Ben-Gurion University of the Negev

Data Structures

202.1.1031

Assignment No. 1

Responsible staff members:
Prof. Paz Carmi (carmip@bgu.ac.il)
Jules Zisser (zisserh@post.bgu.ac.il) Ilan Naiman (naimani@post.bgu.ac.il)

Authors:
John Doe (123456789) Jane Doe (987654321)

Publication date: March 30th, 2025
Submission date: April 20th, 2025

Update date: March 17th, 2025

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

1 Submission Guidelines

- The submission is in pairs or individuals. We recommend working in pairs to encourage discussion and mutual inspiration. For the avoidance of doubt, if the assignment was submitted in pairs, all of the submitted answers should be the outcome of a joint work.
- Whether you submit as a pair or an individual, you must choose a group in the group assignment resource in Moodle. Groups created for individuals are of the form Assignment1_s_20, and groups for pairs are of the form Assignment1_p_100.
- Your submission should be a single pdf file (composed by LaTeX or any other word processor such as MS Word), and its name should be your group name (as chosen in Moodle), i.e., for the pair 17, the file name should be `Assignment1_p_17.pdf`.
- Questions regarding the assignment should be asked in the dedicated forum in Moodle or during the office hours of the responsible staff members. The forum is intended to allow students to discuss the assignment. The crew will answer questions in the forum when clarification is required in the pinned FAQ thread.
- $\log()$ in the course is in base 2, i.e., $\log_2()$ unless stated otherwise.
- A submission that does not include the integrity statement **will receive a score of 0**.

Before starting your work, read the following instructions carefully:

- With each version of your work, submit it so that with each submission a larger part of your work is complete. This way, you won't lose major parts of your work due to a technical issue or unexpected circumstances.
- DO NOT wait until the final hour for submitting your work, because there might be a power stoppage/-computer issues/internet issues/Moodle issues/etc.
- save a continuous backup of your work in some cloud platform, so that you can access it from any device if your computer decides to malfunction.
- In case you work in pairs, both partners must verify that the work was submitted timely and fully. This way, you will prevent uncomfortable situations in cases of technical issues.

2 Integrity Statement

Each of the students submitting this assignment must sign the following integrity statement:

We, ***Dolev Rubin***, ID number ***319127379***, and ***Hila Zuckerman***, ID number ***212735450*** assert that the work we submitted is entirely our own. We have not received any part from any other student in the class, nor did we give parts of it for others to use. We realize that if my work is found to contain code that is not originally our own, a formal case will be opened against us with the BGU disciplinary committee.

3 Complexity Hierarchy

Question 1: (20 points) For each of the functions below place f_i in the table in the correct place according to the following requirements:

- Functions f_i and f_j will be located on the same line if and only if $f_i(n) \in \Theta(f_j(n))$
- For each function f_i in line k and f_j in line $k+1$, $f_i(n) \in O(f_j(n))$

Your task is:

- For each of two functions f_i, f_j you have placed in the same line in the table show constants c_1, c_2 and n_0 suitable for the definition of $f_i(n) \in \Theta(f_j(n))$.
- For each two functions f_i you have placed in line k in the table and f_j you have placed in line $k+1$ show constants c, n_0 suitable for the definition of $f_i(n) \in O(f_j(n))$.

For example, for the following three functions:

1. $f_1(n) = n^2 - 10$
2. $f_2(n) = 6$
3. $f_3(n) = 22n^2 + 10n$

The table will look:

line number	The functions
1	f_2
2	f_1, f_3

- $f_2 \in \ O(f_1):\ 6 \leq c(n^2 - 10)$ for $n \geq n_0 = 4$ and $c = 2$.
- $f_1 \in \ \Theta(f_3):$
  - $f_1 \in \ O(f_3):\ n^2 - 10 \leq c(n^2 - 10)$ for $n \geq n_0 = 1$ and $c = 1$.
  - $f_1 \in \ \Omega(f_3):\ n^2 - 10 \geq c(n^2 - 10)$ for $n \geq n_0 = 6$ and $c = \frac{1}{32}$.

Notice, since there are two functions in Line 2, it is enough to compare f_2 to only one of them.

