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

• 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?

<table>
<thead>
<tr>
<th>Brute Force (3 GB)</th>
<th>Suffix Array (&gt;15 GB)</th>
<th>Suffix Tree (&gt;51 GB)</th>
<th>Hash Table (&gt;15 GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BANANA</td>
<td>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BAN</td>
<td>A$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANA</td>
<td>ANA$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NAN</td>
<td>NA$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANA</td>
<td>NANA$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Naive</td>
<td>Vmatch, PacBio Aligner</td>
<td>MUMmer, MUMmerGPU</td>
<td>BLAST, MAQ, ZOOM, RMAP, CloudBurst</td>
</tr>
<tr>
<td>Slow &amp; Easy</td>
<td>Binary Search</td>
<td>Tree Searching</td>
<td>Seed-and-extend</td>
</tr>
</tbody>
</table>
Searching for **GATTACA**

- Where is **GATTACA** in the human genome?
- **Strategy 1: Brute Force**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No match at offset 1
Searching for GATTACA

- Where is GATTACA in the human genome?
- **Strategy 1: Brute Force**

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GATACA</td>
</tr>
</tbody>
</table>

*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 | ...

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 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**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>C</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>T</td>
<td>A</td>
<td>C</td>
<td>A</td>
</tr>
</tbody>
</table>

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 = \( n \)  
  – Query length = \( m \)  
  – Comparisons: \( (n-m+1) \times m \)  

• Overall runtime: \( O(nm) \)
  – If we double genome or query size, takes twice as long
  – If we double both, takes 4 times as long
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)))]);
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
- \( E = \frac{n-m+1}{4^m} \)  

[Challenge Question: What is the expected distribution & variance?]

[Graphs showing e-value and sequence length cutoff 0.1 for human (3B), fly (130M), and E. coli (5M).]
Brute Force Reflections

Why check every position?
- GATTACA can't start at position 15

| 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 $O(n + m)$
  - If we double both, it just takes twice as long
  - Knuth-Morris-Pratt, 1977

- 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

[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;

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACAGATTACC…</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ACC…</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>AGATTACC…</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>ATTACAGATTACC…</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ATTACC…</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>C…</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>CAGATTACC…</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>CC…</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>GATTACAGATTACC…</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>GATTACC…</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>TACAGATTACC…</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>TACC…</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>TGATTACAGATTACC…</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>TTACAGATTACC…</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>TTACC…</td>
<td>11</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACAGATTACC…</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ACC…</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>AGATTACC…</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>ATTACAGATTACC…</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ATTACC…</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>C…</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>CAGATTACC…</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>CC…</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>GATTACAGATTACC…</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>GATTACC…</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>TACAGATTACC…</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>TACC…</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>TGATTACAGATTACC…</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>TTACAGATTACC…</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>TTACC…</td>
<td>11</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACAGATTACC...</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ACC...</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>AGATTACC...</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>ATTACAGATTACC...</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ATTACC...</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>C...</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>CAGATTACC...</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>CC...</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>GATTACAGATTACC...</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>GATTACC...</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>TACAGATTACC...</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>TACC...</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>TGATTACAGATTACC...</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>TTACAGATTACC...</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>TTACC...</td>
<td>11</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACAGATTACC…</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ACC…</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>AGATTACC…</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>ATTACAGATTACC…</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ATTACC…</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>C…</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>CAGATTACC…</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>CC…</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>GATTACAGATTACC…</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>GATTACC…</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>TACAGATTACC…</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>TACC…</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>TGATTACAGATTACC…</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>TTACAGATTACC…</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>TTACC…</td>
<td>11</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACAGATTACC…</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ACC…</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>AGATTACC…</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>ATTACAGATTACC…</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ATTACC…</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>C…</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>CAGATTACC…</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>CC…</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>GATTACAGATTACC…</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>GATTACC…</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>TACAGATTACC…</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>TACC…</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>TGATTACAGATTACC…</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>TTACAGATTACC…</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>TTACC…</td>
<td>11</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACAGATTACC…</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ACC…</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>AGATTACC…</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>ATTACAGATTACC…</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ATTACC…</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>C…</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>CAGATTACC…</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>CC…</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>GATTACAGATTACC…</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>GATTACC…</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>TACAGATTACC…</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>TACC…</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>TGATTACAGATTACC…</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>TTACAGATTACC…</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>TTACC…</td>
<td>11</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>#</th>
<th>Sequence</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACAGATTACC…</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>ACC…</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>AGATTACC…</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>ATTACAGATTACC…</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>ATTACC…</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>C…</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>CAGATTACC…</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>CC…</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>GATTACAGATTACC…</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>GATTACC…</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>TACAGATTACC…</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>TACC…</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>TGATTACAGATTACC…</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>TTACAGATTACC…</td>
<td>4</td>
</tr>
<tr>
<td>15</td>
<td>TTACC…</td>
<td>11</td>
</tr>
</tbody>
</table>
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;
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!
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

