Design and Analysis of Algorithms

- Understanding of Algorithm
- Upper bounds on specific solutions
- Lower bounds on problems

Algorithms – What Are They?

**Definition.** An __________ is a well-defined computational procedure that transforms inputs into outputs, achieving the desired input-output relationship.

**Definition.** A computational __________ is a specification of the desired input-output relationship.

**Definition.** An __________ of a problem is all the inputs needed to compute a solution to the problem.

**Definition.** A __________ algorithm halts with the correct output for every input instance. We can then say the algorithm __________ the problem.

Example: Sorting

Sorting is a common operation
Many sorting algorithms available, best choice depends on application

**PROBLEM:**

- INPUT: Sequence of \( n \) numbers \( \langle a_1, a_2, \ldots, a_n \rangle \)
- OUTPUT: Permutation (reordering) \( \langle a'_1, a'_2, \ldots, a'_n \rangle \) of the input sequence such that \( a'_1 \leq a'_2 \leq \ldots \leq a'_n \)

**INSTANCE:** \( \langle 6, 4, 3, 7, 1, 4 \rangle \rightarrow \langle 1, 3, 4, 4, 6, 7 \rangle \)

---

**Algorithm: Insertion Sort**

In-Place Sort: uses only a fixed amount of storage beyond that needed for the data.

Pseudocode:

```plaintext
Insertion-Sort(A) ; A is an array of numbers
1 for j = 2 to length(A)
2     key = A[j]
3     i = j - 1
4     while i > 0 and A[i] > key
5         A[i+1] = A[i]
6         i = i - 1
7     A[i+1] = key
```
Analyzing Algorithms

- Predict resource utilization

- Dependent on architecture
  
  Model of computation
  Sequential (RAM model)
  Parallel (PRAM model)

- Running Time = F(Problem Size)
  = F(Input Size)
  = number of primitive operations used to solve problem

- Input Size:
  Sorting: 
  Multiplication: 
  Graphs: 

Operations

Examples: additions, multiplications, comparisons

Constant time $C_i$ per $i^{th}$ line of pseudocode
In reality each operation takes different amount of time

________ constraints on the input, other than size, resulting in
the fastest possible running time

________ constraints on the input, other than size, resulting in
the slowest possible running time

________ average running time over every possible type of
input (usually involves probabilities of different types of input)

---

**Example: Insertion Sort**

- \( n = \text{length}(A) \)
- \( t_j = \text{number of times the while loop test is executed for that value of} \ j \)

**Insertion-Sort(A)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>for ( j ) = 2 to length(A)</td>
</tr>
<tr>
<td>2</td>
<td>key = ( A[j] )</td>
</tr>
<tr>
<td>3</td>
<td>( i = j - 1 )</td>
</tr>
<tr>
<td>4</td>
<td>while ( i &gt; 0 ) and ( A[i] &gt; \text{key} )</td>
</tr>
<tr>
<td>5</td>
<td>( A[i+1] = A[i] )</td>
</tr>
<tr>
<td>6</td>
<td>( i = i - 1 )</td>
</tr>
<tr>
<td>7</td>
<td>( A[i+1] = \text{key} )</td>
</tr>
</tbody>
</table>

**Cost Times (Iterations)**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 )</td>
<td>( n )</td>
</tr>
<tr>
<td>( c_2 )</td>
<td>( n - 1 )</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>( n - 1 )</td>
</tr>
<tr>
<td>( c_4 )</td>
<td>( \sum_{j=2}^{n} t_j )</td>
</tr>
<tr>
<td>( c_5 )</td>
<td>( \sum_{j=2}^{n} t_j - 1 )</td>
</tr>
<tr>
<td>( c_6 )</td>
<td>( \sum_{j=2}^{n} t_j - 1 )</td>
</tr>
<tr>
<td>( c_7 )</td>
<td>( n - 1 )</td>
</tr>
</tbody>
</table>

