Chapter 3

The Complexity of Algorithms and Problems
Analysis of algorithms

• How to analyze?
  – Empirical approach
    • actually running time.
  – Theoretical approach
    • determining mathematically the quantity of resources needed by each algorithm as a function of the size of the instances considered.

• How to measure?
  – Use a particular step or an elementary operation.
  – Examples:

• What measurements?
  – $O$-notation: asymptotically upper bound
  – $\Omega$-notation: asymptotically lower bound
  – $\Theta$-notation: asymptotically tight bound

<table>
<thead>
<tr>
<th>problem</th>
<th>operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>find x in a list of names</td>
<td>comparison</td>
</tr>
<tr>
<td>multiply two matrices</td>
<td>multiplication</td>
</tr>
<tr>
<td>Sort n integers</td>
<td>comparison/data movement</td>
</tr>
</tbody>
</table>
• **Question:**
  - Let $A_1$ and $A_2$ be two algorithms that solve the same problem. Let the time complexity of $A_1$ and $A_2$ be $O(n^2)$ and $O(n)$ respectively.
  - Would the program for $A_2$ run faster than that of $A_1$?

• **Answer:**
  - Not exactly!

• **Example:**
  - $A_1 : n^2$
  - $A_2 : 100n$

**Note:**

$100n > n^2$, for $n < 100$
### Complexity

- How important is order?

<table>
<thead>
<tr>
<th>functions</th>
<th>problem size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$10$</td>
</tr>
<tr>
<td>$\lg n$</td>
<td>3.3</td>
</tr>
<tr>
<td>$n$</td>
<td>10</td>
</tr>
<tr>
<td>$n \lg n$</td>
<td>$0.33 \times 10^2$</td>
</tr>
<tr>
<td>$2^n$</td>
<td>1024</td>
</tr>
<tr>
<td>$n!$</td>
<td>$3 \times 10^6$</td>
</tr>
</tbody>
</table>
What complexities?

• What complexity do we need?
  – Best-case complexity
  – Worst-case complexity
  – Average-case complexity

• Definition:
  – Let $D_n$ be the set of inputs of size $n$ for the problem under consideration, and let $I$ be an element of $D_n$.
  – Let $t(I)$ be the number of basic operations performed by the algorithm on input $I$.
  – Worst-Case Complexity: $W(n) = \max \{ t(I) | I \in D_n \}$
  – Best-Case Complexity: $B(n) = \min \{ t(I) | I \in D_n \}$
  – Average-Case Complexity: $A(n)$
    • $p(I)$ : probability that input $I$ occurs.
    $$A(n) = \sum_{I \in D_n} p(I) \cdot t(I)$$
Example of analysis

- Sequential search:
  - Problem: Find x in list L.
  - Basic operation: Comparison of x with a list entry.

- Analysis:
  - Best case:
    - $B(n) = 1$
  - Worst case:
    - $W(n) = n$
  - Average case:
    - Assume all elements are distinct.
    - Case 1: x is in L
    - Case 2: x is not in L
Example of analysis

- Case 1: Suppose \( x \) is in \( L \)
  - \( l_i \): represent the case where \( x \) appears in the \( i \)-th position in \( L \).
  - \( t(l) \): the # of comparisons.
  - \( p(l_i) \): prob. that \( l_i \) occurred.
  - \( t(l_i) = i \), for \( 1 \leq i \leq n \).

\[
A(n) = \sum_{i=1}^{n} p(l_i) t(l_i) = \sum_{i=1}^{n} \left(\frac{1}{n}\right) i
\]

\[
= \left(\frac{1}{n}\right) \sum_{i=1}^{n} i = \left(\frac{1}{n}\right)(n(n+1)/2) = (n+1)/2
\]
Example of analysis

• Case 2: x is not in L
  – \((n+1)\) inputs should be considered.
  – \(I_{n+1}\): represent the case where \(x\) is not in \(L\).
  – \(q\): prob. that \(x\) is in \(L\).
  – \(p(I_i) = q/n, \text{ for } 1 \leq i \leq n\)
  – \(p(I_{n+1}) = 1 - q\).