- 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: \( n/(2^x) \leq 1 \); \( x = \log_2(n) \)
  - Total Runtime: \( O(m \log n) \)
    - 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;
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

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

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

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

  [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)) \leq 1): return list

  // split list into lo & hi
  pivot = median(list)
  lo = {}; hi = {};
  for (i = 1 to length(list))
    if (list[i] \lt pivot): append(lo, list[i])
    else: append(hi, list[i])
  // recurse on sublists
  return (append(QuickSort(lo), QuickSort(hi))

- Analysis (Assume we can find the median in O(n))
  \[ T(n) = \begin{cases} O(1) & \text{if } n \leq 1 \\ O(n) + 2T(n/2) & \text{else} \end{cases} \]
  \[ T(n) = n + 2\left(\frac{n}{2}\right) + 4\left(\frac{n}{4}\right) + \cdots + n\left(\frac{n}{n}\right) = \sum_{i=0}^{\lg(n)} \frac{2^i n}{2^i} = \sum_{i=0}^{\lg(n)} n = O(n \lg n) \]
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))

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

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

  \[ T(n) = n + 2\left(\frac{n}{2}\right) + 4\left(\frac{n}{4}\right) + \cdots + n\left(\frac{n}{n}\right) = \sum_{i=0}^{\lg(n)} \frac{2^n n}{2^i} = \sum_{i=0}^{\lg(n)} n = O(n \lg n) \]
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**

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

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

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

\[ T(n) = \sum_{i=0}^{\log_{10/9}(n)} n = O(n \log n) \]

- 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 \( O(n \log n) \)?
  • 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

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

```plaintext
for(i = 1 to 100) { cnt[i] = 0; }
for(i = 1 to n) { cnt[list[i]]++; }
for(i = 1 to 100) { while (cnt[i] > 0){print i; cnt[i]--}}
```

[3B instead of 94B]
3. Suffix Trees

- 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

```java
cur = ST.Root;
qrypos = 0;
while (cur)
    edge = cur.getEdge(q[qrypos]);
    dist = matchstrings(edge, 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

```python
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: $O(n)$
- 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
    - Simple: A = 0, C = 1, G = 2, T = 3
      - GATTACA = 2 0 3 3 0 1 0
    - Smart: A = 00₂, C = 01₂, G = 10₂, T = 11₂
      - GATTACA = 10 00 11 11 00 01 00₂ = 9156₁₀

- Running time
  - Construction: O(n)
  - Lookup: O(1) + O(z)
  - Sorts the genome mers in linear time
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

• Use a hash function to shrink the possible range
  – Maps a number $n$ in $[0,R]$ to $h$ in $[0,H]$
    » Use 128 buckets instead of 16,384, or 10B instead of 1T
  – Division: $\text{hash}(n) = H * n / R$;
    » $\text{hash}(\text{GATTACA}) = 128 * 9156 / 16384 = 71$
  – Modulo: $\text{hash}(n) = n \% H$
    » $\text{hash}(\text{GATTACA}) = 9156 \% 128 = 68$

[Why would we want different functions?]
Hash Table Lookup

- By construction, multiple keys have the same hash value
  - Store elements with the same key in a bucket chained together
    - A good hash evenly distributes the values: R/H have the same hash value
  - 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

[Diagram showing hash table lookup with examples]

[How many elements do we expect per bucket?]
Variable Length Queries

- Where are GATTACA and GATTACCA in the human genome?
  - $s = \text{min(length of all queries)}$
  - 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 $\text{length(q)}$ & $s$
    - Works best if $\text{max(length)} \approx \text{min(length)}$
    - Works best if $\text{e-value(m)}$ is $\ll 1$
Exact Matching Review

- E-value depends on length of genome and inversely on query length
  - \( E = \frac{n-m+1}{4^m} \)

<table>
<thead>
<tr>
<th>Method</th>
<th>Space Requirement</th>
<th>Examples</th>
<th>Search Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brute Force (3 GB)</td>
<td></td>
<td>BANANA, BAN, ANA, NAN, ANA</td>
<td>Naive, Slow &amp; Easy</td>
</tr>
<tr>
<td>Suffix Array (&gt;15 GB)</td>
<td></td>
<td>$6$, 5, 3, 1, 0, 4, 2</td>
<td>Vmatch, PacBio Aligner</td>
</tr>
<tr>
<td>Suffix Tree (&gt;51 GB)</td>
<td></td>
<td>A, $A$, ANA$, ANANA$, BANANA$, NA$, NANA$</td>
<td>MUMmer, MUMmerGPU</td>
</tr>
<tr>
<td>Hash Table (&gt;15 GB)</td>
<td></td>
<td></td>
<td>BLAST, MAQ, ZOOM, RMAP, CloudBurst</td>
</tr>
</tbody>
</table>

- 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

• Sequence Homology

• Whole Genome Alignment

• Short Read Mapping