**Analysis**

\[
T(n) = c_1 n + c_2(n - 1) + c_3(n - 1) + c_4 \sum_{j=2}^{n} t_j + c_3 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n - 1)
\]
Best Case: Array already sorted, \( t_j = 1 \) for all \( j \)

\[
T(n) = c_1n + c_2(n - 1) + c_3(n - 1) + c_4(n - 1) + c_5(0) + c_6(0) \\
     + c_7(n - 1) \\
= (c_1 + c_2 + c_3 + c_4 + c_7) \cdot n - (c_2 + c_3 + c_4 + c_7) \\
= an + b \quad \text{(linear in } n)\]

Worst Case: Array in reverse order, \( t_j = j \) for all \( j \)
Note that \( \sum_{j=1}^{n} j = \frac{n(n+1)}{2} \)

\[
T(n) = c_1n + c_2(n - 1) + c_3(n - 1) + c_4\left(\frac{n(n + 1)}{2} - 1\right) + c_5\frac{n(n - 1)}{2} \\
     + c_6\frac{n(n - 1)}{2} + c_7(n - 1) \\
= (c_4/2 + c_5/2 + c_6/2)n^2 \\
     + (c_1 + c_2 + c_3 + c_4/2 - c_5/2 - c_6/2 + c_7)n \\
     - (c_2 + c_3 + c_4 + c_7) \\
= an^2 + bn + c \quad \text{(quadratic in } n)\]

Average Case: Check half of array on average, \( t_j = j/2 \) for all \( j \)

\[T(n) = an^2 + bn + c\]
Analysis

- Concentrate on worst-case running time
- Upper bound
- Average case not much better

Order of Growth

The _______ of a running-time function $\Theta$ is the fastest growing term, discarding constant factors.

**Insertion Sort:**

- Best Case: $an + b \rightarrow \Theta(n)$
- Worst Case: $an^2 + bn + c \rightarrow \Theta(n^2)$

**XYZ-Sort:**

- Worst Case: $\Theta(n^3)$
- Now we can say Insertion-Sort better than XYZ-Sort for large inputs.

Designing Algorithms

- Incremental Design
  - Iterative
<table>
<thead>
<tr>
<th>Complexity</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>.00001 s</td>
<td>.00002 s</td>
<td>.00003 s</td>
<td>.00004 s</td>
<td>.00005 s</td>
</tr>
<tr>
<td>$n^2$</td>
<td>.0001 s</td>
<td>.0004 s</td>
<td>.0009 s</td>
<td>.0016 s</td>
<td>.0025 s</td>
</tr>
<tr>
<td>$n^3$</td>
<td>.001 s</td>
<td>.008 s</td>
<td>.027 s</td>
<td>.064 s</td>
<td>.125 s</td>
</tr>
<tr>
<td>$n^5$</td>
<td>.1 s</td>
<td>3.2 s</td>
<td>24.3 s</td>
<td>1.7 min</td>
<td>5.2 min</td>
</tr>
<tr>
<td>$2^n$</td>
<td>.001 s</td>
<td>1.0 s</td>
<td>17.9 min</td>
<td>12.7 days</td>
<td>35.7 years</td>
</tr>
<tr>
<td>$3^n$</td>
<td>.059 s</td>
<td>58 min</td>
<td>6.5 years</td>
<td>3855 centuries</td>
<td>$2 \times 10^8$ centuries</td>
</tr>
</tbody>
</table>

- Example: Insertion Sort

- Divide-and-Conquer
  - Recursive
  - Example: Merge Sort
    - 1. __________ problem into smaller subproblems
    - 2. __________ subproblems by solving them recursively
    - 3. __________ solutions of subproblems

---

**Example: Merge Sort**

1. __________ the n element sequence to be sorted into two subsequences of n/2 elements each.

