Dr. Amotz Bar-Noy’s
Compendium of Algorithms Problems

Problems, Hints, and Solutions
Chapter 1

Searching and Sorting Problems

1.1 Problems

1.1.1 Sum from Sorted Arrays (Fall 2002 Midterm)

Consider the following problem:

Input: A positive integer \( x > 0 \) and two sorted arrays \( A \) and \( B \) each contains \( n \) positive integers:

\[
0 \leq A[1] \leq A[2] \leq \cdots \leq A[n] \\
0 \leq B[1] \leq B[2] \leq \cdots \leq B[n]
\]

Objective: Find two numbers \( A[i] \) and \( B[j] \) such that \( A[i] + B[j] = x \).

Note that \( x \) and the numbers in both arrays could be very large.

Output: “Yes” if there exist indices \( 1 \leq i, j \leq n \) for which \( A[i] + B[j] = x \) and “No” otherwise.

a. Describe an efficient algorithm to solve this problem (you may use your own words”).

You get 100% for an \( O(n) \) algorithm, 75% for an \( O(n \log n) \) algorithm, 50% for an \( O(n^2) \) algorithm, 20% for no answer, and nothing for a wrong answer.

b. What is the worst-case complexity (number of operations) of your algorithm?

Use the “\( O \)” notation and Explain your answer in at most 3-4 sentences.

1.1.2 Comparator Network: 2 Smallest (Fall 2002 Midterm)

The objective is to construct an efficient network of comparators that finds the 2 smallest number among \( n \) numbers for any positive value of \( n \).

1. Construct an efficient network that finds the smallest number among \( n = 7 \) numbers.

2. What is the exact complexity of what you constructed for \( n = 7 \)?

size ____________________________________________________ depth ____________________________________________
3. What is the exact complexity of a similar network for any integer \( n \)?

<table>
<thead>
<tr>
<th>size</th>
<th>depth</th>
</tr>
</thead>
</table>

4. Construct an efficient network that finds the 2 smallest number among \( n = 8 \) numbers.

5. What is the exact complexity of what you constructed for \( n = 8 \)?

<table>
<thead>
<tr>
<th>size</th>
<th>depth</th>
</tr>
</thead>
</table>

6. What is the exact complexity of what you constructed for \( n = 2^k \) for an integer \( k \geq 1 \)?

<table>
<thead>
<tr>
<th>size</th>
<th>depth</th>
</tr>
</thead>
</table>

7. What is the exact complexity of a similar network for any integer \( n \)?

<table>
<thead>
<tr>
<th>size</th>
<th>depth</th>
</tr>
</thead>
</table>

### 1.1.3 Search with a “Clue“ (Fall 2003 Midterm)

Let \( n \geq 1 \) be an integer and let \( x \) be an unknown integer that is greater or equal to 1 and less or equal to \( n \) (\( 1 \leq x \leq n \)). The goal is to find \( x \) with as minimum as possible comparisons of the type: \( x \leq i \) for \( 1 \leq i \leq n \).

Each of the following three cases restricts the values \( x \) can get. In each case, explain with few sentences how to modify the binary search to get a better performance than \( \log_2 n \) in the worst case. Then compute the exact worst case number of comparisons asked by your modification as a function of \( n \).

To avoid ceilings, in each case there is an assumption about the value of \( n \).

1. \( x \) is an odd number. \( n = 2^k \) is a power of 2.
2. \( x = y^2 \) is a square number. \( n = (2^k)^2 \) is a square of a power of 2.
3. \( x = 2^y \) is a power of 2. \( n = 2^{(2^k)} \) is a power of 2 and the exponent is also a power of 2.

### 1.1.4 Sorting Bitonic Arrays (Fall 2003 Midterm)

Sorting bitonic arrays.

An array \( A \) of \( n \geq 1 \) distinct positive integers is bitonic if there exists an index \( 1 \leq i \leq n \) such that:


Describe a linear time (\( O(n) \)) algorithm that sorts bitonic arrays with very large integers (you cannot use countin-sort or radix-sort). Use words and do not write codes. Explain why your algorithm is linear in the number of comparisons and number of operations.
1.1.5 Trinary Search (Fall 2004 Midterm)

The Trinary-Search procedure finds a value \( x \) in the range \( \{f, \ldots, \ell\} \) for some positive integers \( 1 \leq f < \ell \leq n \). Assume that always the size of the range, \( \ell - f + 1 \), is a power of 3.

- The procedure first checks if \( f = \ell \) (not a comparison). If the answer is “yes” then the search terminates.
- Otherwise, let \( m = \frac{\ell - f + 1}{3} \) be one third of the size of the range. The procedure partitions the input range into the following 3 equal size ranges: \( \{f, \ldots, f + m - 1\} \), \( \{f + m, \ldots, \ell - m\} \), and \( \{\ell - m + 1, \ldots, \ell\} \). For example the range \( \{10, \ldots, 18\} \) is partitioned into the ranges: \( \{10, 11, 12\} \), \( \{13, 14, 15\} \), and \( \{16, 17, 18\} \).
- The first comparison is: \( x \leq f + m - 1 \) (does \( x \) belong to the first range?).
- If the answer is “yes” then the search continues recursively in the first range: \( \{f, \ldots, f + m - 1\} \).
- If the answer is “no” a second comparison is performed: \( x \leq \ell - m \) (does \( x \) belong to the second range?).
- If the answer is “yes” then the search continues recursively in the second range: \( \{f + m, \ldots, \ell - m\} \).
- If the answer is “no” then the search continues recursively in the third range: \( \{\ell - m + 1, \ldots, \ell\} \).

1. How many comparisons between \( x \) and a number in the range (and not between 2 numbers in the range) are performed by Trinary-Search on the range \([1..n]\) in the following cases:

- \( n = 3 \) and \( x = 1, 2, 3 \).

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>#comparisons</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( n = 9 \) and \( x = 1, \ldots, 9 \).

<table>
<thead>
<tr>
<th>x</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>
</tr>
</thead>
<tbody>
<tr>
<td>#comparisons</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( x = n \) for any \( n \).
- \( x = 1 \) for any \( n \).

2. What is the worst case complexity (number of comparisons) of Trinary-Search?

Write the initial values and the recursive formula.

Solve it using the master theorem.

Solve it accurately.

3. What is the average number of comparisons performed by Trinary-Search assuming \( x \) is selected randomly with the uniform distribution from the range \([1..n]\)?

Write the initial values and the recursive formula.

Solve it using the master theorem.

Solve it accurately.
1.1.6 Pairs of nuts and bolts (Fall 2005 Midterm)

Consider the following “comparison” model. There are \( n \) nuts and \( n \) bolts for \( n \geq 1 \). All nuts are of different size and each nut matches exactly one bolt. The sizes of the nuts and the bolts are so close that it is impossible to distinguish between any two nuts or any two bolts. However when matching a nut with a bolt it is clear if they match, if the nut is bigger, or if the nut is smaller. But not how much bigger or how much smaller.

