Algorithm analysis

- Determine the amount of resources an algorithm requires to run
  - computation time, space in memory
- Running time of an algorithm is the number of basic operations performed
  - additions, multiplications, comparisons
  - usually grows with the size of the input
  - faster to add 2 numbers than to add 2,000,000!

Running time

- Worst-case running time
  - upper bound on the running time
  - guarantee the algorithm will never take longer to run
- Average-case running time
  - time it takes the algorithm to run on average
    (expected value)
- Best-case running time
  - lower bound on the running time
  - guarantee the algorithm will not run faster

Comparisons in insertion sort

- Worst case
  - element k requires \((k-1)\) comparisons
  - total number of comparisons:
  \[
  0+1+2+ \ldots + (n-1) = \frac{1}{2} (n)(n-1)
  \]
  \[
  = \frac{1}{2} (n^2-n)
  \]
- Best case
  - elements 2 through \(n\) each require one comparison
  - total number of comparisons:
  \[
  1+1+1+ \ldots + 1 = n-1
  \]
  \[
  (n-1) \text{ times}
  \]

Running time of insertion sort

- Best case running time is linear
- Worst case running time is quadratic
- Average case running time is also quadratic
- on average element \(k\) requires \((k-1)/2\) comparisons
- total number of comparisons:
  \[
  \frac{1}{2} (0+1+2+ \ldots + n-1) = \frac{1}{4} (n)(n-1)
  \]
  \[
  = \frac{1}{4} (n^2-n)
  \]

Mergesort
Merging two sorted lists

<table>
<thead>
<tr>
<th>first list</th>
<th>second list</th>
<th>result of merge</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 27</td>
<td>10 27</td>
<td>10</td>
</tr>
<tr>
<td>10 27</td>
<td>12 20</td>
<td>10 12</td>
</tr>
<tr>
<td>10 27</td>
<td>12 20</td>
<td>10 12 20</td>
</tr>
<tr>
<td>10 27</td>
<td>12 20</td>
<td>10 12 20 27</td>
</tr>
</tbody>
</table>

Comparisons in merging

- Merging two sorted lists of size \( m \) requires at least \( m \) and at most \( 2m-1 \) comparisons
  - \( m \) comparisons if all elements in one list are smaller than all elements in the second list
  - \( 2m-1 \) comparisons if the smallest element alternates between lists

Logarithm

- Power to which any other number \( a \) must be raised to produce \( n \)
  - \( a \) is called the base of the logarithm
- Frequently used logarithms have special symbols
  - \( \lg n = \log_{10} n \) logarithm base 2
  - \( \ln n = \log_n n \) natural logarithm (base \( e \))
  - \( \log n = \log_{10} n \) common logarithm (base 10)
- If we assume \( n \) is a power of 2, then the number of times we can recursively divide \( n \) numbers in half is \( \lg n \)

Comparisons at each merge

<table>
<thead>
<tr>
<th>#lists</th>
<th>#elements in each list</th>
<th>#merges</th>
<th>#comparisons per merge</th>
<th>#comparisons total</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>1</td>
<td>( n/2 )</td>
<td>1</td>
<td>( n/2 )</td>
</tr>
<tr>
<td>( n/2 )</td>
<td>2</td>
<td>( n/4 )</td>
<td>3</td>
<td>( 3n/4 )</td>
</tr>
<tr>
<td>( n/4 )</td>
<td>4</td>
<td>( n/8 )</td>
<td>7</td>
<td>( 7n/8 )</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>2</td>
<td>( n/2 )</td>
<td>1</td>
<td>( n-1 )</td>
<td>( n-1 )</td>
</tr>
</tbody>
</table>

Comparisons in mergesort

- Total number of comparisons is the sum of the number of comparisons made at each merge
  - at most \( n \) comparisons at each merge
  - the number of times we can recursively divide \( n \) numbers in half is \( \lg n \), so there are \( \lg n \) merges
  - there are at most \( n \lg n \) comparisons total

Comparison of sorting algorithms

- Best, worst and average-case running time of mergesort is \( \Theta(n \lg n) \)
- Compare to average case behavior of insertion sort:

<table>
<thead>
<tr>
<th>( n )</th>
<th>Insertion sort</th>
<th>Mergesort</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>25</td>
<td>33</td>
</tr>
<tr>
<td>100</td>
<td>2500</td>
<td>664</td>
</tr>
<tr>
<td>1000</td>
<td>2500000</td>
<td>9965</td>
</tr>
<tr>
<td>10000</td>
<td>2500000000</td>
<td>132877</td>
</tr>
<tr>
<td>100000</td>
<td>250000000000</td>
<td>1660960</td>
</tr>
</tbody>
</table>
Quicksort

- Most commonly used sorting algorithm
- One of the fastest sorts in practice
- Best and average-case running time is $O(n \log n)$
- Worst-case running time is quadratic
- Runs very fast on most computers when implemented correctly

Searching

- Determine the location or existence of an element in a collection of elements of the same type
- Easier to search large collections when the elements are already sorted
  - finding a phone number in the phone book
  - looking up a word in the dictionary
- What if the elements are not sorted?

Sequential search

- Given a collection of $n$ unsorted elements, compare each element in sequence
- Worst-case: Unsuccessful search
  - search element is not in input
  - make $n$ comparisons
  - search time is linear
- Average-case:
  - expect to search $\frac{n}{2}$ the elements
  - make $n/2$ comparisons
  - search time is linear

Searching sorted input

- If the input is already sorted, we can search more efficiently than linear time
- Example: “Higher-Lower”
  - think of a number between 1 and 1000
  - have someone try to guess the number
  - if they are wrong, you tell them if the number is higher than their guess or lower
- Strategy?
- How many guesses should we expect to make?

Best Strategy

- Always pick the number in the middle of the range
- Why?
  - you eliminate half of the possibilities with each guess
- We should expect to make at most $\log_{10} 1000 = 10$ guesses
- Binary search
  - search $n$ sorted inputs in logarithmic time
Binary search

- Search for 9 in a list of 16 elements

Sequential vs. binary search

- Average-case running time of sequential search is linear
- Average-case running time of binary search is logarithmic
- Number of comparisons:

```
<table>
<thead>
<tr>
<th>n</th>
<th>sequential search</th>
<th>binary search</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>256</td>
<td>128</td>
<td>8</td>
</tr>
<tr>
<td>4096</td>
<td>2048</td>
<td>12</td>
</tr>
<tr>
<td>65536</td>
<td>32768</td>
<td>16</td>
</tr>
</tbody>
</table>
```