1. $f_1(n) = 1$
2. $f_2(n) = 1034$
3. $f_3(n) = n^2$
4. $f_4(n) = n^n$

5. $f_5(n) = 2^{n+\log n-10}$
6. $f_6(n) = 2^{(2^n)}$
7. $f_7(n) = 2^n$
8. $f_8(n) = \log(4^n * n^8)$
9. $f_9(n) = \log(n^{\frac{1}{3}})$
10. $f_{10}(n) = \frac{6n^2}{8} - 10$

Answer 1: Fill in the table below (**The Proof elaboration in this question is according to the example given above**):

line number	The functions
1	f_1, f_2
2	f_9
3	f_8
4	f_3, f_10
5	f_7
6	f_5
7	f_4
8	f_6
9	
10	

- $f_1 \in \Theta(f_2):$
  - $f_1 \in O(f_2): 1 \leq c(1034)$ for $n \geq n_0 = 1$ and $c = 1$.
  - $f_1 \in \Omega(f_2): 1 \geq c(1034)$ for $n \geq n_0 = 1$ and $c = \frac{1}{10000}$.
- $f_1 \in O(f_9): 1 \leq c\log(n^{\frac{1}{3}}) = \frac{c}{3}\log(n)$ for $n \geq n_0 = 8$ and $c = 1$.
- $f_9 \in O(f_8): \log(n^{\frac{1}{3}}) \leq c \cdot \log(4^n \cdot n^8) = c \cdot n\log(4) + 8\log(n) = c \cdot (2n + 8\log(n))$ for $n \geq n_0 = 2$ and $c = 1$.
- $f_8 \in O(f_3): \log(4^n \cdot n^8) = 2n + 8\log(n) \leq cn^2$ for $n \geq n_0 = 8$ and $c = 1$.
- $f_{10} \in \Theta(f_3):$
  - $f_{10} \in O(f_3): \frac{6n^2}{8} - 10 \leq cn^2$ for $n \geq n_0 = 1$ and $c = 1$.
  - $f_{10} \in \Omega(f_3): \frac{6n^2}{8} - 10 \geq cn^2$ for $n \geq n_0 = 10$ and $c = \frac{1}{8}$.
- $f_3 \in O(f_7): n^2 \leq c \cdot 2^n$ for $n \geq n_0 = 5$ and $c = 1$.
- $f_7 \in O(f_5): 2^n \leq c \cdot 2^{n+\log(n)-10} = c \cdot 2^{n-10} \cdot 2^{log(n)} = n \cdot 2^{n-10}$ for $n \geq n_0 = 1$ and $c = 2^{11}$.
- $f_5 \in O(f_4): 2^{n+\log(n)-10} \leq c \cdot n^n$ for $n \geq n_0 = 10$ and $c = 1$.
- $f_4 \in O(f_6): n^n \leq c \cdot 2^{(2^n)}$ for $n \geq n_0 = 5$ and $c = 3$.

4 Properties of asymptotic bounds

Question 2: (25 points) Let $f(n)$ and $g(n)$ be asymptotically positive functions.

Definition: We say that a function $f : \mathbb{N} \to \mathbb{N}$ is **unbounded increasing monotone function** if:

1. for all x, y:
$$x < y \iff f(x) < f(y) \qquad (\iff \text{ stands for if and only if})$$
2. for all constant $M \in \mathbb{R}^+$ there is $n_0 \in \mathbb{N}$ such that for all $n > n_0$:
$$f(n) > M$$

The logarithm function is an **unbounded increasing monotone function** defined for $x > 0$.

In conjectures 6 and 7 you can assume that for all constant $c > 0$, there is $x_0 > 0$ such that for all $x > x_0$:
$$c < \frac{x}{\log x}$$

Prove each of the following conjectures:

1. (4 points) $n > \log n$ for all $n \in \mathbb{N}$
2. (2 points) Reflexivity: $f(n) \in \Theta(f(n))$
3. (4 points) Transitivity: If $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$ then $f(n) \in \Theta(h(n))$
4. (3 points) Transpose Symmetry: $f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n))$
5. (4 points) For all $f(n) > 1$ and $g(n) > 1$ two **unbounded increasing monotone functions** such that $f(n) \in \Omega(g(n))$, imply
$$\log(f(n)) \in \Omega(\log(g(n)))$$
6. (4 points) For all constant $k \geq 1$ and for all constant $1 > \varepsilon > 0$:
$$\log^k(n) \in O(n^\varepsilon)$$
7. (4 points) For all constant $k \in \mathbb{N}$ and for all constant $\varepsilon > 0$:
$$n^k \in O((1+\varepsilon)^n)$$