\[
A(n) = \sum_{i=1}^{n+1} p(I_i) t(I_i)
\]

\[
= \sum_{i=1}^{n} \left(\frac{q}{n}\right)^i \left(1-q\right)n = \frac{q}{n} (n(n+1))/2 + (1-q) n
\]

\[
= q((n+1)/2) + (1-q)n
\]

Note:

When \(q=1\), \(A(n) = (n+1)/2\)

When \(q=1/2\), \(A(n) = (n+1)/4 + n/2\)
Straight insertion sort:
- Input: $x_1, x_2, ..., x_n$.
- Output: Sorted sequence $x_1, x_2, ..., x_n$.
- Algorithm:

For $j = 2$ to $n$ do
Begin
  $i = j - 1$;  \hspace{1cm} x = x_i$
  while $x < x_i$ and $i > 0$ do
  Begin
    $x_{i+1} = x_i$;  \hspace{1cm} i = i - 1;
  End
  $x_{i+1} = x$
End.

Example:

```
Initial: 2 1 8 3 4 7 6 5
Sorted: 1 2 3 8
Not yet examined: 4 7 6 5
```

Step:
```
1 2 3 8
x = 4
```

Result:
```
1 2 3 8
7 6 5
```

Sorted sequence: 1 2 3 4 5 6 7 8
Analysis of Insertion sort

• Worst Case:

  1  2  3  ...  i-1
  i-1 i-1 i-2 2 1

  x = x_i

• each x_i needs i-1 comparisons

\[
W(n) = \sum_{i=1}^{n}(i-1) = \frac{n(n-1)}{2} = O(n^2)
\]
Analysis of Insertion sort

- **Average Case:**
  - the prob. that x belongs in any one specific position is $1/i$.

Average number of comparisons to insert x is:

\[
\sum_{i=2}^{n} \left( \frac{i-1}{i} \cdot j + \frac{1}{i} \right) = \sum_{i=2}^{n} \frac{1}{i} (i-1) = \sum_{i=2}^{n} \frac{1}{i} \left( j + i - \frac{1}{i} \right) = \frac{(i+1)}{2} - \frac{1}{i}
\]

\[
\Rightarrow A(n) = \sum_{i=2}^{n} \left[ \frac{(i+1)}{2} - \frac{1}{i} \right] = \frac{n^2}{4} + \frac{3n}{4} - 1 - \sum_{i=2}^{n} \frac{1}{i}
\]

\[
\Rightarrow A(n) \approx \frac{n^2}{4} = O(n^2)
\]
Bubble sort

• Bubble sort:
  – Input: $x_1, x_2, ..., x_n$.
  – Output: Sorted sequence $x_1, x_2, ..., x_n$.
  – Algorithm:

```plaintext
k = n;  FLAG = 1;
while FLAG > 0 do
  Begin
    k = k - 1;  FLAG = 0;
    for j = 1 to k do
      if $x_j > x_{j+1}$ then
        Begin
          $x_j <=> x_{j+1}$;  FLAG = 1;
        End
  End.
```

Example:

<table>
<thead>
<tr>
<th>flag</th>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7 9 4 3 8</td>
</tr>
<tr>
<td>1</td>
<td>7 9 4 8 3</td>
</tr>
<tr>
<td>1</td>
<td>9 7 8 4 3</td>
</tr>
<tr>
<td>0</td>
<td>9 8 7 4 3</td>
</tr>
</tbody>
</table>
Analysis of Bubble sort

• **Worst case:**
  – for each iteration i, it needs n-i comparisons.
  – \( W(n) = \sum(n-i) = n(n-1)/2 = O(n^2) \)

• **Average case:**
  – \( A(n) = n(n-1)/4 \)

• **Best case:**
  – n-1
Lower bounds of sorting algorithms

Analysis:
- Suppose all the numbers are the integers 1, 2, ..., n.
- \( \pi : \) a permutation on \{1, 2, ..., n\}, denoted as \( x_1, x_2, ... x_n \).
- \( \pi(i) : \) correct position of value \( x_i \) in the sorted sequence

Example:
\[
\pi : \begin{array}{cccccc}
2 & 4 & 1 & 5 & 3 \\
\hline
x1 & x2 & x3 & x3 & x5 \\
\pi(1) & \pi(2) & \pi(3) & \pi(4) & \pi(5) \\
\end{array}
\]

