Description: This document includes many of the problems given in the last two decades in the midterm and final exams of the Brooklyn College course CISC 7200X Analysis of Algorithms that was taught by Amotz Bar-Noy at the CIS department.

Disclaimer: Knowing and memorizing the solutions to all of these problems does not guarantee success in future exams.

General Remarks:

• There are three main categories of problems:
  – Problems about growth of function, recursions, and running times of Pseudocodes.
  – Problems about arrays of numbers that are partitioned by their complexities.
  – Problems about graphs.
In each category or sub-category, the problems are not ordered by their difficulty. They are arbitrarily ordered

• Some of the problems were modified, some of them were combined, and some of them were partitioned into several parts.

• Some of the problems are very simple and some are composed of many parts. This depends on the other problems in the exam.

• When it is required to design an algorithm, usually it is also required to prove (or justify) why the algorithm is correct, to find the running time (complexity) of the algorithm, and to justify the running time claim. If the target running time is given, it is still required to justify the running time claim.

• Full credit is granted for the most efficient algorithm. Partial credit is granted for non-efficient solutions. However, the solutions must be correct.

• High level descriptions are preferable. Pseudo-codes have higher risks to have mistakes.
1 Growth of Functions, Recursions, and Pseudocodes

1.1 Growth of Functions

1. TRUE (T) or FALSE (F)?

(a) \( f = O(g) \) implies \( f = \Omega(g) \).
(b) \( f = \Theta(g) \) implies \( f = O(g) \).
(c) \( (f = O(g) \text{ and } f = \Omega(g)) \) implies \( f = \Theta(g) \).
(d) \( f = \Theta(g) \) implies \( f = O(g) \).
(e) \( f = \Omega(g) \) implies \( f = O(g) \).
(f) \( f = O(g) \) implies \( f = \Theta(g) \).
(g) \( f = \Omega(g) \) implies \( f = \Theta(g) \).
(h) \( f = \Theta(g) \) implies \( (f = O(g) \text{ and } f = \Omega(g)) \).

2. Fill in the following table with one of the three: \( O \), \( \Omega \), \( \Theta \).

Remark: If \( f = \Theta(g) \) then \( f = O(g) \) and \( f = \Omega(g) \) are wrong answers.

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( ??? )</th>
<th>( g(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>( n )</td>
<td>( n^2 )</td>
</tr>
<tr>
<td>b</td>
<td>( n^2 )</td>
<td>( n )</td>
</tr>
<tr>
<td>c</td>
<td>( 2n )</td>
<td>( 5n )</td>
</tr>
<tr>
<td>d</td>
<td>( 1000000n )</td>
<td>( (1/100000)n )</td>
</tr>
<tr>
<td>e</td>
<td>( \log_2(n) )</td>
<td>( \log_2(n) )</td>
</tr>
<tr>
<td>f</td>
<td>( \log_2(n) )</td>
<td>( \log_{10}(n) )</td>
</tr>
<tr>
<td>g</td>
<td>( n \log_2(n) )</td>
<td>( n/\log_2(n) )</td>
</tr>
<tr>
<td>h</td>
<td>( 2^n )</td>
<td>( n^{100} )</td>
</tr>
<tr>
<td>i</td>
<td>( 2^n )</td>
<td>( 3^n )</td>
</tr>
<tr>
<td>j</td>
<td>( 2^n )</td>
<td>( n! )</td>
</tr>
</tbody>
</table>

3. Which of the following 10 functions are \( O(n) \)? Which are \( \Omega(n) \)? Which are \( \Theta(n) \)?

Remark: If \( f = \Theta(n) \) then \( f = O(n) \) and \( f = \Omega(n) \) are wrong answers.

<table>
<thead>
<tr>
<th>( f(n) )</th>
<th>( ??? )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>( 2^n )</td>
</tr>
<tr>
<td>b</td>
<td>( n^2 )</td>
</tr>
<tr>
<td>c</td>
<td>( 2n )</td>
</tr>
<tr>
<td>d</td>
<td>( n/\log_2(n) )</td>
</tr>
<tr>
<td>e</td>
<td>( \log_3(n) )</td>
</tr>
<tr>
<td>f</td>
<td>( 100 \log_2(n) \log_2(n) )</td>
</tr>
<tr>
<td>g</td>
<td>( n \log_2(n) )</td>
</tr>
<tr>
<td>h</td>
<td>( 10^{10}n/100^{100} )</td>
</tr>
<tr>
<td>i</td>
<td>( n^n )</td>
</tr>
<tr>
<td>j</td>
<td>( n! )</td>
</tr>
</tbody>
</table>
1.2 Recursions

1. Below there are definitions for ten recursive formulas \( R_1, R_2, \ldots, R_{10} \). Match them with their solutions \( S_1, S_2, \ldots, S_{10} \).

**Remark I:** Each formula has a different solution.

**Remark II:** Ignore floors and ceilings by assuming that \( n \) is a power of 2 when \( T(n) \) is a function of \( T(n/2) \).

- \( R_1: T(1) = 1; \ T(n) = 4T(n/2) \)
- \( R_2: T(1) = 2; \ T(n) = T(n-1) + 2 \)
- \( R_3: T(1) = 2; \ T(n) = 2T(n-1) \)
- \( R_4: T(1) = 0; \ T(n) = T(n/2) + 1 \)
- \( R_5: T(1) = 0; \ T(n) = 2T(n/2) + n \)
- \( R_6: T(1) = 1; \ T(n) = nT(n-1) \)
- \( R_7: T(1) = 1; \ T(n) = 2T(n/2) \)
- \( R_8: T(1) = 0; \ T(n) = T(n/2) + 2\log(n) - 1 \)
- \( R_9: T(1) = 1; \ T(n) = 8T(n/2) \)
- \( R_{10}: T(1) = 1; \ T(n) = 2^n(T(n/2))^2 \)

<table>
<thead>
<tr>
<th>recursion</th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
<th>( R_4 )</th>
<th>( R_5 )</th>
<th>( R_6 )</th>
<th>( R_7 )</th>
<th>( R_8 )</th>
<th>( R_9 )</th>
<th>( R_{10} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>solution</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

2. Let \( n = 2^k \) be a power of 2 for \( k \geq 0 \) and let \( T(n) \) be a recursive formula defined as follows

\[
T(1) = 1; \quad T(n) = aT(n/2) + f(n)
\]

for a constant \( a \) that can be 1, 2, or 4 and a function \( f(n) \) that can be 0, 1, or \( n \).

Match the nine possible solutions for \( T(n) \) with the nine possible combinations of \( a \) and \( f(n) \). Note that you need to assign \( 2n - 1 \) to two different recursive formulas.

- \( T_1(n) = 1 \)
- \( T_2(n) = \log_2(n) + 1 \)
- \( T_3(n) = n \)
- \( T_4(n) = 2n - 1 \)
- \( T_5(n) = 2n - 1 \)
- \( T_6(n) = n(\log_2(n) + 1) \)
- \( T_7(n) = n^2 \)
- \( T_8(n) = \frac{4n^2 - 1}{3} \)
- \( T_9(n) = 2n^2 - n \)

<table>
<thead>
<tr>
<th>( T(n) ) formula</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( T_5 )</th>
<th>( T_6 )</th>
<th>( T_7 )</th>
<th>( T_8 )</th>
<th>( T_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(n) ) and a combination</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
3. Solve the following recursive formulas. Find the exact formula for $T(n)$ as a function of $n$.