The objective is to find the largest pair or to order all the \( n \) pairs using as minimum as possible matchings.

You may assume that the nuts and the bolts are labelled and that you “remember” all previous results and may conclude any consistent partial order that “agrees” with these results.

1. Describe an \( O(n^2) \) simple algorithm that sorts all the \( n \) pairs.
   State and prove the exact worst-case complexity of your algorithm?

2. Describe an efficient algorithm that finds the largest pair.
   State and prove the worst-case number of matchings performed by your algorithm.
   A full credit will be given to an algorithm that always uses at most \( 2n - 2 \) matchings.

3. Show that no algorithm can guarantee finding the largest pair with less than \( 2n - 2 \) matchings.
   Hint: Define an adversary strategy. Recall that the adversary establishes the order depending on the algorithm matches. For example, the adversary may select the new nut and/or bolt whenever a matching is between a nut and/or bolt that did not participate in a matching before.

4. Describe a “better” than \( \Theta(n^2) \) algorithm that sorts all the \( n \) pairs.
   State and prove the complexity of your algorithm.
   Hint: It seems like it is easier to adapt Quick-Sort.

1.1.7 Partitioning from multiple sorted arrays (Fall 2006 Midterm)

Let \( A \) and \( B \) be two increasingly sorted arrays each of size \( n \). Assume that all the \( 2n \) entries are integers and all of them are distinct. The goal is to partition the \( 2n \) numbers into two sets: one containing the largest \( n \) integers and one containing the smallest \( n \) integers.

1. Describe an algorithm that finds these sets.
   Describe an efficient algorithm that finds these sets with as minimum as possible comparisons between integers from both arrays.
   You may answer only this part if you are sure that your algorithm is correct.

2. What is the worst case number of comparisons made by your first algorithm as a function of \( n \)?
   Justify your answer. You may assume that \( n \) is a power of 2 and may ignore floors and ceilings.
   What is the worst case number of comparisons made by your second algorithm as a function of \( n \)?
   Justify your answer. You may assume that \( n \) is a power of 2 and may ignore floors and ceilings.

1.1.8 Matrix with some ordering (Fall 2006 Midterm)

Let \( M \) be an \( n \times n \) matrix containing \( n^2 \) distinct integers in which the integers of each row are sorted in an increasing order (from left to right) and the integers of each column are sorted in an increasing order (from top to bottom).
1. Let $x$ be one of the numbers in $M$. Describe an algorithm to find the position of $x$ in $M$.

   Describe an efficient $O(n)$ algorithm to find the position of $x$ in $M$.

   You may answer only this part if you are sure that your algorithm is correct.

2. What is the worst case number of comparisons made by your first algorithm as a function of $n$?
   Justify your answer. You may assume that $n$ is a power of 2 and may ignore floors and ceilings.

   What is the worst case number of comparisons made by your second algorithm as a function of $n$?
   Justify your answer. You may assume that $n$ is a power of 2 and may ignore floors and ceilings.

3. A bonus question: Show an $\Omega(n)$ lower bound for the worst case number of comparisons made by any algorithm. Justify your answer.

1.1.9 Array with One Missing (Fall 2007 Midterm)

Let $A = A[1], \ldots, A[n]$ be an array of $n$ distinct integers, in which each integer is in the range $[1..n+1]$. That is, exactly one integer out of $\{1, \ldots, n+1\}$ is missing from $A$. The goal is to find the missing number as efficiently (fast) as possible.


For each part, describe an efficient algorithm to find the missing integer. Justify the correctness of your algorithm. Next, analyze the worst case time complexity of your algorithm, using $O$-notation. Explain your analysis.

1. The integers in $A$ appear in an arbitrary order.

2. $A$ is sorted.

1.1.10 Overlapping Sorted Arrays (Fall 2008 Midterm)

Let


and


be two sorted arrays of size $n$. The numbers in each array are distinct. However, there could be some numbers that appear in both arrays.

The goal in part (a) is to find one of the numbers that appear in both arrays if such numbers exist or to announce that the arrays are disjoint. The goal in part (b) is to find one of the numbers that appear only in one of the arrays if such numbers exist or to announce that the arrays are identical. In both parts, an efficient solution should make as few as possible comparisons between numbers in the worst-case.

For each part, justify the correctness of your algorithm and compute the number of comparisons in the worst-case. A full credit will be given to an answer that states the exact number of comparisons. Partial credit will be given for answers using the big $O$ notation.

Partial credit will be given to correct but not efficient solutions. If you are not sure about your efficient solution, you may provide the less efficient solution.

1. Describe an efficient algorithm to find a number that appears in both arrays if exists. If such a number does not exist, the algorithm should output that both arrays are disjoint.

2. Describe an efficient algorithm to find a number that appears in only one of the arrays if exists. If such a number does not exist, the algorithm should output that both arrays are identical.
1.1.11 Using Ternary Comparison (Fall 2009 Midterm)

Let $S$ be a set of $n > 0$ distinct integers. Assume that $n$ is a power of 2. Here are some known results for the number of comparisons needed to execute some tasks on $S$:

- Finding the largest number in $S$ is possible with $n - 1$ comparisons.
- Finding the largest number in $S$ is impossible with $n - 2$ comparisons.
- Finding both the largest and smallest numbers in $S$ is possible with $3n/2 - 2$ comparisons.
- Finding both the largest and second largest numbers in $S$ is possible with $n + \log_2(n) - 2$ comparisons.

Assume now that $n > 0$ is a power of 3 and that each comparison can compare three elements from the set $S$ and order them from the largest to the smallest. Call such a comparison: a ternary comparison.

For each of the following tasks, you are given the option to provide two solutions. The first can be inefficient but must be correct. The second should be more efficient than the first. If you are sure about your efficient solution, then you don’t have to provide the inefficient solution. The purpose of the inefficient solution is to guarantee some partial credit.

1. Describe an efficient algorithm that uses as few as possible ternary comparisons to find the largest number in the set $S$. Explain why your algorithm is correct and state the exact number of ternary comparisons it uses in the worst case.

2. Describe an efficient algorithm that uses as few as possible ternary comparisons to find both the largest and the smallest numbers in the set $S$. Explain why your algorithm is correct and state the exact number of ternary comparisons it uses in the worst case.

3. Describe an efficient algorithm that uses as few as possible ternary comparisons to find both the largest and the second largest numbers in the set $S$. Explain why your algorithm is correct and state the exact number of ternary comparisons it uses in the worst case.