Definition:
- Inversion : a pair \((p(i),p(j))\) s.t. \(i<j\) and \(p(i)>p(j)\).
- I.e. \( x_i \) and \( x_j \) are out of order

Example:
- 2, 4, 1, 5, 3
- inversions: (2,1) (4,1) (4,3) (5,3)
Lower bounds of sorting algorithms

• Discussion:
  – If a sorting algorithm removes at most one inversion after each key comparison, then the number of comparisons performed on input p(1), p(2), ..., p(n) \((x_1, x_2, ..., x_n)\) is at least the number of inversion of p.
  – There is a permutation that has \(n(n-1)/2\) inversions.
    • Which permutation?
  – Thus, the worst-case behavior of any sorting algorithm that removes at most one inversion per key comparison must be in \(\Omega(n^2)\).

• Note:

lower bound on the average number of comparisons \(\Leftrightarrow\) average number of inversions in permutations
Lower bounds of sorting algorithms

• Discussion:
  – Transpose permutation: 2 4 1 5 3 \(\leftrightarrow\) 3 5 1 4 2
  – Any pair \((i,j)\) is an inversion in exactly one of the permutations \(\pi\) and transpose of \(\pi\).
  – There are \(n(n-1)/2\) inversions.
  – Each pair of permutation \((\pi\) and transpose of \(\pi)\) has \(n(n-1)/2\) inversions between them.
  – Average: \(n(n-1)/4\) inversions.

• Theorem:
  – Any algorithm that sorts by comparison of keys and removes at most one inversion after each comparison must do at least \(n(n-1)/2\) comparisons in the worst case and at least \(n(n-1)/4\) comparisons on the average.
Quick sort

• Basic idea:

\[ x \]

\[ \downarrow \text{split} \]

\[
\begin{array}{ccc}
\leq x & x & \geq x \\
\end{array}
\]

sort recursively by Quicksort

sort recursively by Quicksort

E.g.:
Analysis of Quicksort

• Worst case:

\[ \sum_{k=2}^{n} (k-1) = \frac{n(n-1)}{2} \]

• Average case:
  – split can move a key across a large section of the list.
  – \( A(k) \) : the average number of key comparisons for sorting \( k \) numbers.

\[
A(n) = (n-1) + \sum_{i=1}^{n} \frac{1}{n} (A(i-1)+A(n-i-1))
\]

\[ n>1, A(1)=A(0)=0 \]

\[
A(n) = (n-1) + \frac{1}{n} \sum_{i=1}^{n} 2A(i) \approx 1.4(n+1)\lg n
\]
Lower bound of problems

- Discussion:
  - How to measure the difficulty of a problem?
  - The lower bound of a problem is the minimum time-complexity of all algorithms which can be used to solve this problem.
  - The lower bound is the higher the better.

- Definition: (Optimal algorithm for a problem)

\[
\text{time complexity of algorithm} = \text{Lower bound of problem}
\]

Optimal

Algorithm

Problem
Lower bound of sorting

• **Definition:** *(Comparison Sorting Model)*
  – the sorted order is determined based only on comparisons between the input element.
  – Comparison sorts can be viewed abstractly in terms of decision trees.

• **Examples:**
  – Insertion Sort, Bubble Sort, Selection Sort and Quick Sort.
Lower bound of sorting

• Definition: (Decision Tree)
  – represents the comparisons performed by a sorting algorithm when its
    operates on an input of a given size.

• Examples:
  – decision tree of input $x_1, x_2, x_3$. 
Lower bound of sorting

- **Properties:**
  - The longest path from the top to a leaf node, which is called the depth of the tree, represents the worst case time-complexity of the algorithm.
  - To find the lower bound of sorting problem, we have to find the smallest depth of a tree, among all possible binary decision tree.

- **Lower bound for the worst case:**
  - a lower bound on the heights of decision trees is a lower bound on the running time of any comparison sort algorithm.
  - **Theorem:**
    - Any decision tree that sorts n elements has height $\Omega(n \log n)$.
  - **proof:**
    - $n! \leq 2^h$, $h \geq \log(n!) \geq \log((n/2)^{n/2}) = (n/2) \log (n/2) = \Omega(n \log n)$. 
Lower bound of sorting

- Lower bound for the average case:
  - Definition: (External Path Length, EPL)
    - the sum of the lengths of all paths from the root to a leaf

  - Note: The smaller the EPL, the better the average behavior

- Lemma:
  - Any binary tree with $l$ leaves, the epl is minimized if all leaves are on at most two adjacent levels.

