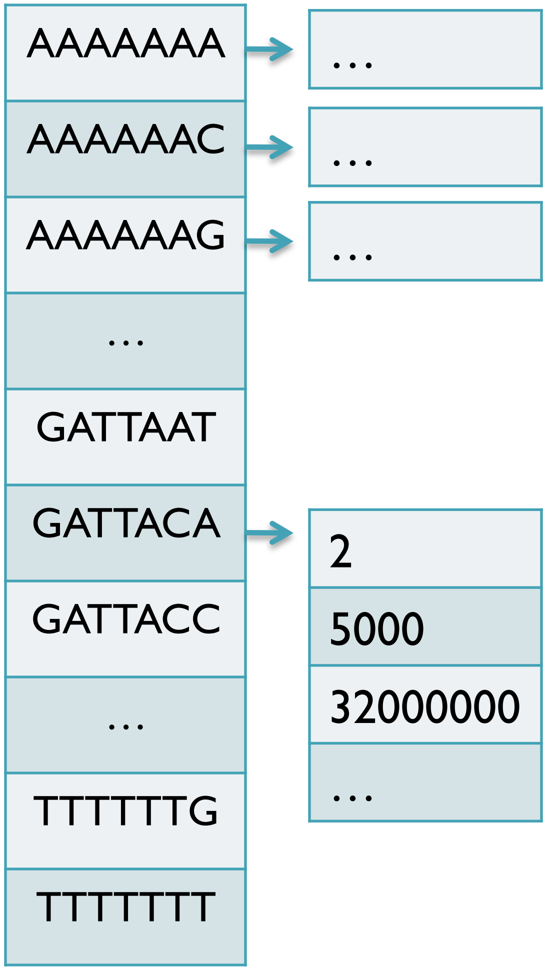
Applications

- Sorting all suffixes: O(n) [HOW?]
- Check for query: O(m)
- Find all z occurrences of a query O(m + z)
- Find maximal exact matches O(m)
- Longest common substring O(m)

- Used for many string algorithms in linear time
    - Many can be implemented on suffix arrays using a little extra work

4. Hashing

- Where is GATTACA in the human genome?
  - Build an inverted index (table) of every kmer in the genome
- How do we access the table?
  - We can only use numbers to index
    - table[GATTACA] <- error, does not compute
  - Encode sequences as numbers
    - Easy: $A = 1_{10}, C = 2_{10}, G = 3_{10}, T = 4_{10}$
      - $\text{GATTACA} = 3144121_{10}$
    - Smart: $A = 00_2, C = 01_2, G = 10_2, T = 11_2$
      - $\text{GATTACA} = 10001111000100_2 = 9156_{10}$
  - Running time
    - Construction: O(n)
    - Lookup: O(1) + O(z)
    - Sorts the genome mers in linear time