4. Prove a lower bound on the number of ternary comparisons that are needed to find the largest number in the set $S$.

1.1.12 Two Largest / Two Smallest (Fall 2010 Midterm)

Let $A$ be an arbitrary (un-sorted) array of $n$ ($4 \leq n$) distinct (very large) positive integers. Describe an efficient algorithm that finds the 2 largest numbers and the 2 smallest numbers in $A$.

What is the exact worst-case number of comparisons made by your algorithm? You may assume that $n$ is a nice number. Justify your answer.

1.1.13 Counting Ordered Pairs from Two Sorted Arrays (Fall 2010 Midterm)


Describe an efficient algorithm to determine if $A > B$, or $A < B$, or $A = B$. What is the worst-case number of comparisons made by your algorithm? Justify your answer.
1.1.14 One Sorted and One Unsorted Array (Fall 2003 Final)

$A$ is a sorted array with $n > 0$ positive integers. $B$ is an un-sorted array with $k > 0$ positive integers. The goal is to create a sorted array $C$ that contains all the $n + k$ integers.

The optimization objective is to minimize the number of comparisons. All other operations are “free”. However, the integers may be very large so you must use comparisons and you cannot use Radix Sort or any similar type of sorting algorithm.

The following 2 algorithms accomplish the task:

- Algorithm $P$
  - Sort($B$)
  - Copy $A$ to $C$
  - Merge($A, B$) into $C$. For each number in $B$: insert it into $C$.

- Algorithm $Q$
  - Copy $A$ to $C$.
  - For each number in $B$: insert it into $C$.

1. Describe an efficient implementation for Algorithm $P$.
   What is the complexity of your implementation? Explain.

2. Describe an efficient implementation for Algorithm $Q$.
   What is the complexity of your implementation? Explain.

3. Fix $n > 0$. In answering the following three questions use the ‘$O, \Omega, \Theta, o, w$” notation and ignore constants.

   For which values of $k$ Algorithm $P$ is better? For which values of $k$ Algorithm $Q$ is better? For which values of $k$, the performance of both algorithm is equal? Explain.

1.1.15 Unknown Array Procedure (Fall 2004 Final)

Let $A$ be an array of size $n$ containing $n$ distinct positive integers. Define the following procedure $g$ on $A$:

$$g(f, \ell, x, y):$$

(1) if $\ell - f = 0$ then
   $$x = A[f] \text{ and } y = A[\ell]$$

(2) if $\ell - f = 1$ then
   then $x = A[f] \text{ and } y = A[\ell]$
   else $x = A[\ell] \text{ and } y = A[f]$

(3) if $\ell - f > 1$ then
   $$m = \lfloor(\ell + f)/2\rfloor$$
   $$g(f, m, x_1, y_1)$$
   $$g(m + 1, \ell, x_2, y_2)$$
   if $x_1 < x_2$ (* comparison *)
   then $x = x_1$
   else $x = x_2$
   if $y_1 < y_2$ (* comparison *)
   then $y = y_2$
   else $y = y_1$

The input of $g$ is a sub-array of $A$ starting with $f$ and ending with $\ell$ for $1 \leq f \leq \ell \leq n$. Procedure $g$ returns $x$ and $y$ which are two values from the array.

(a) I. Run $g(1, 8, x, y)$ on the vector $A = [21, 13, 3, 89, 34, 5, 55, 8]$. What are the values of $x$ and $y$?
   $$x = _____ \quad y = _____$$

   II. Run $g(1, 11, x, y)$ on the vector $A = [144, 21, 13, 3, 233, 89, 34, 5, 55, 377, 8]$. What are the values of $x$ and $y$?

   $$x = _____ \quad y = _____$$
III. Who are \( x \) and \( y \) when running \( g(1,n,x,y) \) on \( A \)? Justify your answer.

(b) \( I. \) Let \( T(n) \) be the number of comparisons between the array values (and not indices) performed by \( g(1,n,x,y) \) on arrays of size \( n \). Write the recursive formula for \( T(n) \) assuming \( n = 2^k \) is a power of 2 and that \( n \geq 2 \). Do not forget the initial value. Explain your answer.

Solve this formula accurately.

II. Let \( R(n) \) be the depth of the recursion of \( g(1,n,x,y) \) on arrays of size \( n \). Write the recursive formula for \( R(n) \) assuming \( n = 2^k \) is a power of 2 and that \( n \geq 2 \). Do not forget the initial value. Explain your answer.

Solve accurately this formula.

(c) \( I. \) Implement procedure \( g \) on a comparator network for \( n = 8 \).

II. What are the size and depth of the above network?

\[
\text{Size}(8): \quad \text{Depth}(8): \quad \text{__________}
\]

III. What are the size and depth of a network that implements \( g \) for \( n = 2^k \)?

\[
\text{Size}(n = 2^k): \quad \text{Depth}(n = 2^k): \quad \text{__________}
\]

IV. Implement procedure \( g \) on a comparator network for \( n = 11 \).

V. What are the size and depth of the above network?

\[
\text{Size}(11): \quad \text{Depth}(11): \quad \text{__________}
\]

4. A comparison network has \( n + k \) input lines for \( n \geq 1 \) and \( k \geq 1 \). The top \( n \) input lines are sorted and the bottom \( k \) input lines are unsorted. The goal is to sort all the \( n + k \) input lines.

Describe an efficient implementation for Algorithm \( P \).

What is the size and depth of your implementation? Use the “\( O \)” notation and explain your answer.

1.1.16 Sorting \( k \)-Distance Arrays (Fall 2005 Final)

Let \( A \) be an array containing \( n \) distinct numbers. The numbers are very large and therefore the only way to sort \( A \) is by using comparisons (no radix-like sorting is possible).

\( A \) is called a \( k \)-distance array, for a parameter \( 0 \leq k \leq n - 1 \), if each of its elements is at most \( k \) locations away from its position in the sorted array. More formally, if \( A[i] \) is \( A[j] \) in the sorted array then \( |i - j| \leq k \).

1. Let \( A \) be a 0-distance array \( (k = 0) \). What could you say about its structure? Explain.

2. Describe an algorithm to sort \( 1 \)-distance arrays that uses at most \( O(n) \) comparisons. Prove that the algorithm is correct and explain why the complexity is \( O(n) \). Full credit will be given to an algorithm that uses at most \( n - 1 \) comparisons.

3. The following is a version of Insertion-Sort:

\[
\text{for } i = 1 \text{ to } n - 1 \text{ do} \\
\quad j = i + 1 \\
\quad \text{while } (A[j] < A[j - 1]) \text{ and } (j > 1) \text{ do} \\
\quad \quad \text{exchange } A[j] \text{ with } A[j - 1] \\
\quad \quad \text{end-while} \\
\quad j = j - 1 \\
\text{end-for}
\]
What is the worst-case number of comparisons between array’s elements, performed by the above algorithm, when applied to a $k$-distance array $A$? State this number as a function of $k$ and $n$. Justify your answer.