- Lemma:
  - The minimum epl for binary trees with $l$ leaves is $l\lceil\lg l\rceil + 2(l - 2\lceil\lg l\rceil)$.

- Lemma:
  - The average path length is at least $\lceil\lg l\rceil = \lceil\lg n!\rceil \approx n \lg n - 1.5 n$
Lower bound of sorting

• Lower bound for the average case:

\[ X \quad \text{level } k \quad \Rightarrow \quad Y \quad \text{level } d \]

\[ X \quad \text{level } k - 1 \quad \Rightarrow \quad Y \quad \text{level } d \]

\[ X \quad \text{level } k \quad \Rightarrow \quad Y \quad \text{level } d-1 \]
Summary of lower bound of sorting

- Lower bound of sorting:
  - worst-case: $\Omega(n \log n)$
  - Average-case: $\Omega(n \log n)$

<table>
<thead>
<tr>
<th>Sorting algorithms</th>
<th>Worst-case</th>
<th>Average-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Quick sort</td>
<td>$O(n^2)$</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>

- Question:
  - Is there any sorting algorithm which is optimal in worst case?
- Answer:
  - Yes, heapsort, for example.

Note:
The above analysis is for the comparison sorts on sorting problem.
Finding a lower bound

- Goal: find a lower bound of a problem, namely A.
- Initial states:
  1. a problem B whose lower bound has been found.
  2. an algorithm C for problem A.

\[
\text{Input of B} \rightarrow \text{Input of A} \rightarrow \text{solved by algorithm C} \rightarrow \text{Answer of A} \rightarrow \text{Answer of B}
\]

lower bound of A \geq lower bound of B
Finding a lower bound

• Example: convex hull
  – Goal:
    • find a lower bound of convex hull problem.
  – Initial states:
    • 1. the lower bound of sorting problem is \( \Omega(n \log n) \).
    • 2. suppose algorithm C can solve convex hull problem.
  – Steps:
    • 1. input of sorting problem: \( x_1, x_2, \ldots, x_n \).
    • 2. transforming to the input of convex hull problem: \( <x_i, y_i> \), \( y_i = x_i^2 \).
    • 3. solve the convex hull problem by algorithm C.
    • 4. the answer of convex hull problem is also the answer of sorting problem.
Finding a lower bound

- **Example: diameter**
  - **Goal**: find a lower bound of diameter problem.
  - **Initial states**:
    - 1. the lower bound of set disjoint problem is $\Omega(n \log n)$.
    - 2. suppose algorithm C can solve convex hull problem.
  - **Set disjoint problem**:
    - $A = \{a_1, a_2, \ldots, a_n\}$, $B = \{b_1, b_2, \ldots, b_n\}$
    - test whether $A$ and $B$ do not share any element.
  - **Step**:
    - mapping from set disjoint to set diameter:
      - $a_i \Rightarrow y = a_i x$
      - $b_i \Rightarrow y = b_i x$
Finding a lower bound

Examples:

- \((x_1, x_1^2)\)
- \((x_2, x_2^2)\)
- \((x_3, x_3^2)\)
- \((x_4, x_4^2)\)
Finding a lower bound

• Reasoning:
  – If the diameter equals to 2, then set A and B share some elements.
  – Otherwise, set A and B do not share any element.
  – Therefore, we can use the algorithm C for set diameter problem to solve the set disjoint problem.
  – Since the lower bound of set disjoint problem is $\Omega(n \log n)$, algorithm C can not run faster than $\Omega(n \log n)$.
  – In other words, there can not be any algorithm for set diameter problem which run fast than $\Omega(n \log n)$.
  – Thus, the lower bound of set diameter problem is $\Omega(n \log n)$. 
Other sorting algorithms

- Selection sort
- Knockout sort
- Heap sort
- Merge sort
- Shell sort
- Radix sort
- External sort