2. __________ (sort) the two subsequences recursively using merge sort
3. ________ (merge) the two sorted subsequences to produce the sorted answer

---

**Example: Merge Sort**

Recursion bottoms out when subproblem contains only one element \((p = r)\)

```
MergeSort(A,p,r)
1   if p < r
2     then q = ⌊(p+r)/2⌋
3     MergeSort(A,p,q)
4     MergeSort(A,q+1,r)
5     Merge(A,p,q,r)
```

---

**Example: Merge Sort**

Merge\((A,p,q,r)\) procedure has \(\Theta(n)\) running time, \(n = r - p + 1\)

```
Merge(A,p,q,r) ; Exercise 1.3-2
1   for j = p to r
3   i = p
4   w = q + 1
5   while (p ≤ q) and (w ≤ r)
6     if B[p] ≤ B[w]
7       then A[i] = B[p]
8           p = p + 1
9       else A[i] = B[w]
```
10       \( w = w + 1 \)
11       \( i = i + 1 \)
12       if \( p > q \)
13       then \( p = w \)
14       \( q = r \)
15       for \( j = p \) to \( q \)
16       \( A[i] = B[j] \)
17       \( i = i + 1 \)

---

**Analyzing Divide-and-Conquer Algorithms**

Definition: A ________________ or _______________ describes the running time of a recursive algorithm on a problem of size \( n \) in terms of the running time of the algorithm on smaller inputs.

For small enough input size \( (n \leq c, \text{for example}) \), running time is constant.

- Example: \( T(n) = \Theta(n^0) = \Theta(1) \)
- Sorting one element in Merge Sort

---

**For larger input size:**

- \( D(n) = \__________ \)
  
  each of size ________
Thus we generate the recurrence:

\[
T(n) = \begin{cases} 
\Theta(1) & \text{if } n \leq c \\
aT(n/b) + D(n) + C(n) & \text{otherwise}
\end{cases}
\]

---

**Analysis of Merge Sort**

Note if \( n = 1 \), then \( T(n) = \Theta(1) \)

1. **DIVIDE** computes middle of array in constant time: \( D(n) = \Theta(1) \)
2. **CONQUER** sorts 2 subarrays of size \( n/2 \) in time: \( 2T(n/2) \)
3. **COMBINE** (Merge) procedure: \( C(n) = \Theta(n) \)

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

Note that \( \Theta(n) \) dominates \( \Theta(1) \).

We will show later that \( T(n) = \Theta(n \log n) \), where \( \log = \log_2 n \).

---

**Summary**

Does the computational difference amount to much?

- Fast computer (1 GHz) running efficient insertion sort
- $10^9$ instructions per second
- $2n^2$ instructions to sort $n$ numbers

- Slow computer running inefficient merge sort
  - $100 \times 10^6$ instructions per second
  - $50n \lg n$ instructions to sort $n$ numbers

- Sorting one million numbers ($n = 10^6$)

- Fast computer: \[
\frac{2(10^6)^2 \text{ instructions}}{10^9 \text{ instructions/sec}} = \]

- Slow Computer: \[
\frac{50(10^6) \lg(10^6) \text{ instructions}}{10^8 \text{ instructions/sec}} = \]

- MORAL: A little computational shrewdness can go a long way.

---

**Growth of Functions**

Definition: The ___________ efficiency of an algorithm is the order of growth of the algorithm as the input size approaches the limit (increases without bound).

The asymptotically more efficient algorithm is usually the better choice for all but small inputs.

---

**Theta**

$f(n) = \Theta(g(n))$, $g(n)$ is the Asymptotically Tight Bound for $f(n)$
\( \Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \} \)

“\( f(n) = \Theta(g(n)) \)” means \( f(n) \) is an element of the set of functions \( g(n) \).

Here is a graphical depiction of \( \Theta \).

Click mouse to advance to next frame.

---

**Example:**

Show that \( n^2 - 2n = \Theta(n^2) \) and \( 200n^2 - 100n = \Theta(n^2) \).

\[
\begin{align*}
c_1 n^2 & \leq n^2 - 2n \leq c_2 n^2 & & c_1 n^2 & \leq 200n^2 - 100n \leq c_2 n^2 \\
c_1 & \leq 1 - 2/n \leq c_2 & & c_1 & \leq 200 - 100/n \leq c_2 \\
c_1 & \leq 1/3, \quad c_2 \geq 1 & & c_1 & \leq 100, \quad c_2 \geq 200 \\
n \geq 3, \quad n \geq 1 & & n \geq 1, \quad n \geq 1 & & n_0 = 3, \quad n_0 = 1
\end{align*}
\]

Because some choice for \( c_1, c_2, \) and \( n_0 \) exists, then the functions are both \( \Theta(n^2) \).