Answer 2:

1. $n > \log n$ for all $n \in N$

We know that $n < 2^n$ for all $n \in N$, then: $\log(n) < log(2^n) = n \cdot \log(2) = n$

2. Reflexivity: $f(n) \in \Theta(f(n))$

By the definition of Θ-notation, $f(n) \in \Theta(f(n))$ if there exist constants $c_1, c_2 > 0$ and n_0 such that:
$$c_1 f(n) \leq f(n) \leq c_2 f(n) \quad \forall n \geq n_0.$$

By choosing $c_1 = 1$ and $c_2 = 1$, the inequality holds in all cases, proving reflexivity.

3. Transitivity: If $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$, then $f(n) \in \Theta(h(n))$

By the definition of Θ:

$$c_1 g(n) \leq f(n) \leq c_2 g(n),$$

$$d_1 h(n) \leq g(n) \leq d_2 h(n).$$

Now we will multiply the equation by the relevant constant on each side:

$$(c_1 d_1) h(n) \leq f(n) \leq (c_2 d_2) h(n).$$

Since $c_1 d_1$ and $c_2 d_2$ are positive constants, $f(n) \in \Theta(h(n))$.

4. Transpose Symmetry: $f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n))$

By the definition of Big-O:

$$f(n) \leq c g(n) \quad \text{for a constant } c > 0 \text{ and sufficiently large } n.$$

Rewriting:

$$g(n) \geq \frac{1}{c} f(n),$$

which is the definition of $g(n) \in \Omega(f(n))$, proving the transpose symmetry.

5. If $f(n) \in \Omega(g(n))$ for increasing monotone functions, then $\log f(n) \in \Omega(\log g(n))$

Since $f(n) \in \Omega(g(n))$, there exist constants $c > 0$ and n_0 such that:

$$f(n) \geq c g(n), \quad \forall n \geq n_0.$$

Taking logarithms:

$$\log f(n) \geq \log(c g(n)) = \log c + \log g(n).$$

Since $\log c$ is a constant, we have:

$$\log f(n) \in \Omega(\log g(n)).$$

6. $\log^k(n) \in O(n^\epsilon)$ for constants $k \geq 1$ and $1 > \epsilon > 0$

We need to prove that:

$$\log^k(n) \leq c \cdot n^\epsilon$$

for constants $c, n > n_0$. We will log both sides:

$$\log(\log^k(n)) \leq log(c \cdot n^\epsilon) = \log(c) + \epsilon \cdot log(n)$$

We choose $c = 1$, and we will now show that there exists n_0 such that for every $n > n_0$ the inequality holds.

$$k \cdot \log(\log((n)) \leq \epsilon \cdot \log(n) + 0$$
$$\tfrac{k}{\epsilon} \leq \tfrac{\log(n)}{\log(\log(n))}$$

We mark $x = \frac{k}{\epsilon}$ and $n_0 = \lceil 2^{2^x} \rceil$:

$$x \leq \tfrac{log(2^{2^x})}{\log(\log(2^{2^x}))} = \tfrac{2^x}{x}$$

According to what we know about ϵ and k, $1 \leq x$ so the inequality holds.

7. $n^k \in O((1+\epsilon)^n)$ for constants k and $\epsilon > 0$

We need to prove that:

$$n^k \leq c \cdot (1+\epsilon)^n$$

for constants $c, n > n_0$. We will log both sides and choose $c = 1$:

$$\log(n^k) = k \cdot \log(n) \leq \log((1+\epsilon)^n) = n \cdot log(1+\epsilon)$$

We learned in class that $\log(n) \in O(n)$, so this inequality holds for certain n_0.

5 Recurrence Examples

Question 3: (15 points) Find and prove an asymptotic tight bound Θ for $T(n)$ in each of the following recurrences. Assume that $T(c) = c$ for $c \leq 4$.