(a) $n \geq 1$ is a power of 2 integer.
\[
T(1) = 0 \\
T(n) = 2T(n/2) + n
\]

(b) $n \geq 1$ is a power of 2 integer.
\[
T(1) = 0 \\
T(n) = T(n/2) + 100
\]

(c) $n \geq 1$ is an integer.
\[
T(1) = 1 \\
T(n) = T(n - 1) + 1
\]

(d) $n \geq 1$ is an integer.
\[
T(1) = 1 \\
T(n) = T(n - 1) + n
\]

(e) $n \geq 1$ is a power of 2 integer.
\[
T(1) = 1 \\
T(n) = 4T(n/2)
\]

(f) $n \geq 1$ is a power of 2 integer.
\[
T(1) = 1 \\
T(n) = 2T(n/2) + n
\]

(g) $n \geq 1$ is a power of 3 integer.
\[
T(1) = 0 \\
T(n) = T(n/3) + 5
\]

(h) $n \geq 1$ is an odd integer.
\[
T(1) = 1 \\
T(n) = T(n - 2) + 1
\]

(i) $n \geq 1$ is an integer.
\[
T(1) = 2 \\
T(n) = T(n - 1) + 2n
\]

(j) $n \geq 1$ is a power of 3 integer.
\[
T(1) = 1 \\
T(n) = 9T(n/3)
\]

4. Solve the following two double recursions for both $T(n)$ and $S(n)$. Find the exact formula for both $T(n)$ and $S(n)$ as a function of $n$.

(a) Assume that $n$ is a positive integer.
\[
T(1) = 1 \\
T(n) = S(n) + 1 \\
S(1) = 0 \\
S(n) = T(n - 1) + 1
\]

(b) Assume that $n$ is a power of 2 integer.
\[
T(1) = 0 \\
T(n) = 2S(n/2) \\
S(1) = 1 \\
S(n) = T(n) + n
\]
1.3 Running Times of Psaudocodes

1. For the following procedures, find the exact value of \( c \) when the procedure terminates.

(a) \( f(n) \) (* \( n > 0 \) is an integer *)
\[
c = 0 \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad \text{for } j = i \text{ to } n \text{ do} \\
\quad \quad c := c + 1
\]

(b) \( f(n) \) (* \( n > 0 \) is an even integer number *)
\[
c = 0 \\
\text{for } i = 1 \text{ to } n/2 \text{ do} \\
\quad \text{for } j = n/2 \text{ to } n \text{ do} \\
\quad \quad c := c + 1
\]

2. For the following procedures, find the exact value of \( c \) when the procedure terminates.

(a) \( f(n) \) (* \( n > 0 \) is an integer *)
\[
c = 0 \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad \text{for } j = 1 \text{ to } 2n \text{ do} \\
\quad \quad \text{for } k = 1 \text{ to } 3n \text{ do} \\
\quad \quad \quad c := c + 1
\]

(b) \( f(n) \) (* \( n > 0 \) is an integer number *)
\[
c = 0 \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad \text{for } j = 1 \text{ to } i \text{ do} \\
\quad \quad \text{for } k = 1 \text{ to } j \text{ do} \\
\quad \quad \quad c := c + 1
\]

3. For the following procedures, find the exact value of \( c \) when the procedure terminates.

(a) \( f(n) \) (* \( n = k^2 \) is a positive square integer *)
\[
c = 0 \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad \text{if } i \text{ is a square number} \\
\quad \quad \text{then } c := c + 1
\]

(b) \( f(n) \) (* \( n \) is a power of 2 integer *)
\[
c = 0 \\
\text{while } n > 1 \text{ do} \\
\quad n := n/2 \\
\quad c := c + 1
\]

(c) \( f(n) \) (* \( n > 1000 \) is a power of 2 *)
\[
c = 0 \\
\text{while } n > 512 \text{ do} \\
\quad n := n/2 \\
\quad c := c + 1
\]
1.4 Multiple Choice Problems

1. Mark the correct answer in the following multi-choice problems.

(a) Which of the following describes most accurately the relationship between \( \log_2(n^2) \) and \( \log_{10}(\sqrt{n}) \)?
   i. \( \log_2(n^2) = o(\log_{10}(\sqrt{n})) \)
   ii. \( \log_2(n^2) = O(\log_{10}(\sqrt{n})) \)
   iii. \( \log_2(n^2) = \Theta(\log_{10}(\sqrt{n})) \)
   iv. \( \log_2(n^2) = \Omega(\log_{10}(\sqrt{n})) \)
   v. \( \log_2(n^2) = \omega(\log_{10}(\sqrt{n})) \)

(b) Which of the following describes most accurately the relationship between \( n \log_2(n) \) and \( n \)?
   i. \( n \log_2(n) = o(n) \)
   ii. \( n \log_2(n) = O(n) \)
   iii. \( n \log_2(n) = \Theta(n) \)
   iv. \( n \log_2(n) = \Omega(n) \)
   v. \( n \log_2(n) = \omega(n) \)

(c) For a power of 3 integer \( n \), what is the solution to the recursive formula
   \( T(1) = 1 \) and \( T(n) = T(n/3) + 1 \)?
   i. \( T(n) = \Theta(\log(n)) \).
   ii. \( T(n) = \Theta(\log^2(n)) \).
   iii. \( T(n) = \Theta(n) \).
   iv. \( T(n) = \Theta(n \log(n)) \).
   v. \( T(n) = \Theta(n^2) \).

(d) For a power of 2 integer \( n \), what is the solution to the recursive formula
   \( T(1) = 1 \) and \( T(n) = T(n/2) + n \)?
   i. \( T(n) = \Theta(\log(n)) \).
   ii. \( T(n) = \Theta(\log^2(n)) \).
   iii. \( T(n) = \Theta(n) \).
   iv. \( T(n) = \Theta(n \log(n)) \).
   v. \( T(n) = \Theta(n^2) \).

(e) For any positive integer \( n \), what is the solution to the recursive formula
   \( T(1) = 1 \) and \( T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) \)?
   i. \( T(n) = \Theta(\log(n)) \).
   ii. \( T(n) = \Theta(\log^2(n)) \).
   iii. \( T(n) = \Theta(n) \).
   iv. \( T(n) = \Theta(n \log(n)) \).
   v. \( T(n) = \Theta(n^2) \).
(f) For any positive integer \( n \), what is the solution to the recursive formula

\[
T(1) = 1 \quad \text{and} \quad T(n) = T(n-1) + n?
\]

i. \( T(n) = \Theta(\log(n)) \).
ii. \( T(n) = \Theta(\log^2(n)) \).
iii. \( T(n) = \Theta(n) \).
iv. \( T(n) = \Theta(n \log(n)) \).
v. \( T(n) = \Theta(n^2) \).

(g) Algorithm \( A \) solves Problem \( P \) with a worst-case tight bound of \( \Theta(n) \) on its running time. Which of the following statements may be true?

i. For infinitely many values of \( n \), there are instances of \( P \) of length \( n \) for which the running time of \( A \) is \( n^2 \).
ii. For all \( n \), and for all instances of \( P \) of length \( n \), the running time of \( A \) is \( \Omega(n \log n) \).
iii. For infinitely many values of \( n \), there are instances of \( P \) of length \( n \) for which the running time of \( A \) is \( \log(n) \).
iv. For all \( n \), and for all instances of \( P \) of length \( n \), the running time of \( A \) is \( O(\sqrt{n}) \).
v. None of the above.