Because coefficients on the high-order term only affect constants, they are dropped from the \( \Theta \) notation.

\( \Theta(n^0) = \Theta(1) \) constant
**O(g(n))**

\[ f(n) = O(g(n)), \text{ g(n) is an Asymptotic Upper Bound for f(n)} \]

\[ O(g(n)) = \{ f(n) : \text{there exist positive constants c and } n_0 \text{ such that } 0 \leq f(n) \leq c \cdot g(n) \text{ for all } n \geq n_0 \} \]

See Figure 3.1b, page 43, for graphical depiction of \( O \).

**Examples:**

- \( n^2 - 2n = O(n^2) \)

- \( 200 \ n^2 - 100n = O(n^2) = O(n^3) = ... \)

- \( n = O(n^2) \)

- Does \( f(n) = \Theta(g(n)) \) imply \( f(n) = O(g(n)) \)? ___

- Does \( f(n) = O(g(n)) \) imply \( f(n) = \Theta(g(n)) \)? ___

---

**Omega(g(n))**

\[ f(n) = \Omega(g(n)), \text{ g(n) is an Asymptotic Lower Bound for f(n)} \]

\[ \Omega(g(n)) = \{ f(n) : \text{there exist positive constants c and } n_0 \text{ such that } 0 \leq c \cdot g(n) \leq f(n) \text{ for all } n \geq n_0 \} \]

See Figure 3.1c, page 43, for graphical depiction of \( \Omega \).

Examples:

- \( n^2 - 2n = \Omega(n^2) \)
• $200 \ n^2 - 100 \ n = \Omega(n^2) = \Omega(n) = \Omega(1)$

• $n^2 = \Omega(n)$

• Does $f(n) = \Theta(g(n))$ imply $f(n) = \Omega(g(n))$? ________

• Does $f(n) = \Omega(g(n))$ imply $f(n) = \Theta(g(n))$? ________

---

**Theorem 2.1**

$f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

---

**Asymptotic Notation in Equations**

**Example:** $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$

In other words, there is some function $f(n)$ element of $\Theta(n)$ that makes the equation true; namely, $3n + 1$.

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n) = \Theta(n^2).$$

In other words, for any function $f(n)$ element of $\Theta(n)$, there is some function $h(n)$ element of $\Theta(n^2)$; namely, $2n^2 + f(n)$.

---

**Summary**

**o notation** $f(n) = o(g(n))$, $g(n)$ is an upper bound of $f(n)$ that is not asymptotically tight
\( \omega \) notation \( f(n) = \omega(g(n)) \), \( g(n) \) is a lower bound of \( f(n) \) that is not asymptotically tight

Example: \( f(n) = 3n^3 + 4 \)

- \( f(n) = \Theta(n^3) \)
- \( f(n) = O(n^3) = O(n^4) = ... \)
- \( f(n) = \Omega(n^3) = \Omega(n^2) = \Omega(n) = \Omega(1) \)
- \( f(n) = o(n^4) = o(n^5) = ... \)
- \( f(n) = \omega(n^2) = \omega(n) = \omega(1) \)

---

**Some Useful Mathematical Tools To Remember**

- monotonically increasing, strictly increasing
- monotonically decreasing, strictly decreasing
- \( \lg n = \log_2 n = \) binary logarithm
- \( \ln n = \log_e n = \) natural logarithm, \( e = 2.7182... \)
- \( \log_b a^n = n\log_b a \)
- \( \log_c (ab) = \log_c a + \log_c b \)
- \( a = b^{\log_b a} \)
- \( a^{\log_b n} = n^{\log_b a} \)
- \( \log_a b = \frac{\log_c b}{\log_c a} \)
- Stirling’s approximation to \( n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n (1 + \Theta(1/n)) \)
- Splitting summations
• Mathematical Induction
• Review other tools starting on page 51