**Remark:** Even though the algorithm does not know the value of $k$, the complexity does depend on $k$. Do not look for the worst value of $k$, instead find an upper bound on the number of comparisons given that the array is a $k$-distance array.

4. For which values of $k$, does your algorithm use at most $o(n \log n)$ comparisons?

**Remark:** For these values of $k$, your algorithm is “better” than Merge-Sort.

5. Describe an algorithm that sorts $k$-distance arrays. Your algorithm should perform less than $O(n \log n)$ comparisons in the worst case for any value of $k$.

**Remark:** It is possible to design an algorithm that uses $O(n \log k)$ comparisons. Since $k \leq n$, it follows that such an algorithm is better than an algorithm that uses $O(n \log n)$ comparisons for any value of $k$.

**Hint:** What happens after sorting the first $2k$ entries of the array?

6. Prove that any algorithm that sorts $k$-distance arrays, must use $\Omega(n \log k)$ comparisons.

**Hint:** For given $n$ and $k$ (you may assume that $k$ divides $n$), define a family of un-sorted $k$-distance arrays. Next, use the lower bound of $\Omega(h \log h)$ comparisons needed to sort arrays of size $h$ to show that for arrays in these family, any sorting algorithm requires $\Omega(n \log k)$ comparisons.

### 1.1.17 Majority in an Array (Fall 2006 Final)

Let $A$ be an array containing $n$ very large positive integers not necessarily distinct. The array is not sorted and you cannot use “radix-sort” like algorithms to sort it. A majority is a number that appears at least $\lceil (n + 1)/2 \rceil$ times in the array (note that there can be at most one majority).

Describe an $O(n)$-time algorithm that finds a majority in $A$ if exists. Explain why your algorithm is correct and why it has time complexity $O(n)$. If you cannot find an $O(n)$-time algorithm, describe a correct algorithm, explain why it is correct, state its time complexity, and justify your complexity claim.

### 1.1.18 Does Array Contain Integer in Range (Fall 2006 Final)


Describe an efficient algorithm to find if the array $A$ contains an integer that is greater than $x$ and smaller than $y$.

How many comparisons are made by your algorithm in the worst case?

Explain why your algorithm is correct and justify your complexity claim.

### 1.1.19 Element Same as Index (Fall 2009 Final)


### 1.1.20 Inversions in an Array (Fall 2009 Final)

Let $A = A[1], \ldots, A[n]$ be an array of $n$ distinct positive integers (the value of these integers could be very very large). An inversion is a pair of indices $i$ and $j$ such that $i < j$ but $A[i] > A[j]$. 
For example in the array $[30000, 80000, 20000, 40000, 10000]$, the pair $i = 1$ and $j = 3$ is an inversion because $A[1] = 30000$ is greater than $A[3] = 20000$. On the other hand, the pair $i = 1$ and $j = 2$ is not an inversion because $A[1] = 30000$ is smaller than $A[2] = 80000$. In this array there are 7 inversions and 3 non-inversions.

1. What is the structure of an array with the minimum possible number of inversions? What is this number as a function of $n$?

2. What is the structure of an array with the maximum possible number of inversions? What is this number as a function of $n$?

3. Describe an efficient algorithm that counts the number of inversions in any array. What is the running time of your algorithm?

1.1.21 Less-Than Question Game (Fall 2010 Final)


In each round, the algorithm may ask $k$ questions (comparisons) of the type: “is $x \leq A[j_1]$?”, “is $x \leq A[j_2]$?”, ..., “is $x \leq A[j_k]$?” for some indices $j_1, j_2, \ldots, j_k$.

Design an efficient algorithm that finds the index $i$ for which $A[i] = x$ in minimum number of rounds. What is the worst case number of rounds of your algorithm? Justify your answer.

In order to avoid floors and ceilings you may assume that $n$ is a “nice” number.

1.1.22 Cycles in an Array (Fall 2010 Final)

Let $A = [A[1], A[2], \ldots, A[n]]$ be an array containing all the numbers $1, \ldots, n$ ($A$ represents a permutation). A sequence of indices $(i_1, i_2, \ldots, i_k)$ is a cycle of size $k$ if


Example: If $A[i] = i$ then the singelton sequence $(i)$ is a cycle of size 1.

Example: In the array $[4, 6, 3, 5, 8, 7, 2, 1]$ the sequence $(1, 4, 5, 8)$ is a cycle of size 4. The other 2 cycles are $(2, 6, 7)$ and $(3)$.

Design an efficient algorithm that finds the number of cycles in $A$. What is the running time of your algorithm? Justify your answer.

1.1.23 Find Non-Max Non-Min Numbers (2003 GC Exam)

Let $A$ be an array of $n \geq 5$ distinct positive integers that may be very large compared to $n$. For each one of the following three tasks describe an efficient algorithm that tries to minimize the number of comparisons required to accomplish the task in the worst case. Use words and illustrations and do not write code. State the number of comparisons your algorithm uses in the worst case. Explain your answer.

1. Find a number in the array that is neither the maximum nor the minimum.

2. Find a number in the array that is neither the maximum and the second maximum nor the minimum and the second minimum.

3. For $1 \leq k < n/2$, find a number in the array that is neither among the top $k$ numbers nor among the bottom $k$ numbers.
Distinct numbers: The question is well defined since the numbers are distinct and \( n \geq 5 \).

Hint: You do not need to “touch” all the \( n \) numbers in order to solve the three tasks.

Complexity: For task (a), you are expected to find the optimal solution. For task (b), you are expected to find at least a near-optimal solution. For task (c), you are expected to find an algorithm that is optimal up to a constant factor as a function of \( k \).

1.1.24 Small Array Tasks (2004 GC Exam)

Let \( A \) be an array of size \( n \), whose elements are very large positive integers (therefore, you cannot use radix-sort like algorithms to sort the array).

For each one of the following 4 tasks describe an efficient algorithm to solve it. Find the running time of your algorithm and explain why this is the running time. Non-efficient solutions may receive partial credit.

Use your own words and do not write computer programs. You may apply known algorithms (such as, Merge-Sort) as sub-procedures without describing them and you may assume that their running time is known (\( O(n \log n) \) for Merge-Sort) without explaining it.

1. Find the maximum difference between any of the integers in the array.
2. Find the minimum positive difference between any of the integers in the array.
3. Decide whether or not all of the integers in the array are identical.
4. Decide whether or not all of the integers in the array are distinct.

1.1.25 Find Distinct Element (2006 GC Exam)

Let \( A \) be an array containing \( n \geq 1 \) very large positive integers (if the array is not sorted, you cannot use radix sort, bucket sort, or counting sort to sort it). Your goal is to find one of the numbers that appears only once in the array if such a number exists.

For each of the following five cases describe (words are better than codes!) an efficient algorithm to solve the above task and state the worst-case complexity of your algorithm.