(h) For Problem \( P \) with parameters \( k \) and \( n \), Algorithm \( A \) has a worst-case tight bound of \( \Theta(kn) \) on its running time and Algorithm \( B \) has a worst-case tight bound of \( \Theta(k^2 \sqrt{n}) \) on its running time. For which value of \( k \) both algorithms have the same asymptotic complexity?

i. \( k = \Theta(1) \)
ii. \( k = \log_2(n) \)
iii. \( k = \sqrt{n} \)
iv. \( k = n \)
v. \( k = n \sqrt{n} \)

(i) Which of the following statements is always correct?

i. If \( f(n) = O(g(n)) \) then \( f(n) \leq g(n) \) for all \( n \geq 1 \).
ii. If \( g(n) = \Omega(f(n)) \) then \( g(n) \geq f(n) \) for all \( n \geq 2^{100} \).
iii. If \( f(n) = O(g(n)) \) and \( g(n) = \Omega(h(n)) \) then \( f(n) = \Theta(h(n)) \).
iv. If \( f(n) = O(g(n)) \) and \( g(n) = O(h(n)) \) then \( f(n) = O(h(n)) \).
v. None of the above.

(j) What is the time complexity of the following segment of code?

\[
x = 1 \\
\text{while } x \leq n^{100} \text{ do} \\
x = 2x
\]

i. \( \Theta(1) \)
ii. \( \Theta(\log(n)) \)
iii. \( \Theta(n) \)
iv. \( \Theta(n \log(n)) \)
v. \( \Theta(n^2) \)

7
2. Mark the correct answer in the following multi-choice problems.

(a) What is the solution to the recursive formula
\[ T(1) = 1 \text{ and } T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)? \]

i. \( T(n) = \Theta(\log(n)) \).
ii. \( T(n) = \Theta(\log^2(n)) \).
iii. \( T(n) = \Theta(n) \).
iv. \( T(n) = \Theta(n \log(n)) \).
v. \( T(n) = \Theta(n^2) \).

(b) Algorithm \( A \) solves Problem \( P \) with a worst-case tight bound of \( \Theta(n) \) on its running time. Which of the following statements is always wrong?

i. There are instances of \( P \) of length \( n \) for which the running time of \( A \) is \( n^2 \).
ii. For all instances of \( P \) of length \( n \), the running time of \( A \) is \( \Omega(n) \).
iii. There are instances of \( P \) of length \( n \) for which the running time of \( A \) is \( \log(n) \).
iv. For all instances of \( P \) of length \( n \), the running time of \( A \) is \( \Theta(n) \).

(c) For Problem \( P \) with parameters \( k \) and \( n \), Algorithm \( A \) has a worst-case tight bound of \( \Theta(kn) \) on its running time and Algorithm \( B \) has a worst-case tight bound of \( \Theta(k^2 \sqrt{n}) \) on its running time. For which value of \( k \) both algorithms have the same asymptotic complexity?

i. \( k = \Theta(1) \)
ii. \( k = \Theta(\sqrt{n}) \).
iii. \( k = \Theta(n) \).
iv. \( k = \Theta(n\sqrt{n}) \).

(d) Which of the following statements is always correct?

i. If \( f(n) = O(g(n)) \) then \( f(n) \leq g(n) \) for all \( n \geq 1 \).
ii. If \( f(n) = O(g(n)) \) then \( f(n) \leq g(n) \) for all \( n \geq 2^{100} \).
iii. If \( f(n) = O(g(n)) \) and \( g(n) = \Omega(h(n)) \) then \( f(n) = \Theta(h(n)) \).
iv. If \( f(n) = O(g(n)) \) and \( g(n) = O(h(n)) \) then \( f(n) = O(h(n)) \).

(e) What is the time complexity of the following segment of code?
\[
x = 1 \\
\text{while } x \leq n^{100} \text{ do} \\
x = 2x
\]

i. \( \Theta(1) \).
ii. \( \Theta(\log(n)) \).
iii. \( \Theta(n) \).
iv. \( \Theta(n \log(n)) \).
v. \( \Theta(n^2) \).
3. Mark the correct answer in the following multi-choice problems.

(a) What is the sum of the following sequence?

$$\sum_{i=1}^{i=n} 2i = 2 + 4 + 6 + \cdots + (2n - 2) + 2n$$

i. $4n$
ii. $2n \log_2(n)$
iii. $n(n + 1)/2$
iv. $n(n + 1)$
v. $2n^2$

(b) For $n = 2^k$, what is the sum of the following sequence?

$$\sum_{i=0}^{i=k} 3n^{2^i} = 3n + \frac{3n}{2} + \frac{3n}{4} + \cdots + 12 + 6 + 3$$

i. $4n - 2$
ii. $6n - 3$
iii. $n \log_2(n) - 3$
iv. $n^2 - 3$
v. $3n(n + 1)/2 - 6$

(c) What is the solution to the following recursive formula?

$$T(1) = 2 \quad \text{and} \quad T(n) = T(n-1) + 2n$$

i. $T(n) = \Theta(\log(n))$
ii. $T(n) = \Theta(\log^2(n))$
iii. $T(n) = \Theta(n)$
iv. $T(n) = \Theta(n \log(n))$
v. $T(n) = \Theta(n^2)$

(d) For $n = 2^k$, what is the solution to the following recursive formula?

$$T(1) = 3 \quad \text{and} \quad T(n) = T(n/2) + 3n$$

i. $T(n) = \Theta(\log(n))$
ii. $T(n) = \Theta(\log^2(n))$
iii. $T(n) = \Theta(n)$
iv. $T(n) = \Theta(n \log(n))$
v. $T(n) = \Theta(n^2)$

(e) Algorithm A solves problem $P$ with complexity $\Theta(k^2)$ and Algorithm B solves problem $P$ with complexity $\Theta(n + k)$. For which values of $n$ and $k$ both algorithms have the same complexity?

i. $n = \Theta(1)$
ii. $n = \Theta(\log(k))$
iii. $n = \Theta(k)$
iv. $n = \Theta(k^2)$
v. $n = \Theta(2^k)$
2 Array Problems

Remarks:

- Unless stated otherwise, the complexity model is the comparison model. That is, radix-sort like sorting algorithms and hashing cannot be used in the solution. For these problems, the running time (complexity) is the number of comparisons performed by the algorithm.

- The complexity depends on the array size which is usually $n$. But for some problems there exists another parameter that might be part of the complexity analysis. Most of the arrays contain $n$ integers for $n \geq 1$. In some problems, $n$ could be larger than some constant $c$ (for example, $c = 2, 3, 4, \ldots$) and in some problems, $n$ is a power of 2 or a power of another constant.

- An arbitrary array is an array that is not necessarily sorted. The integers in the arrays could be negative, positive, or 0 unless stated otherwise and they do not have to be distinct unless stated otherwise.

2.1 $\Theta(\log n)$ Problems

1. Assume $n$ is a power of 2.
   Let $x < y$ be two positive integers.
   Describe an efficient algorithm that finds an index $i$ (if exists) such that $A[i]$ is a number between $x$ and $y$: $x \leq A[i] \leq y$.
   What is the exact worst-case number of comparisons made by your algorithm?

2. Assume $n$ is a power of 2.
   Let $x$ be a positive integer.
   Describe an efficient algorithm that finds an index $i$ (if exists) such that $A[i]$ is a number between $x$ and $2x$: $x \leq A[i] \leq 2x$.
   What is the exact worst-case number of comparisons made by your algorithm?

3. Assume $n$ is a power of 2.
   Let $x \leq y$ be two positive integers.

   (a) Describe an efficient algorithm that returns “TRUE” if one of the numbers $x, x+1, \ldots, y$ appears in the array.
   What is the exact worst-case number of comparisons made by your algorithm?

   (b) Describe an efficient algorithm that returns “TRUE” if all the numbers $x, x+1, \ldots, y$ appear in the array.
   What is the exact worst-case number of comparisons made by your algorithm?