AAAAAAA → …

AAAAAAC → …

AAAAAAG → …

…

GATTAAT

GATTACA → 2, 5000, 32000000, …

GATTACC

…

TTTTTTG

TTTTTTT

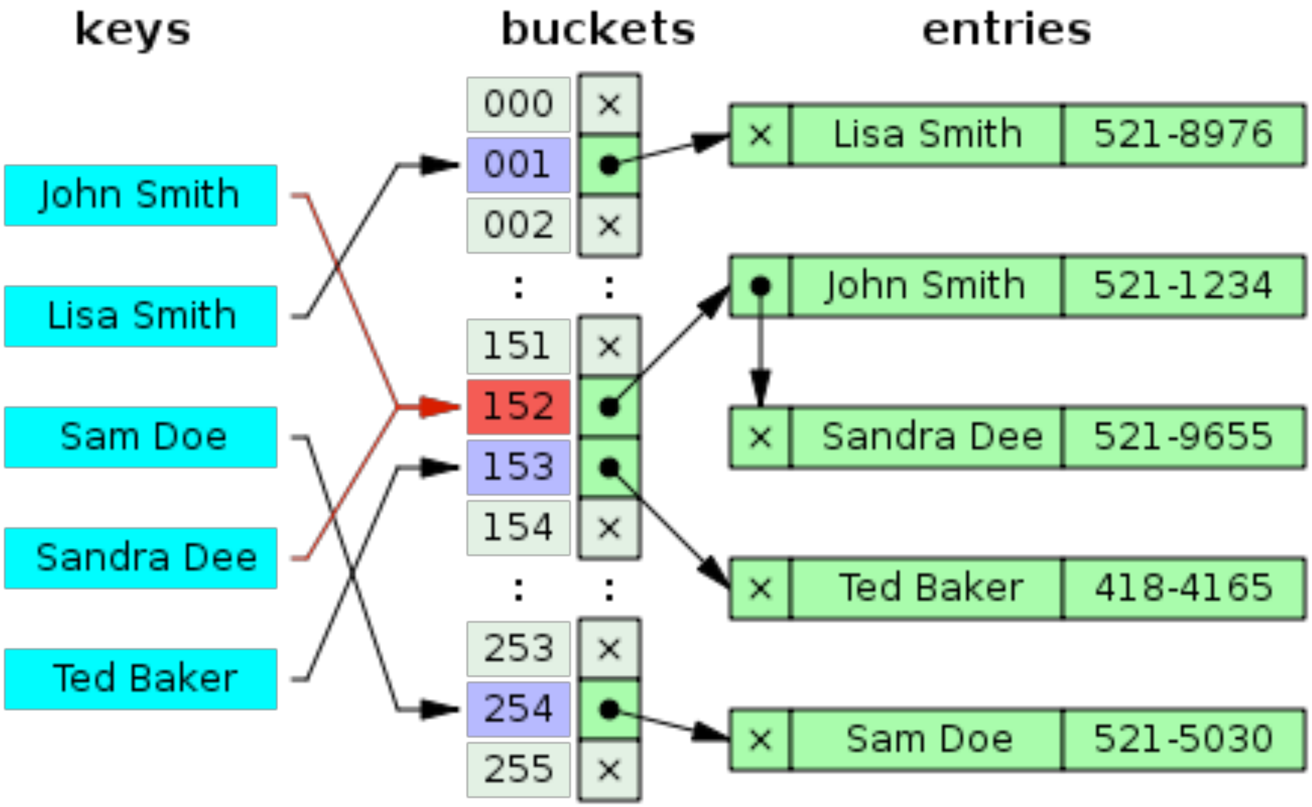
Hash Tables and Hash Functions

- Number of possible sequences of length k = 4^k
  - 4^7 = 16,384 (easy to store)
  - 4^{20} = 1,099,511,627,776 (impossible to directly store in RAM)
    - There are only 3B 20-mers in the genome
      - ⇒ Even if we could build this table, 99.7% will be empty
      - ⇒ But we don't know which cells are empty until we try

- Hash Function: hash(n) -> h
  - Maps a number *n* in [0,R] to h in [0,H] where H << R
    - More than one *n* will have the same *h*
    - A good hash function evenly distributes the values
      - R/H have the same hash value
    - A really good hash function also spreads out the values
      - Pr(hash(n)==hash(m)) = 1/H
    - A common (decent) choice is hash(n) = n mod H

Hash Table Lookup

- By construction, multiple keys have the same hash value
  - Store elements with the same key in a bucket chained together
  - Looking up a value scans the entire bucket
    - Slows down the search as a function of the hash table load
    - Warning: This complexity is usually hidden in the hash table code



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

Variable Length Queries

- Where are GATTACA and GATTACCA in the human genome?
  - s = min(length of all queries)
  - Build an inverted index of all s-mers (seeds) in the genome
    - GATTACA => 2, 5000, 32000000, …
    - GATTACC => 5500, 10101, 1000000, …

- Seed-and-extend to find end-to-end exact matches
  - Check every occurrence of the qry seed (first s characters)
    - ~1 in 4 are GATTACCA, 1 in 4 are GATTACCC, etc
  - The specificity of the seed depends on length(q) & s
    - Works best if max(length) =~ min(length)
    - Works best if e-value(m) is << 1