Explain why your algorithms are correct and justify your complexity claims.

2. \( A \) is an arbitrary array: the integers in the array may appear in any order.
3. There are exactly 2 distinct integers in the entire array \( A \) (but \( A \) is not sorted).
4. If a number \( x \) appears in \( A \) more than once, then all of its appearances are consecutive (but \( A \) is not sorted).
5. \( A \) is a bitonic array: there exists an index \( i \) (not known to the algorithm), \( 1 \leq i \leq n \), such that \( A[1] \leq A[2] \leq \cdots \leq A[i-1] \leq A[i] \geq A[i+1] \geq \cdots \geq A[n-1] \geq A[n] \).

1.1.26 Pancake Flipping (2007 GC Exam)

Story: There are \( n \) pancakes all of different sizes that are stacked on top of each other. It is allowed to slip a flipper under one of the pancakes and flip over the whole sack above the flipper. The purpose is to arrange pancakes according to their size with the biggest at the bottom.
Model: Let $A$ be an array of size $n$ containing the numbers $1, \ldots, n$ in any order ($A$ represents an arbitrary permutation). For any $2 \leq i \leq n$, the $F_i$ operation (flip) is to reverse the prefix of size $i$ of the array.

**Example I:** $F_5([1, 2, 3, 4, 5, 6, 7, 8]) = [5, 4, 3, 2, 1, 6, 7, 8]$.

**Example II:** $F_3([3, 2, 1, 4, 5, 6, 7, 8]) = [1, 2, 3, 4, 5, 6, 7, 8]$.

**Problem:** Sort the array with as few as possible flips.

**Remark:** In the comparison model, array operations were free and the objective is to minimize the number of comparisons. In the pancake model, comparisons are free since the final location of each pancake is known. The objective is to minimize the number of array (flip) operations.

1. Sort the array $[8, 7, 6, 5, 1, 2, 3, 4]$ with as few as possible flips. Specify the list of flips that would sort the array.
2. Sort the array $[8, 6, 4, 2, 1, 3, 5, 7]$ with as few as possible flips. Specify the list of flips that would sort the array.
3. Describe an (efficient) algorithm to sort any array of size $n$ with flips. Use words!
4. How many flips, as a function of $n$, your algorithm uses in the worst-case? Be as accurate as you can.

### 1.1.27 Identifying Number with Different Query Types (2008 GC Exam)

Let $n$ be a positive integer that is a power of 2. The goal is to “identify” or “approximate” an unknown number in the range $[1..n]$ with as few as possible queries in the worst case.

When the query is “is the number less or equal to $x$?” then the binary search procedure would identify any number with exactly $\log_2(n)$ queries.

For each one of the following goals and associated query types, describe a search algorithm and state its worst case complexity. Explain why your algorithm is correct and justify your complexity claim.

1. **Goal:** Identify the number.
   **Query type:** “For two numbers $x < y$, is the unknown number smaller than $x$, or greater than $y$, or between $x$ and $y$?”

2. **Goal:** Approximate the number to be in a range $[x..2x]$ for some $1 \leq x \leq n$.
   **Query type:** “For a number $x$, is the unknown number between $x$ and $2x$?”

### 1.1.28 Small Array Algorithms (2009 GC Exam)

Let $A$ be an unsorted array of $n$ distinct very large integers (it is therefore impossible to sort $A$ using linear-time sorting algorithms).

Let $B$ be an array that contains all the $n$ integers from $A$ in an increasing order.

For each one of the following tasks, state the worst-case complexity – number of comparisons – of executing it with an efficient deterministic algorithm on $A$ and on $B$.

Justify your answer with few sentences. You may use known algorithms whose complexity is known.

1. **Find($x$):** for an arbitrary integer $x$, returns true if $x$ is one of the $n$ integers; otherwise returns false.
2. **Next($x$):** for an arbitrary integer $x$, returns $y$ if $y$ is the smallest integer among the $n$ integers such that $x < y$; otherwise returns $\infty$.
3. **Max:** returns the maximum integer among the $n$ integers.
4. **Average**: returns the average of all \( n \) integers.

5. **Median**: returns the median of the \( n \) integers. You may assume that \( n \) is an odd number.

### 1.1.29 Distinct Integers in Two Sorted Arrays (2010 GC Exam)

Let \( A \) be a sorted array containing \( n \) distinct positive integers and let \( B \) be a sorted array containing \( k \) distinct positive integers.

1. Assume \( n = k \). Describe an efficient algorithm that counts the number of distinct integers that appear in both arrays. What is the running time (number of comparisons) of your algorithm as a function of \( n \)?

2. Assume \( k = \log_2(n) \). Describe an efficient algorithm that counts the number of distinct integers that appear in both arrays. What is the running time (number of comparisons) of your algorithm as a function of \( n \)?

### 1.1.30 Find Missing Element Minimizing Array Accesses (2011 GC Exam)

Let \( A = A[1], \ldots, A[n] \) be an array of \( n \) distinct integers, in which each integer is in the range \([1..n+1]\). That is, exactly one integer out of \( \{1, \ldots, n+1\} \) is missing from \( A \). The goal is to find the missing number with as few accesses as possible to the array \( A \). Note, that any operation with any integer from the array is considered as accessing the array.

1. The integers in \( A \) appear in an *arbitrary* order. Describe an efficient algorithm to find the missing integer. Justify the correctness of your algorithm. Analyze the worst case complexity (number of accesses to array \( A \)) of your algorithm. Justify your analysis.

1.2 Hints

Individual subsections with hints for each problem here.
1.3 Solutions

Individual subsection with solutions for each problem here.
Chapter 2

Graph Problems

2.1 Problems

2.1.1 Longest Path (Final Exam 2003)

Let $G$ be a simple undirected graph with $n$ vertices and $m$ edges.

A simple path in $G$ is a path in which all the vertices are different. The length of a simple path is the number of vertices in the path.

A Hamilton path is a simple path whose length is $n$. $G$ is Hamiltonian if it has a Hamilton path. The Hamiltonian-Path problem is to determine whether a graph is Hamiltonian. The Hamiltonian-Path problem is an NP-Complete problem.

A simple path is maximal if it cannot be extended by another vertex. That is, the path $v_1 - v_2 - \cdots - v_k$ is maximal if both $v_1$ and $v_k$ have no neighbors outside the path.

1. The Longest-Path problem is to find the longest simple path in a graph. The goal is to prove that this problem is an NP-Complete problem.

First, define the decision problem associated with the optimization version of the Longest-Path problem.

Next prove that the Longest-Path problem is an NP-Hard problem by describing a polynomial time reduction from the Hamiltonian-Path problem.

Finally, prove that the Longest-Path problem is an NP problem and thus it is an NP-Complete problem.