4. Let $A = A[1] \leq A[2] \leq \cdots \leq A[n]$ be a sorted array of $n$ numbers. Let $x$ be a number. Design an $O(\log(n))$ algorithm to find the number of times $x$ appears in the array.
5. **Definition:** An array $A$ with $n$ distinct integers is called a hill array if there exists an index $m$ such that the array is in an ascending order from index 1 to index $m$ and in a descending order from index $m$ to index $n$:


**Example:** $A = [2, 4, 6, 8, 7, 5, 3, 1]$ is a hill array for which $m = 4$.

**Example:** $A = [13, 34, 21, 8, 5, 3, 2, 1]$ is a hill array for which $m = 2$.

**Example:** $A = [1, 2, 8, 5, 3, 13, 21, 34]$ is not a hill array.

Note that an ascending-sorted array is a hill array for which $m = n$ and that a descending-sorted array is a hill array for which $m = 1$.

Let $A$ be a hill array. Describe an efficient algorithm to find the index $m$.

What is the worst-case number of comparisons made by your algorithm?


8. Let $n \geq 1$ be an integer and let $1 \leq x \leq n$ be an **unknown** integer in the range $[1..n]$. 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 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 performed by your modified search as a function of $n$.

**Remark:** To avoid **ceilings**, in each case there is an assumption about the value of $n$.

(a) $x$ is an **odd** number. $n = 2^k$ is a power of 2.

(b) $x = y^2$ is a **square** number. $n = (2^k)^2$ is a square of a power of 2.

(c) $x = 2^y$ is a **power** of 2. $n = 2^{(2^k)}$ is a power of 2 and the exponent is also a power of 2.
9. The following *Trinary-Search* procedure looks for $x$ in the range $[f..\ell]$ for some positive integers $1 \leq f < \ell \leq n$. Initially, $\ell = 1$ and $f = n$ for the range $[1..n]$. Assume that $n$ is a power of 3 and that the size of the range, $\ell - f + 1$, is always 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: $[f..(f + m - 1)]$, $[(f + m)..(\ell - m)]$, and $[(\ell - m + 1)..\ell]$. For example, the range $[10..18]$ 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 “YES”, then the search continues recursively in the first range: $[f..(f + m - 1)]$.
- If “NO”, then the second comparison is: $x \leq \ell - m$ (does $x$ belong to the second range?).
- If “YES”, then the search continues recursively in the second range: $[(f + m)..(\ell - m)]$.
- If “NO”, then the search continues recursively in the third range: $[(\ell - m + 1)..\ell]$.

(a) For the range $[1..n]$, what is the exact number of comparisons between $x$ and a number in the range that are performed by Trinary-Search?

i. $n = 3$ and $x = 1, 2, 3$.
ii. $n = 9$ and $x = 1, \ldots, 9$.
iii. $x = n$ for any $n$.
iv. $x = 1$ for any $n$.

(b) Let $T(n)$ be the worst case complexity (number of comparisons) of Trinary-Search. Write the initial value and the recursive formula for $T(n)$.

What is the solution of this recursion using the $\Theta$-notation.

What is the exact solution of this recursion?
2.2 Θ(n) Problems

1. Let \( A = A[1], A[2], \ldots, A[n] \) be an arbitrary array of \( n \) (\( 4 \leq n \)) distinct positive integers.
   Describe an efficient algorithm that finds the two largest integers and the two smallest integers in \( A \).
   What is the exact worst-case number of comparisons made by your algorithm?

2. For \( n \geq 1 \), let \( A \) and \( B \) be two sorted arrays each of \( n \) of distinct positive integers:

   (a) Describe an efficient algorithm that finds an integer (if exists) that appears in both arrays. What is the worst-case number of comparisons made by your algorithm?

   (b) Describe an efficient algorithm that finds (if exists) an integer that appears in one of the arrays but not in both. What is the worst-case number of comparisons made by your algorithm?

3. Let \( A = A[1], \ldots, A[n] \) be an array of \( n \) positive integers. Let \( S = A[1] + \cdots + A[n] \) be the sum of all the numbers in the array. For \( 1 \leq i \leq n \), let

   be the sum of all the numbers in the array except \( A[i] \). Design a linear time \( (O(n)) \) algorithm to compute \( S[1], \ldots, S[n] \) only with plus operations (The algorithm cannot use the minus operation).

   
   Example: For \( n = 11, k = 50, \) and \( A = [1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144] \), the output is 4 and 9 because \( A[9] - A[4] = 55 - 5 = 50 \).

   What is the complexity (number of comparisons) of your algorithm?


   (a) Describe an efficient algorithm that finds two indices \( i < j \) (if exist) such that \( A[j] = A[i] + 1 \).
   What is the worst-case number of comparisons of the type \( A[j] = A[i] + 1 \) that are performed by your algorithm?

   (b) Describe an efficient algorithm that finds two indices \( i < j \) (if exist) such that \( A[j] = (A[i])^2 \).
   What is the worst-case number of comparisons of the type \( A[j] = (A[i])^2 \) that are performed by your algorithm?

6. Let \( A = A[1], A[2], \ldots, A[n] \) be an arbitrary array of \( n \) distinct integers. For any index \( i \) between 1 and \( n \), define \( B[i] \) to be the smallest index \( j \) such that \( A[j] > A[i] \) and \( i < j \leq n \), if such an index exists. Otherwise, \( B[i] = n + 1 \).
   
   Design an efficient algorithm that constructs the array \( B \) for a given array \( A \).

   What is the worst-case number of comparisons performed by your algorithm?
7. Let \( A = A[1], A[2], \ldots, A[n] \) be an arbitrary array of \( n \) distinct integers. Assume that \( n > 2 \) is an odd number.

For a given integer \( 1 \leq k < n/2 \), describe an efficient algorithm that finds the \( k \) smallest integers in \( A \) that are larger than the median.

**Example:** For \( n = 11 \), \( k = 3 \), and \( A = [3, 55, 1, 144, 13, 21, 8, 5, 89, 2, 34] \), the median is 13 and the 3 smallest integer that are larger than 13 are 55, 21, 34.

What is the complexity of your algorithm?

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

\[
\begin{align*}
\end{align*}
\]

Describe a **linear** time \((O(n))\) algorithm that sorts bitonic arrays.

9. **Definition:** An array \( A \) of size \( n \geq 2 \) is called a \( k \)-sorted array for an integer \( 0 \leq k < n \), if for any index \( i \), if \( j \) is the index of \( A[i] \) when the array is sorted then \( |i - j| \leq k \). In other words, \( A[i] \) is at most \( k \) locations away from its position in the sorted version of array \( A \).

**Example:** Consider the array \( A = [5, 3, 8, 1, 21, 2, 13, 34] \) the sorted version of which is \( A' = [1, 2, 3, 5, 8, 13, 21, 34] \). It follows that

\[
\begin{align*}
A[1] &= 5 \text{ is 3 locations away from its position at } A'[4], \\
A[2] &= 3 \text{ is 1 location away from its position at } A'[3], \\
A[3] &= 8 \text{ is 2 locations away from its position at } A'[5], \\
A[4] &= 1 \text{ is 3 locations away from its position at } A'[1], \\
A[5] &= 21 \text{ is 2 locations away from its position at } A'[7], \\
A[6] &= 2 \text{ is 4 locations away from its position at } A'[2], \\
A[7] &= 13 \text{ is 1 location away from its position at } A'[6], \\
A[8] &= 34 \text{ is 0 locations away from its position at } A'[8].
\end{align*}
\]

As a result, \( A \) is a 4-sorted array but not a 3-sorted array.