Exact Matching Review

- E-value depends on length of genome and inversely on query length
  - $E = (n-m+1)/4^m$

Brute Force
(3 GB)

```
BANANA
BAN
 ANA
  NAN
   ANA
```

Naive

Slow & Easy

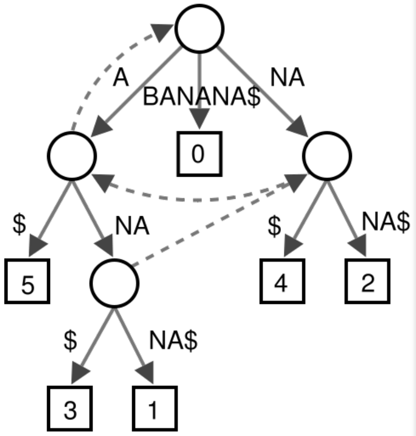
Suffix Array
(>15 GB)

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Vmatch, PacBio Aligner

Binary Search

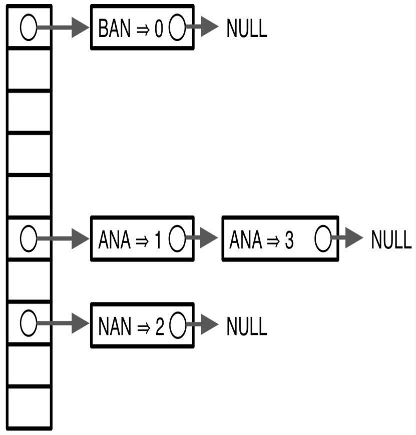
Suffix Tree
(>51 GB)



MUMmer, MUMmerGPU

Tree Walking & DFS

Hash Table
(>15 GB)