2. The following algorithm generates a maximal path:

Algorithm Maximal-Path
Start with an arbitrary vertex $v$
In a “DFS manner” extend $v$ to one side until it is impossible.
In a “DFS manner” extend $v$ to the other side until it is impossible.

What is the complexity of Algorithm Maximal-Path when the graph is maintained by adjacency lists? Explain.

What is the complexity of Algorithm Maximal-Path when the graph is maintained by an adjacency matrix? Explain.
3. The guaranteed performance of Algorithm Maximal-Path is the guaranteed length of the output path no matter which was the starting vertex and how the DFS search was implemented. For example, the guaranteed performance of Algorithm Maximal-Path on the Null Graphs is 1 since there are no edges and the guaranteed performance of Algorithm Maximal-Path on the Complete Graphs is $n$ since all the edges exist.

What is the guaranteed performance of Algorithm Maximal-Path on a Complete Bipartite Graphs in which one size has $x$ vertices, the other side has $y$ vertices, $x + y = n$, and $x < y$? Explain.

What is the guaranteed performance of Algorithm Maximal-Path on a Cycle Graph with $n$ vertices? Explain.

What is the guaranteed performance of Algorithm Maximal-Path on a Star Graph with $n$ vertices? Explain.

What is the guaranteed performance of Algorithm Maximal-Path on a graphs with $n$ vertices in which the degree of every vertex is at least $\delta$ for $0 \leq \delta \leq n - 1$? Explain.

2.1.2 Removing One-Edge Vertices (Final Exam 2006)

Let $G$ be a simple undirected graph with $n$ vertices and $m$ edges. Apply the following process on $G$:

- As long as $G$ contains at least one vertex with degree 1 (exactly one neighbor), remove one of these vertices and the edge that contains it from $G$.

For which graphs does this process terminate with a graph that has exactly one vertex? Explain.

2.1.3 Set-Generating Algorithm from Graph (Final Exam 2006)

The following algorithm is applied to a simple undirected graph.

let $S$ be the empty set;
let $R$ be the set of all vertices;
while $R$ is not empty do
  let $v$ be an arbitrary vertex from $R$;
  add $v$ to $S$;
  remove $v$ from $R$;
  remove from $R$ all the neighbors of $v$ that are in $R$;
end-while

What can you say about the set $S$ when the algorithm terminates? Explain.

2.1.4 Quick Graph Analysis Algorithm (Final Exam 2006)

The following algorithm gets as an input a simple and undirected graph with at least 3 vertices. Let the vertices be 1, 2, …, $n$.

for $u = 1$ to $u = n$ do
  for $v = 1$ to $v = n$ do
    for $w = 1$ to $w = n$ do
      if $(u, v)$ and $(u, w)$ and $(v, w)$ are edges
        then return “$(u, v, w)$” and abort
print “???”

1. If the algorithm returns “$(u, v, w)$”, what could you say about these three vertices? What should be printed instead of the “???” at the end of the algorithm? Justify your answers.

2. What is the worst-case time complexity of the algorithm? Explain.
2.1.5 Greedy Graph Coloring (Final Exam 2006)

Let \( G \) be a simple and undirected graph with \( n \) vertices. A coloring of \( G \) is an assignment of the numbers 1, \ldots, \( x \) to the vertices in a way that the two end-points of any edge are colored with two distinct colors.

The greedy coloring algorithm considers the vertices in any order and assigns the smallest possible color to a vertex after examining the colors of its neighbors (those that were colored at this time).

Show that this greedy algorithm is not optimal. You need to find a graph for which greedy uses more colors than required. Illustrate the graph, color it optimally, and color it with the greedy algorithm.

Full credit is given for an example in which the ratio between the number of colors used by the algorithm to the number of colors used by an optimal coloring is \( \Omega(n) \).

2.1.6 Graph with Extra Edges (Final Exam 2007)

Let \( G \) be a simple undirected graph with the \( n \) vertices \( \{1, 2, \ldots, n\} \) and \( m \) edges.

Define \( H \) as follows. The vertices of \( H \) are the vertices of \( G \). The edge \( (u, v) \) is in \( H \) if either \( (u, v) \) is an edge in \( G \) or there exists another vertex \( w \) such that both \( (u, w) \) and \( (w, v) \) are edges in \( G \).

In other words, there exists an edge \( (u, v) \) in \( H \) if and only if there exists a path of length at most 2 from \( u \) to \( v \) in \( G \).

1. Which is \( H \) if \( G \) is the complete graph? Explain.
2. Which is \( H \) if \( G \) is the null graph? Explain.
3. Which is \( H \) if \( G \) is the complete bipartite graph? Explain.
4. Is \( H \) a tree if \( G \) is a tree? Explain.
5. Assume \( G \) is represented by an adjacency matrix. Describe an algorithm to generate the adjacency matrix of \( H \). Justify the correctness of your algorithm.
   What is the complexity of your algorithm? Explain.
   Be sure to be correct and only then think of efficiency.
6. Assume \( G \) is represented by sorted adjacency lists. Describe an algorithm to generate the sorted adjacency lists of \( H \). Justify the correctness of your algorithm.
   What is the complexity of your algorithm? Explain.
   Be sure to be correct and only then think of efficiency.

2.1.7 Does Graph Have Triangle (Final Exam 2008)

Let \( G \) be a simple undirected graph with \( n \) vertices and \( m \) edges. A triangle in a graph is a set of three vertices \( u, v, w \) such that the three edges \( (u, v), (v, w), (w, u) \) are in the graph.

1. Describe a correct algorithm that decides whether or not a graph has a triangle. What is the complexity of your algorithm? Explain your answer.
2. Describe a correct and efficient algorithm that decides whether or not a graph has a triangle. What is the complexity of your algorithm? Explain your answer.
2.1.8 Pseudocode on Graph (Final Exam 2009)

The following algorithm is applied to a simple undirected graph with \( n \) vertices in the set \( V \).

let \( S \) be a set containing an arbitrary vertex \( v \in V: S = \{v\} \)
let \( R \) be the set off all the neighbors of \( v: R = \{\text{neighbors}(v)\} \)

**while** \( R \) is not empty **do**
- let \( v \in R \) be an arbitrary vertex from \( R \)
- add \( v \) to \( S: S = S \cup \{v\} \)
- omit \( v \) from \( R: R = R \setminus \{v\} \)
- omit from \( R \) all vertices that are not neighbors of \( v: R = R \setminus \{\text{nonneighbors}(v)\} \)

What could you say about set \( S \) when the algorithm terminates? Justify your answer.

2.1.9 Maximum/Maximal Independent Set in a Graph (Final Exam 2009)

An independent set in a graph is a set of vertices such that there is no edge between any pair of vertices in the set.

A maximum independent set in a graph is an independent set that has the largest number of vertices among all possible independent sets.