**Remark:** Note that a sorted array is a 0-sorted array and that any array is for sure an \((n - 1)\)-sorted array but maybe a \(k\)-sorted array for \( k < n - 1 \).

Let \( n \) be an even number. Describe an efficient algorithm that transforms an arbitrary array \( A \) of \( n \) integers to be an \( n/2 \)-sorted array.

What is the worst-case number of comparisons made by your algorithm?

10. Let \( A = A[1], \ldots, A[n] \) and \( B = B[1], \ldots, B[n] \) be two arbitrary arrays. Define \( A - B \) to be the sum of all \( n^2 \) differences \( A[i] - B[j] \) for \( 1 \leq i \leq n \) and \( 1 \leq j \leq n \).

\[
A - B = \sum_{i,j \in \{1,\ldots,n\}} (A[i] - B[j]) .
\]

Let \( C = C[1], \ldots, C[2n] \) be an array with \( 2n \) distinct positive integers. Describe an efficient algorithm that partitions \( C \) into two arrays \( A \) and \( B \) each containing a different set of \( n \) integers from \( C \) such that \( A - B \) is maximum.

What is the worst-case number of comparisons made by your algorithm?

11. Let \( A = A_1 < A_2 < \cdots < A_n \) be a sorted array containing \( n \) distinct positive integers and let \( k \) be a positive integer.

Describe an \( O(n)\)-time algorithm that finds (if exist) two indices \( 1 \leq i, j \leq n \) such that \( A_i + A_j = k \).
12. Let $A = A[1], \ldots, A[n]$ be an arbitrary array containing $n$ positive integers.

A **majority** is a number that appears more than $n/2$ 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.


Let $Count$ be the number of pairs $i$ and $j$ for which $A[i] > B[j]$.

Describe an efficient algorithm to compute $Count$.

What is the worst-case number of comparisons performed by your algorithm?

14. Let $A = A[1], \ldots, A[n]$ be an arbitrary array containing $3n$ ($n \geq 1$) distinct integers.

Describe an efficient algorithm that partitions the $3n$ integers from $A$ into three arrays $B$, $C$, and $D$, each of size $n$. Array $D$ will contain the $n$ largest integers from $A$, array $B$ will contain the $n$ smallest integers from $A$, and array $C$ will contain the rest of the integers from $A$.

What is the worst-case number of comparisons performed by your algorithm?

15. Let $M$ be an $n \times n$ matrix containing $n^2$ distinct integers. The $n$ integers of each row of $M$ are sorted in an ascending order (from left to right) and the integers of each column of $M$ are sorted in an ascending order (from top to bottom).

Let $x$ be one of the numbers in $M$. The goal is to find the position of $x$ in $M$ using only comparisons between $x$ and integers in the matrix. That is, you need to find the indices $1 \leq i, j \leq n$ such that $M[i, j] = x$.

(a) Describe an algorithm that accomplishes this task with $O(n)$ comparisons.

(b) **Bonus** question: Prove that any algorithm that solves this problem must use $\Omega(n)$ comparisons.
16. **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 stack 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.

(a) Sort the following arrays with as few as possible flips. Specify the list of flips that sorts the arrays.

i. \([8, 7, 6, 5, 1, 2, 3, 4] \)

ii. \([8, 6, 4, 2, 1, 3, 5, 7] \)

iii. \([2, 8, 6, 4, 3, 5, 7, 1] \)

(b) Describe an algorithm to sort any array of size \( n \) with at most \( 2n - 3 \) flips.
2.3 $\Theta(n \log n)$ Problems

1. Let $A = A[1], A[2], \ldots, A[n]$ and $B = B[1], B[2], \ldots, B[n]$ be arbitrary arrays each containing $n$ distinct positive integers.

Design an efficient algorithm that determines if there exists an integer $k$ and reordering of both arrays such that after the reordering $A[i] + B[i] = k$ for all $1 \leq i \leq n$.

What is the worst-case number of comparisons performed by your algorithm?

2. Let $A = A[1], A[2], \ldots, A[n]$ and $B = B[1], B[2], \ldots, B[n]$ be arbitrary arrays containing $2n$ distinct positive integers. Array $A$ dominates array $B$ if it is possible to rearrange the arrays in a way such that $A[i] > B[i]$ for all $1 \leq i \leq n$.

(a) Show an example of two arrays $A$ and $B$ such that $A$ does not dominate $B$ and $B$ does not dominate $A$. Try to find an example with a very small $n$.

(b) Describe an efficient algorithm that decides if array $A$ dominates array $B$.

What is the worst-case number of comparisons performed by your algorithm?


For example in the array $[3, 8, 2, 4, \ldots]$, the pair $i = 1$ and $j = 3$ is an inversion because $A[1] = 3$ is greater than $A[3] = 2$. On the other hand, the pair $i = 1$ and $j = 2$ is not an inversion because $A[1] = 3$ is smaller than $A[2] = 8$. In this array there are 7 inversions and 3 non-inversions.

Describe an efficient algorithm that counts the number of inversions in any array.

What is the worst-case number of comparisons performed by your algorithm?
2.4 Problems with mixed complexity answers: $\Theta(n), \Theta(\log n)$ and/or $\Theta(n \log n)$


**Examples:** [1, 3, 2, 5, 13, 8, 34, 21] is almost sorted while [1, 3, 5, 2, 8, 21, 13, 34] is not because 2 is too far from its sorted position.

(a) Describe a linear time ($\Theta(n)$) algorithm that sorts almost sorted arrays.

(b) Describe a logarithmic time ($\Theta(\log(n))$) algorithm that finds if an integer $k$ appears in an almost sorted array $A$.

2. Let $A$ be an arbitrary array of $n$ positive integers. The goal is to find one of the numbers that appears **only once** in the array if exists.

For each of the following cases, describe an efficient algorithm that solves the above task and state the worst-case complexity (number of comparisons) of your algorithm.


(b) $A$ is an arbitrary array: the integers in the array may appear in any order.

(c) There are exactly 2 distinct integers in the entire array $A$ (but $A$ is not sorted).

(d) If a number $x$ appears in $A$ more than once, then all of its appearances are consecutive (but $A$ is not sorted).

(e) $A$ is a bitonic array. That is, there exists an index $i$ between 1 and $n$ such that:


3. Let $A$ be an arbitrary array of $n$ positive integers. The goal is to find one of the numbers that appears **more than once** in the array if exists.

For each of the following cases, describe an efficient algorithm that solves the above task and state the worst-case complexity (number of comparisons) of your algorithm.


(b) $A$ is an arbitrary array: the integers in the array may appear in any order.

(c) There are exactly 2 distinct integers in the entire array $A$ (but $A$ is not sorted).

(d) If a number $x$ appears in $A$ more than once, then all of its appearances are consecutive (but $A$ is not sorted).

(e) $A$ is a bitonic array. That is, there exists an index $i$ between 1 and $n$ such that:


(a) Describe an *efficient* algorithm that finds the *maximum* difference between any two integers in the array. In other words, compute $M = \max_{1 \leq i,j \leq n} \{ A[i] - A[j] \}$.

What is the worst-case number of comparisons performed by your algorithm?

(b) Describe an *efficient* algorithm that finds the *minimum positive* difference between any two integers in the array. In other words, compute $m = \min_{1 \leq i,j \leq n} \{|A[i] - A[j]|\}$.

What is the worst-case number of comparisons performed by your algorithm?
2.5 Complexity that Depends on Another Parameter


Describe an efficient algorithm that finds the $k$ ($1 \leq k \leq 2n$) smallest numbers among the $2n$ numbers of both arrays.