BLAST, MAQ, ZOOM, RMAP, CloudBurst

Seed-and-extend

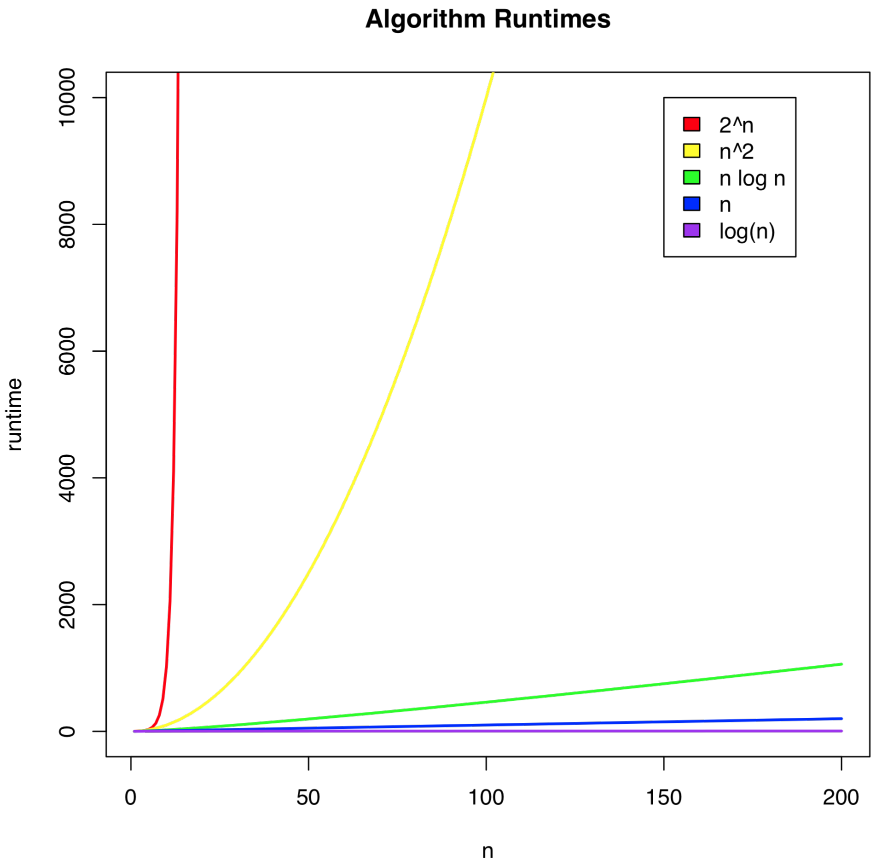
Algorithms Summary

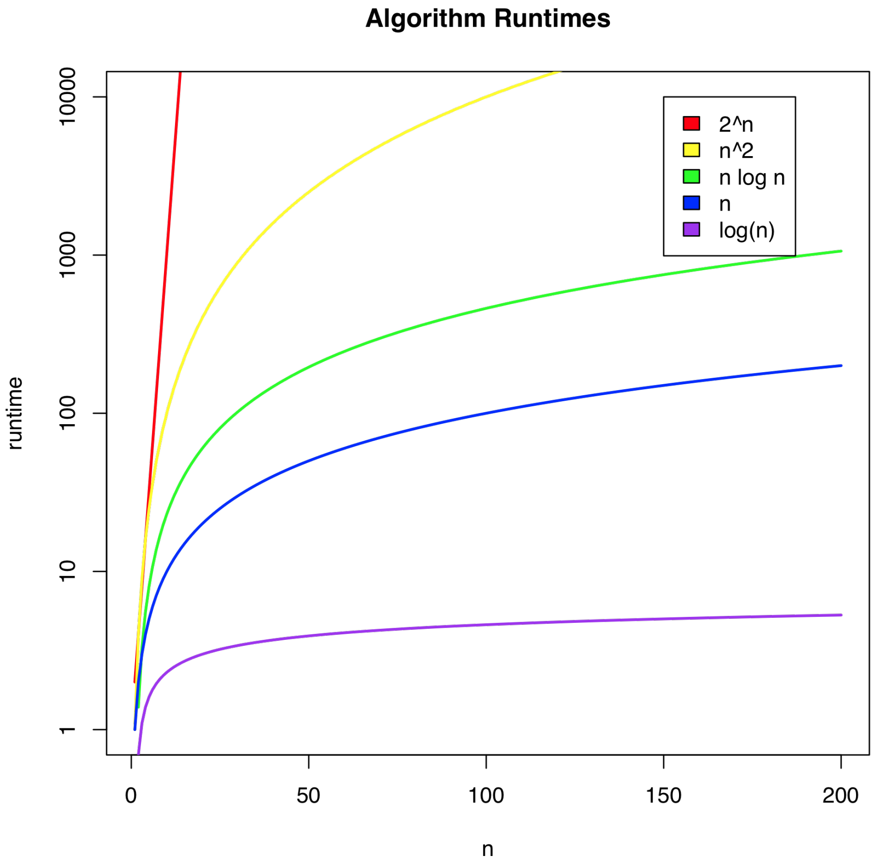
- Algorithms choreograph the dance of data inside the machine
  - Algorithms add provable precision to your method
  - A smarter algorithm can solve the same problem with much less work

- Techniques
  - 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
  - Randomization: Avoid the demon
  - Hashing: Storing sets across a huge range of values
  - Indexing: Focus on the search on the important parts
    - Different indexing schemes have different space/time features

- Data Structures
  - Primitives: Integers, Numbers, Strings
  - Lists / Arrays / Multi-dimensional arrays
  - Trees
  - Hash Table

Algorithmic Complexity





What is the runtime as a function of the input size?

Next Time

- In-exact alignment
  - Smith & Waterman (1981) *Identification of Common Molecular Subsequences*. J. of Molecular Biology. 147:195-197.
- Sequence Homology
  - Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ (1990). *Basic local alignment search tool*. J of Molecular Biology. 215 (3): 403–410.
- Whole Genome Alignment
  - A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O.White, and S.L. Salzberg (1999) *Alignment of Whole Genomes*. Nucleic Acids Research (27):11 2369-2376.
- Short Read Mapping
  - Langmead B, Trapnell C, Pop M, Salzberg SL. (2009) *Ultrafast and memory-efficient alignment of short DNA sequences to the human genome*. Genome Biology. 10:R25.