A maximal independent set in a graph is an independent set that is not contained in another independent set. That is, any new vertex that would be added to the set would create a non-independent set.

1. Find the maximum independent set in a complete bipartite graph that has \( x \) vertices in one side and \( y \) vertices in the other side for \( x + y = n \). What is the size of this set?
2. Find the smallest possible maximal independent set in a complete bipartite graph that has \( x \) vertices in one side and \( y \) vertices in the other side for \( x + y = n \). What is the size of this set?
3. Finding a maximum independent set is an NP-Complete problem whereas finding a maximal independent set is not. Describe an efficient (greedy) algorithm that finds one of the maximal independent sets in a graph. What is the worst case running time of your algorithm if the graph is maintained with adjacency lists? What is the worst case running time of your algorithm if the graph is maintained with adjacency matrix? Explain your answers.

2.1.10 Is Graph Bipartite? (Final Exam 2010)

Let \( G \) be a simple graph with \( n \) vertices and \( m \) edges. Describe an efficient algorithm that determines if \( G \) is bipartite or not. What is the running time of your algorithm? Justify your answer.

2.1.11 Path Between Vertices (Final Exam 2010)

Let \( G \) be a simple graph with \( n \geq 2 \) vertices and \( m \) edges. Let \( v \) and \( u \) be two distinct vertices in \( G \). Describe an efficient algorithm to decide if there exists a path from \( u \) to \( v \). What is the running time of your algorithm? Justify your answers.

2.1.12 Dominating Set (GC Exam 2004)

A dominating set in a graph is a set of vertices such that all other vertices have at least one neighbor in this set. A minimum dominating set in a graph is a dominating set that has the smallest number of vertices among all possible dominating sets.
(a) For each one of the following 6 families of graphs, find the exact size of a minimum dominating set. Justify your answer. For each case, the graph $G$ is simple and undirected with $n$ vertices.

(a) $G$ is the null graph (no edges).
(b) $G$ is the complete graph (all possible $\binom{n}{2}$ edges exist).
(c) $G$ is the star graph (a tree with a root connected to all other $n - 1$ vertices).
(d) $G$ is the path graph (a chain of $n$ vertices and $n - 1$ edges).
(e) $G$ is the cycle graph (a cycle of $n$ vertices and $n$ edges).
(f) $G$ is a complete bipartite graph that has $x$ vertices on one side and $y$ vertices on the other side for $x + y = n$ (all possible $x \cdot y$ edges between the two sides exist).

A minimal dominating set in a graph is a dominating set that does not contain another dominating set (deleting any vertex from the set would create a non-dominating set).

(b) A minimal dominating set is not always a minimum dominating set. Demonstrate this with a simple example.

Finding a minimum dominating set is an NP-Complete problem whereas finding a minimal independent set is not.

(c) Describe a simple algorithm that finds one of the minimal dominating sets in a graph. What is the worst case running time of your algorithm if the graph is maintained with adjacency lists? What is the worst case running time of your algorithm if the graph is maintained with an adjacency matrix? Explain your answers.

2.1.13 Minimum/Minimal Vertex Cover (GC Exam 2005)

A vertex cover in a graph is a set of vertices such that each edge has at least one of its two endpoints in this set.

A minimum vertex cover in a graph is a vertex cover that has the smallest number of vertices among all possible vertex covers.

(a) For each one of the following six families of graphs, find the exact size of a minimum vertex cover. Justify your answer. For each case, the graph $G$ is simple and undirected with $n$ vertices.

(a) $G$ is the null graph (no edges).
(b) $G$ is the complete graph (all possible $\binom{n}{2}$ edges exist).
(c) $G$ is the star graph (a tree with a root connected to all other $n - 1$ vertices).
(d) $G$ is the path graph (a chain of $n$ vertices and $n - 1$ edges).
(e) $G$ is the cycle graph (a cycle of $n$ vertices and $n$ edges).
(f) $G$ is a complete bipartite graph that has $x$ vertices on one side and $y$ vertices on the other side for $x + y = n$ (all possible $x \cdot y$ edges between the two sides exist).

A minimal vertex cover in a graph is a vertex cover that does not contain another vertex cover (deleting any vertex from the set would create a set of vertices that is not a vertex cover).

(b) A minimal vertex cover is not always a minimum vertex cover. Demonstrate this with a simple example.
Finding a minimum vertex cover in a graph is an NP-Complete problem whereas finding a minimal vertex cover is not.

(c) Describe a simple algorithm that finds one of the minimal vertex covers in a graph. What is the worst case running time of your algorithm if the graph is maintained with adjacency lists? What is the worst case running time of your algorithm if the graph is maintained with an adjacency matrix? Explain your answers.

2.1.14 Clique / Triangle (GC Exam 2006)

Let $G$ be a simple undirected graph with $n \geq 1$ vertices and $m$ edges.

A clique in $G$ is a set of vertices such that all possible edges between any pair of vertices in the clique exist. A maximum clique in $G$ is a clique that has the largest number of vertices among all the cliques of $G$.

1. If $G$ does not have a clique of size $k$, can $G$ have a clique of size $k + 1$? Explain.
2. What is the size of the maximum clique in a bipartite graph? Explain.
3. Describe an $O(n^k)$-time algorithm to find a clique of size $k$ in $G$ if such a clique exists. Use simple words and do not write program codes. Explain why the complexity of your algorithm is $O(n^k)$.
4. Describe an algorithm to find a triangle (a clique of size 3) in $G$ if exists. Your algorithm should have a better than $O(n^3)$-time complexity (for example, $O(nm)$ is better than $O(n^3)$). Use simple words and do not write program codes. Justify your complexity claim.

2.1.15 Analyzing Classes of Graphs: Forests and Trees (GC Exam 2007)

Let $G$ be a simple undirected graph with $n \geq 1$ vertices and $m$ edges. A forest is a graph with no cycles. A tree is a connected forest. Consider the following nine classes of graphs:

1. Connected graphs with $m = n - 1$ edges.
2. Bipartite graphs with $m = n - 1$ edges.
3. Graphs for which between any pair of vertices there exists at most one path.
4. Graphs with no cycles for which adding any edge would create exactly one cycle.
6. Graphs with no triangles.
7. Graphs that can be colored with at most two colors.
8. Connected graphs for which omitting any edge would disconnect them.
9. Connected graphs for which the BFS traversal tree is identical to the DFS traversal tree when both traversals start with the same vertex and both traversals consider the same order on the vertices.

Which one of the following three statements is true for each one of the above nine classes of graphs? Justify your answers with 2-3 sentences.

1. All the graphs in this class are trees.
2. All the graphs in this class are forests but not all of them are trees.
3. Some graphs in this class are not trees.
2.1.16 Identify a Celebrity (GC Exam 2008)

Story: A celebrity among a group of \( n \) people is a person who knows nobody but is known to everybody else. The task is to identify a celebrity by only asking questions to people of the form: “Do you know him/her?”