What is the worst-case number of comparisons made by your algorithm?

2. The range of an array $X$ is $[\min(X) .. \max(X)]$ where $\min(X)$ is the minimum number in the array and $\max(X)$ is the maximum number in the array. For two arrays $X$ and $Y$, define $X \leq Y$ if the range of $Y$ contains the range of $X$. That is, $\min(X) \geq \min(Y)$ and $\max(X) \leq \max(Y)$.

Let $A$ be an arbitrary array containing $n$ ($n \geq 2$) distinct integers and let $B$ be an arbitrary array containing $m \geq 2$ distinct integers. Design an efficient algorithm that decides if:

(i) either $A \leq B$, 
(ii) or $B \leq A$, 
(iii) or neither $A \leq B$ nor $B \leq A$.

What is the worst-case number of comparisons made by your algorithm as a function of $n$ and $k$?

3. Let $A = A[1] < \cdots < A[n]$ be a sorted array of $n \geq 1$ integers and $B = B[1] < \cdots < B[m]$ be a sorted array of $m \geq 1$ integers. Assume that all the $n + m$ integers are distinct.

Describe an efficient algorithm that finds the $k$-th largest number among the $n + m$ numbers for any possible value of $k$ between 1 and $n + m$.

What is the worst-case number of comparisons made by your algorithm?
3 Graph Problems

Remarks:

- Graphs usually have \( n \) vertices and \( m \) edges. The vertices are denoted by \( 1, 2, \ldots, n \). In some problems, all the degrees in the graph are equal and this degree is denoted by either \( d \) or \( \Delta \). Sometimes, this degree parameter stands for the maximum degree in the graph.

- The complexity of the algorithms should be expressed (using the \( O \)-notation) as a function of all the parameters. Usually only \( n \) and \( m \) but sometimes also \( d \) or \( \Delta \) or any other parameter.

- When the graph representation is not part of the problem, the analysis of the algorithm should specify the data structure used by the algorithm.

- A simple graph has no parallel edges and no self-loops.

3.1 Relatively Short Problems

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

2. Let \( G \) be a simple graph with \( n \) vertices and \( m \) edges. Assume that \( G \) is a \( d \)-regular graph (the degree of each vertex is exactly \( d \)). 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 efficient algorithm that always finds a simple path of length \( \text{at least } d+1 \) in \( G \). What is the complexity of your algorithm as a function of \( n \) and \( m \)? Explain why your algorithm never fails to find such a path.

3. Let \( G \) be a simple undirected graph with \( n \geq 4 \) vertices and \( m \) edges. A 4-cycle \((x, y, z, w)\) in \( G \) is an empty 4-cycle if the edges \((x, y), (y, z), (z, w), (w, x)\) exist but the edges \((x, z), (y, w)\) do not exist. Describe an efficient algorithm to determine if \( G \) has an empty 4-cycle. What is the complexity of your algorithm as a function of \( n \) and \( m \)?

4. 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. Describe an efficient algorithm that decides whether or not a graph has a triangle. What is the complexity of your algorithm?

5. Let \( G \) be a graph with \( n \) vertices and \( m \) edges. A leaf in \( G \) (not necessarily a tree) is a vertex that has exactly one neighbor. Describe an efficient algorithm that counts the number of leaves in \( G \). What is the complexity of your algorithm if the graph is represented by adjacency lists and what is the complexity of your algorithm if the graph is represented by an adjacency matrix?

6. Let \( G \) be a graph with \( n \) vertices and \( m \) edges. A dominating set \( D \) is a set of vertices such that any other vertex in \( G \) has at least one neighbor in \( D \). An independent set \( I \) is a set of vertices such that there is no edge between any two vertices from \( I \). Describe an algorithm that finds one of the independent and dominating sets in \( G \). That is, this set must be both a dominating set and an independent set. What is the complexity of your algorithm if the graph is represented by adjacency lists and what is the complexity of your algorithm if the graph is represented by an adjacency matrix?
3.2 More Involved Problems

1. Let \( V = \{1, 2, \ldots, n\} \) be a set of vertices. The union of two simple graphs \( G_1 = (V, E_1) \) and \( G_2 = (V, E_2) \) is a graph \( G = (V, E) \) whose edges contain all the edges from \( E_1 \) and \( E_2 \) \((E = E_1 \cup E_2)\).

   (a) Answers to the following five questions.
   
   i. Could the union of two different trees be a tree?
   
   ii. Could the union of two different paths be a cycle?
   
   iii. Is the union of two different bipartite graphs always a bipartite graph?
   
   iv. For \( n \geq 3 \), could the union of two different bipartite graphs be a complete graph?
   
   v. Is the union of two different regular graphs always a regular graph? In a regular graph, all the degrees are equal.

   (b) Describe an algorithm that constructs the union \( G \) of the graphs \( G_1 \) and \( G_2 \). What is the complexity of your algorithm if the three graphs are represented by adjacency lists and what is the complexity of your algorithm if the three graph are represented by adjacency matrices?

2. Let \( G \) be a simple, connected, undirected graph with \( n \) vertices and \( m \) edges. A sequence of vertices \( P = (v_1 - v_2 - \cdots - v_k) \) is a simple path in \( G \) of length \( k \) if all the vertices in the path \( P \) are distinct and if \((v_i - v_{i+1})\) is an edge for all \( 1 \leq i < k \). The following greedy algorithm tries to find a long path.

   1. Let \( v_1 \) be an arbitrary vertex
   2. \( k = 1 \)
   3. \( P = (v_1) \)
   4. While \( v_k \) has a neighbor \( u \) not in \( P \) do
      
      4.1. \( k = k + 1 \)
      4.2. \( v_k = u \)
      4.3. \( P = (v_1 - v_2 - \cdots - v_{k-1} - v_k) \)
   5. Return \((P = (v_1 - v_2 - \cdots - v_k))\)

   (a) What is the complexity of this algorithm if the graph is represented by adjacency lists and what is its complexity if the graph is represented by an adjacency matrix.

   (b) Assume that \( G \) is a \( d \)-regular graph (each vertex has exactly \( d \) neighbors). What could you say about the length of the output-path of this greedy algorithm? Find a function \( f(d) \) such that \( k > f(d) \) for any \( d \)-regular graph.

3. Let \( G \) be a simple graph with \( n \) vertices and \( m \) edges. Assume that \( G \) is a \( d \)-regular graph (the degree of each vertex is exactly \( d \)). An edge coloring of \( G \) is an assignment of integers (colors) to all the edges of \( G \) such that two adjacent edges must be assigned different colors. The goal is to minimize the number of colors that are required to color all the edges of \( G \).

   (a) Describe a coloring algorithm that colors all the edges of \( G \) with as few as possible colors.

   \textbf{Hint:} A trivial algorithm assigns each edge a different color. Think of a greedy strategy to color the \( m \) edges that is performing better.

   (b) What is the complexity of your algorithm as a function of \( n, m, \) and \( d \)?

   (c) Give an upper bound on the number of colors used by your algorithm as a function of \( n, m, \) and \( d \).

   \textbf{Hint:} \( m \) is the exact number of colors used by the trivial algorithm that assigns a different color to each edge. You are asked to design an algorithm with a smaller bound.
4. Let \( G \) be a simple (no parallel edges and no loops) connected graph with \( n \) vertices and \( m \) edges. The following table contains 10 definitions for classes of graphs. Which of them are always trees? Which of them are never trees? Which of them are sometimes trees and sometimes not trees?