1. Model the story with a directed graph that is represented with an adjacency matrix. In particular, answer the following questions:
   (a) Who are the vertices of this graph?
   (b) What is the meaning of a directed edge in this graph?
   (c) What is a celebrity?
   (d) What is the equivalent in graph terms to the defined question?
   (e) What is the complexity objective?
   (f) Can a group have more than one celebrity?

2. Design an efficient algorithm to identify a celebrity or determine that the group has no such person. How many questions does your algorithm need in the worst case? Be as accurate as possible and justify your answer.

2.1.17 Tournament (GC Exam 2009)

A tournament is a directed simple graph with \( n \) vertices and \( m = n(n-1)/2 \) edges. Between any pair of vertices \( u \) and \( v \), either the edge \((u \to v)\) exists or the edge \((v \to u)\) exists. A tournament is cyclic if there exists a directed cycle in the tournament.

1. Prove that if a tournament is cyclic, then it must have a directed triangle which is a directed cycle of size 3.

2. Let \( T \) be a directed tournament with \( n \) vertices that is represented by the \( n \times n \) adjacency matrix. Let \( v_1 \to v_2 \to \cdots \to v_k \to v_1 \) be a simple directed cycle with \( k \) distinct vertices in the tournament \( T \).
   Describe an efficient algorithm that finds a directed triangle in the tournament \( T \). The only complexity parameter is the number of times the algorithm access the adjacency matrix.

2.1.18 Minimal Dominating Set (GC Exam 2010)

Let \( G \) be a simple and undirected graph with \( n \) vertices. Assume \( G \) is represented by the adjacency matrix \( A \). (In an adjacency matrix \( A \), if \( A[i,j] = 1 \) then there is an edge between vertex \( i \) and vertex \( j \) and if \( A[i,j] = 0 \) then there is no edge between vertex \( i \) and vertex \( j \).)

A set of vertices \( D \) is a dominating set if for every vertex \( u \) not in \( D \) there exists a vertex \( v \) in \( D \) such that the edge \((u,v)\) belongs to \( G \).

A dominating set \( D \) is a minimal dominating set if any proper subset \( D' \) of \( D \) is not a dominating set.

1. Let \( D \) be a set of vertices of size \( k \). Describe an efficient algorithm to verify if \( D \) is a dominating set in \( G \). What is the time-complexity of your algorithm as a function of \( n \) and \( k \)?

2. Let \( D \) be a set of vertices of size \( k \). Describe an efficient algorithm to verify if \( D \) is a minimal dominating set in \( G \). What is the time-complexity of your algorithm as a function of \( n \) and \( k \)?

- Be accurate with your complexity claims. State the complexity as a function of both \( k \) and \( n \) since \( k \) could be much smaller than \( n \).
- Justify the correctness of your algorithms and your complexity claims.
2.1.19 Simple Path / Simple Cycle in a $d$-Regular Graph (GC Exam 2010)

Let $G$ be a simple and undirected graph that is represented by an adjacency matrix. The degree of a vertex in $G$ is the number of its neighbors. $G$ is a $d$-regular graph if the degree of all vertices is exactly $d$.

1. A simple path in $G$ is a path for which all the vertices are different. The length of a simple path is the number of vertices in the path. Describe an algorithm that always finds a simple path of length at least $d + 1$ in a $d$-regular graph. Explain why your algorithm never fails to find such a path. What is the worst-case complexity (number of times the algorithm accesses the adjacency matrix) of your algorithm as a function of $n$ and $d$? Justify your answer.

2. A simple cycle in $G$ is a cycle for which all the vertices are different. The length of a simple cycle is the number of vertices in the cycle. Describe an algorithm that finds a simple cycle of length at least $d + 1$ in a $d$-regular graph. Explain why your algorithm always succeeds. What is the worst-case complexity (number of times the algorithm accesses the adjacency matrix) of your algorithm as a function of $n$ and $d$? Justify your answer.
2.2 Hints

Individual subsections with hints for each problem here.
2.3 Solutions

Individual subsection with solutions for each problem here.
Chapter 3

Other Problems

3.1 Problems

3.1.1 Integer Exponentiation (Final Exam 2007)

Given two positive integers $n$ and $m$ the goal is to compute $n^m$ with as minimum as possible integer multiplications.

1. Describe a correct algorithm to compute $n^m$ and analyze its complexity (number of multiplications).
   Justify the correctness of your algorithm and explain your complexity analysis.

2. Assume that $m = 2^k$ is a power of 2 where $k \geq 1$ is a known integer.
   Describe a more efficient correct algorithm to compute $n^m$ and analyze its complexity (number of multiplications).
   Justify the correctness of your algorithm and explain your complexity analysis.

3. Describe a more efficient correct algorithm to compute $n^m$ for any value of $m$ and analyze its complexity (number of multiplications).
   Justify the correctness of your algorithm and explain your complexity analysis.

3.1.2 Bin-Packing (GC Exam 2005)

The input to the bin-packing problem is a set of $n \geq 1$ items: $I_1, I_2, \ldots, I_n$ whose weights are $0 < w_1, w_2, \ldots, w_n \leq 1$ respectively (the weight of item $I_i$ is a real number $0 < w_i \leq 1$).

A legal packing is a set of items whose total weight is less or equal to one. The goal is to find a legal packing that optimizes a given objective function. Consider the following two objectives:

**COUNT:** Pack as many items as possible. Their total weight is unimportant as long as it is not more than 1.

**FULL:** Pack the bin with items whose total weight is as close to 1 as possible. The number of packed items is unimportant and one item whose weight is 1 (if such exists) is an optimal packing.

The following is a schematic greedy algorithm:
• Order the $n$ items according to some predefined rule and consider the items following this order.

• When considering item $I$: pack it if the sum of the already packed items plus the weight of item $I$ is still not more than 1. Otherwise, ignore item $I$.

Consider the following two orders:

**Lightest First**: The items are ordered from the lightest to the heaviest.

**Heaviest First**: The items are ordered from the heaviest to the lightest.

In the following five questions you need to show how “good” the greedy algorithms generated by these two orders are while applied to the two objectives functions.

1. Prove that the **Lightest First** order is optimal for the objective function **COUNT**.

2. Show that the **Lightest First** order is not optimal and could be “very bad” for the objective function **FULL**.

3. Show that the **Heaviest First** order is not optimal and could be “very bad” for the objective function **COUNT**.

4. Show that the **Heaviest First** order is not optimal for the objective function **FULL**.

5. Prove that the **Heaviest First** order finds a solution whose total weight is at least half the total weight of the optimal solution for the objective function **FULL**.
3.2 Hints

Individual subsections with hints for each problem here.
3.3 Solutions

Individual subsection with solutions for each problem here.