<table>
<thead>
<tr>
<th>Definition</th>
<th>Always</th>
<th>Never</th>
<th>Sometimes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m = n - 1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bipartite Graphs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycles</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acyclic Graphs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( m = n )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Planar Graphs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Paths</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n = 2 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n = 3 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complete Graphs for ( n &gt; 2 )</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) Let \( G \) be a simple undirected connected Graph. Assume that the DFS tree and the BFS tree are **identical** when the search starts with some vertex \( v \). What must be the structure of \( G \)?

(b) Let \( G \) be a simple undirected connected Graph. Apply the following process on \( G \):

- As long as \( G \) contains a vertex (or vertices) with degree 1 (vertices that have exactly one neighbor), **remove** one of these vertices and the edge that contains it from \( G \).

For which graphs this process terminates with a final graph that has exactly one vertex?

5. Let \( G \) be a simple graph with \( n \) vertices and \( m \) edges. Denote the vertices by \( 1, 2, \ldots, n \) and denote the edges by \( e_1, \ldots, e_m \). Let \( d(v) \) be the degree of a vertex \( v \).

- A vertex \( v \) is an **isolated vertex** if it has no neighbors: \( d(v) = 0 \).
- A vertex \( v \) is a **star vertex** if all the other vertices are its neighbors: \( d(v) = n - 1 \).
- An edge \((v, u)\) is an **isolated edge** if it does not intersect another edge: \( d(v) = d(u) = 1 \).

(a) The following algorithm finds an isolated vertex if exists:

\[
\text{for } v = 1 \text{ to } n \text{ if } d(v) = 0 \text{ return } v.
\]

(b) The following algorithm finds a star vertex if exists:

\[
\text{for } v = 1 \text{ to } n \text{ if } d(v) = n - 1 \text{ return } v
\]

(c) The following algorithm finds an isolated edge if exists:

\[
\text{for } i = 1 \text{ to } m \text{ let } e_i = (v, u) \text{ if } d(v) = 1 \text{ and } d(u) = 1 \text{ then return } (v, u)
\]

For all three algorithms answer the following two questions:

- What is the complexity of Algorithm A if the graph is represented by adjacency lists?
- What is the complexity of Algorithm A if the graph is represented by an adjacency matrix?

**Remark:** The degrees of the vertices are not part of the input. The graph is represented either by \( n \) lists or by an \( n \times n \) matrix.
6. The input of the following four greedy algorithms is a simple undirected graph $G = (V, E)$. All algorithms partition the graph's vertex set $V$ into two disjoint sets $S$ and $R$ ($V = S \cup R$). For each algorithm, write what you know about the structure of $S$ and $R$ and the relations between the vertices of $S$ and the vertices of $R$ when the algorithm terminates.

(a) Algorithm $A$:
- Initially: both $S = \emptyset$ and $R = \emptyset$ are empty;
- As long as $G$ has vertices, do the following:
  - Let $v$ be an arbitrary vertex in $G$;
  - Add $v$ to $S$ and add all of the neighbors of $v$ in $G$ to $R$;
  - Remove $v$ and all of its neighbors from $G$;

(b) Algorithm $B$:
- Initially: both $S = \emptyset$ and $R = \emptyset$ are empty;
- As long as $G$ has vertices, do the following:
  - Let $v$ be an arbitrary vertex in $V$;
  - If $v$ is connected to all the vertices in $S$ then add $v$ to $S$ otherwise add $v$ to $R$;
  - Remove $v$ from $G$;

(c) Algorithm $C$:
- Initially: $S = \emptyset$ is empty and $R = V$ contains all the vertices;
- As long as $G$ has edges, do the following:
  - Let $(u, v)$ be an arbitrary edge in $G$;
  - Add $v$ to $S$ and remove $v$ from $R$;
  - Remove all the edges that contain $v$ from $G$.

(d) Algorithm $D$:
- Initially: $S = \emptyset$ is empty and $R = V$ contains all the vertices;
- As long as $G$ has edges, do the following:
  - Let $(u, v)$ be an arbitrary edge in $G$;
  - Add both $u$ and $v$ to $S$ and remove both $u$ and $v$ from $R$;
  - Remove $u$ and $v$ and all of their adjacent edges from $G$.

7. Let $G$ be a simple undirected graph with the $n$ vertices and $m$ edges. Define the graph $H$ 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 $G$.

(a) Which is $H$ if $G$ is the complete graph? Explain.
(b) Which is $H$ if $G$ is the null graph? Explain.
(c) Which is $H$ if $G$ is the complete bipartite graph? Explain.
(d) Is $H$ a tree if $G$ is a tree? Explain.
(e) 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.
(f) 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.
8. **Story:** A celebrity among a group of \( n \geq 1 \) people is a person who knows nobody but is known by everybody else. The task is to identify a celebrity by only asking people questions of the form: “Do you know him/her?”

(a) Can a group have more than one celebrity? Justify your answer.

(b) Can a group have no celebrity? Justify your answer.

(c) Model the story with a directed graph that is represented with an adjacency matrix. In particular, answer the following questions:
   i. Who are the vertices of this graph?
   ii. What is the meaning of a directed edge in this graph?
   iii. What is a celebrity?
   iv. What is the equivalent in graph terms to the defined question?
   v. What is the complexity objective?

(d) 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.
3.3 Involved Problems

1. A legal coloring of a simple and undirected graph is an assignment of colors (positive integers) to the vertices in a way that any two adjacent vertices (neighbors) must be colored with different colors. The following is a greedy algorithm that colors the vertices of an input graph with the colors: 1, 2, 3, .... Assume that the graph has \( n \) vertices denoted by: \( v_1, v_2, \ldots, v_n \).

   ```plaintext
   for \( i = 1 \) to \( n \) do color\((v_i)\) = 0
   for \( i = 1 \) to \( n \) do
       \( c = 1 \)
       while (\( v_i \) has a neighbor \( v_j \) such that color\((v_j)\) = \( c \)) do \( c = c + 1 \)
       color\((v_i)\) = \( c \)
   end-for
   ```

Note that the algorithm does not specify how to implement the condition of the `while` loop. An implementation may include additional data structures beyond the regular graph representation (adjacency lists or an adjacency matrix).

(a) Find an assignment of the letters \( A, B, \ldots, J \) to the vertices of the Petersen graph for which the greedy algorithm that orders the vertices in alphabetical order would use 4 colors. Near each vertex, write its letter and color.

(b) Let \( G \) be a \( \Delta \)-regular graph. That is, any vertex has exactly \( \Delta \) neighbors. Prove, that the greedy algorithm needs only the colors 1, \ldots, \( \Delta + 1 \). Prove this for any value of \( n \) and \( 0 \leq \Delta < n \).

(c) Let \( L(G) \) be the line graph of \( G \). The vertices of \( L(G) \) represent the edges of \( G \). Two vertices in \( L(G) \) are connected by an edge if the corresponding edges in \( G \) share a vertex. Assume that \( G \) is a \( \Delta \)-regular graph. Prove, that the greedy algorithm, applied on the graph \( L(G) \), needs only the colors 1, \ldots, \( 2\Delta - 1 \). Prove this for any value of \( n \) and \( 0 \leq \Delta < n \).

**Hint:** What is the maximum degree in \( L(G) \) (prove your claim)?

(d) A bipartite graph can be colored with 2 colors. Construct a small bipartite graph and order its vertices such that the greedy algorithm that follows this order would use more than 2 colors.

(e) In the previous part, your example should be a graph with a small number of vertices. In this part, you are asked to scale your solution for infinite values of \( n \). Construct a family of bipartite graphs and order the vertices of each graph in the family such that the greedy algorithm that follows this order would use \( \Omega(n) \) colors. If possible illustrate these graphs and show the order among the vertices. Use the picture to prove the \( \Omega(n) \) claim.
2. Let $G = (V, E)$ be a simple and undirected graph with $n$ vertices and $m$ edges.

A matching in $G$ is a set of edges such that no two edges in the set share the same vertex: if $M \subset E$ is a matching then for any two edges $(u_1, v_1)$ and $(u_2, v_2)$ in $M$, all 4 vertices $u_1, u_2, v_1, v_2$ are distinct.

A maximum matching in $G$ is a matching that has the largest number of edges among all possible matchings.

A perfect matching in $G$ is a matching that covers all the vertices: for any vertex $v$, there exists an edge $(v, u)$ in the matching.

A maximal matching in $G$ is a matching that is not contained in another matching: any new edge that would be added to the set would create a non-matching set.

(a) Which of the following statements are true? Write True or False

i. Every maximal matching is a maximum matching.

ii. Every maximum matching is a perfect matching.

iii. Every perfect matching is a maximal matching.

iv. Every maximal matching is a perfect matching.

v. Every perfect matching is a maximum matching.

vi. Every maximum matching is a maximal matching.

vii. A graph with odd number of vertices does not have a perfect matching.

viii. A graph with an even number of vertices always has a perfect matching.

ix. The size of a maximum matching in a graph is at most $\lfloor n/2 \rfloor$.

x. The size of a maximum matching in a connected graph is at least $\lfloor n/2 \rfloor$.

(b) Find the size of the maximum matching for each of the following family of graphs. Assume that $n$ is large enough and note that $n$ can be either odd or even. Justify your answer with few sentences.

i. Null graphs (graphs with no edges).

ii. Complete graphs (all edges exist).

iii. Path graphs.

iv. Cycle graphs.

v. Star graphs.

(c) Illustrate a small graph that has a maximal matching that is not a maximum matching. Label the vertices & edges and list the edges in the maximal matching and the maximum matching.

(d) The following greedy algorithm finds a maximal matching in a graph:

- Mark all edges as CANDIDATE.
- While a CANDIDATE edge exists do the following:
  - Select one of the CANDIDATE edges and mark it as MATCH.
  - Mark all the CANDIDATE edges that intersect this edge as NON-MATCH.
- Return the set of all edges that are marked as MATCH.

i. What is the complexity of this algorithm if the graph is represented with adjacency lists. Justify your answer.

ii. What is the complexity of this algorithm if the graph is represented with an adjacency matrix. Justify your answer.
3. Let $G$ be a simple and undirected graph with $n$ vertices and $m$ edges.

- 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.

(a) For each one of the following 6 families of graphs, find the exact size of a maximum independent set. Justify your answer with 2-3 sentences (with words and/or illustrations).

i. $G$ is the null graph.
ii. $G$ is the complete graph.
iii. $G$ is the star graph.
iv. $G$ is the path graph.
v. $G$ is the cycle graph.
vi. $G$ is a complete bipartite graph that has $x$ vertices in one side and $y$ vertices in the other side for $x + y = n$.

(b) A maximal independent set is not always a maximum independent set. Demonstrate this by finding the only two different maximal independent sets in the star graph. Assume that $v_1$ is the center of the star and that it is connected to $v_2, v_3, \ldots, v_n$.

(c) Finding a maximum independent set is an NP-Complete problem whereas finding a maximal independent set is not.

The following greedy algorithm finds a maximal independent set in a graph:

- Mark all vertices as CANDIDATE.
- As long as there exists a CANDIDATE vertices do the following:
  - Select one of the CANDIDATE vertices and mark it as INDEPENDENT.
  - Mark all the CANDIDATE neighbors of this vertex as NON-INDEPENDENT.
- Return the set of all vertices that are marked INDEPENDENT.

i. What is the complexity of your algorithm if the graph is represented by adjacency lists? What is the complexity of your algorithm if the graph is represented by an adjacency matrix? Explain.

ii. Assume that the maximum degree in the graph is $d$ for some $1 \leq d < n$. Find a lower bound on the size of the maximal independent set found by your algorithm. Explain your answer.

(d) Finding maximum independent set in tree graphs.

i. Describe an efficient algorithm to find a maximum independent set in a tree graph. Assume that the tree is rooted at the node $r$, that each internal node in the tree has pointers to all of its children, and that a leaf node has no pointers.

ii. What is the worst-case running time of your algorithm as a function of $n$? Explain. **Hint:** Show how to solve the problem for the tree rooted at $r$ with the solutions to some of the subtrees of this tree.
4. Let $G$ be a simple and undirected graph with $n$ vertices and $m$ edges. Study the following definitions and algorithm and then answer the multiple-choice questions.

- 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.

The following greedy algorithm finds a maximal independent set in a graph:

- Mark all vertices as CANDIDATE.
- While a CANDIDATE vertex exists do the following:
  - Select one of the CANDIDATE vertices and mark it as INDEPENDENT.
  - Mark all the CANDIDATE neighbors of this vertex as NON-INDEPENDENT.
- Return the set of all vertices that are marked as INDEPENDENT.

(a) What is the exact size of a maximum independent set in the null graph ($n \geq 1$ vertices and $m = 0$ edges)?
   i. 0.
   ii. 1.
   iii. $n$.
   iv. $m$.
   v. $\lceil n/2 \rceil$.

(b) What is the exact size of a maximum independent set in the complete graph (all possible edges exist)?
   i. 0.
   ii. 1.
   iii. $n$.
   iv. $m$.
   v. $\lceil n/2 \rceil$.

(c) What is the exact size of a maximum independent set in the star graph?
   i. 1.
   ii. 2.
   iii. $\lceil n/2 \rceil$.
   iv. $n - 1$.
   v. $n$.

(d) What is the exact size of a maximum independent set in the path graph?
   i. $n$.
   ii. $\lceil n/2 \rceil$.
   iii. $\lceil n/2 \rceil$.
   iv. $\lceil n/3 \rceil$.
   v. 1.
(e) What is the exact size of a maximum independent set in the cycle graph?
   i. $n$.
   ii. $\lfloor n/2 \rfloor$.
   iii. $\lceil n/2 \rceil$.
   iv. $\lfloor n/3 \rfloor$.
   v. 1.

(f) What is the exact size of a maximum independent set in the complete bipartite graph that has $x$ vertices in one side and $y$ vertices in the other side for $x + y = n$?
   i. $\min\{x, y\}$.
   ii. $\max\{x, y\}$.
   iii. $x + y$.
   iv. $\max\{x, y\} - \min\{x, y\}$.
   v. $\lfloor n/2 \rfloor$.

(g) Which of the following statements is true?
   i. A maximal independent set is always a maximum independent set.
   ii. A maximum independent set is always a maximal independent set.
   iii. The first two statements are true.
   iv. The first two statements are false.

(h) The complexity of the greedy algorithm if the graph is represented by adjacency lists is
   i. $O(n^2)$.
   ii. $O(n \log n)$.
   iii. $O(n + m)$.
   iv. $O(n)$.

(i) The complexity of the greedy algorithm if the graph is represented by an adjacency matrix is
   i. $O(n^2)$.
   ii. $O(n \log n)$.
   iii. $O(n + m)$.
   iv. $O(n)$.

(j) Assume that the maximum degree in the graph is $d$ for some $1 \leq d < n$. The size of the independent set found by the greedy algorithm is always greater or equal to $x$ but sometimes could be $x$.
   i. $x = 1$.
   ii. $x = \lfloor n/(d + 1) \rfloor$.
   iii. $x = \lceil n/d \rceil$.
   iv. $x = \lfloor n/2 \rfloor$. 