1. $T(n) = 4T(\frac{n}{2}) + n$
2. $T(n) = T(\frac{3}{7}n) + 1$
3. $T(n) = 6T(\frac{n}{2}) + n^4 \log n$
4. $T(n) = 8T(\frac{n}{2}) + n^4 + 3n^3 \log n$
5. $T(n) = T(\sqrt{n}) + n$

Answer 3:

1. $T(n) = 4T(n/2) + n$

Using the Master Theorem, we compare with the recurrence form:

$$T(n) = aT(n/b) + f(n),$$

where $a = 4$, $b = 2$, and $f(n) = n$.

We check the condition $f(n) = O(n^c)$ where $c = \log_b a = \log_2 4 = 2$. Since $f(n) = O(n^1)$ and $n^1 = O(n^2)$, Case 1 of the Master Theorem applies:

$$T(n) = \Theta(n^2).$$

2. $T(n) = T(3n/7) + 1$

This recurrence can be solved using the iterative method:

$$\begin{aligned} T(n) &= T(3n/7) + 1 \\ &= T(9n/49) + 1 + 1 \\ &= \dots \\ &= T(3^k n/7^k) + k. \end{aligned}$$

Stopping when $n = 1$, we find $k = \log_{7/3} n$, leading to:

$$T(n) = \Theta(\log n).$$

3. $T(n) = 6T(n/2) + n^4 \log n$

Here, $a = 6$, $b = 2$, and $f(n) = n^4 \log n$.

$$c = \log_2 6 \approx 2.585.$$

Since $f(n) = n^4 \log n = \Omega(n^{4-\epsilon})$ for small ϵ, we use the Master Theorem's Case 3 with the regularity condition, leading to:

$$T(n) = \Theta(n^4 \log n).$$

4. $T(n) = 8T(n/2) + n^4 + 3n^3 \log n$

Here, $a = 8$, $b = 2$, and $f(n) = n^4 + 3n^3 \log n$.

$$c = \log_2 8 = 3.$$

Since $f(n) = n^4 + 3n^3 \log n = \Omega(n^{4-\epsilon})$ for small ϵ, the dominating term is n^4. By the Master Theorem, Case 3 applies:

$$T(n) = \Theta(n^4).$$

5. $T(n) = T(\sqrt{n}) + n$

Using the recursion tree method:

$$\begin{aligned} T(n) &= T(n^{1/2}) + n \\ &= T(n^{1/4}) + n^{1/2} + n \\ &= \dots \\ &= T(1) + n + n^{1/2} + n^{1/4} + \dots . \end{aligned}$$

The sum approximates to $O(n)$, so we conclude:

$$T(n) = \Theta(n).$$

6 Time Complexity

Question 4: (10 points)

Find the time complexity of the following algorithms in terms of Θ (you are not required to prove this asymptotic bound). You are expected to analyze the asymptotic tight bound of each line in the algorithms, as presented in class for the analysis of insertion sort. In addition, show the calculation of $T(n)$ as done in the lecture.

Algorithm 1 Search $(array[int], key)$

```
1: return Foo Search($array, key, 0, array.length - 1$)
```

Algorithm 2 Foo Search (array[int], key, left, right)

```
1: $output \leftarrow -1$    // default value for not found
2: $i \leftarrow left - 1$
3: if left $\leq$ right then
4:    for $j = left$ to $right - 1$ do
5:       if $array[j] \leq array[right]$ then
6:          $i \leftarrow i + 1$
7:          exchange $array[i]$ with $array[j]$
8:       end if
9:    end for
10:   exchange $array[i+1]$ with $array[right]$
11:   if $array[i+1] = key$ then
12:      $output \leftarrow i + 1$
13:   else if $array[i+1] \leq key$ then
14:      $output \leftarrow$ Foo Search($array, key, left, i$)
15:   else
16:      $output \leftarrow$ Foo Search($array, key, i + 2, right$)
17:   end if
18: end if
19: return output
```

Answer 4: Fill in the table and calculate the sum $T(n)$ below (do not fill in the grayed-out entries):

Line number	Number of times the line is performed	Cost of a single repetition	Total cost of this line
1	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
2	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
3	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
4	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
5	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
6	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
7	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$
8	▨	▨	▨
9	▨	▨	▨
10	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
11	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
12	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
13	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
14	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
15	▨	▨	▨
16	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
17	▨	▨	▨
18	▨	▨	▨
19	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Total	▨	▨	$\Theta(n)$

In the worst case, the "right" element is always the smallest or largest element in the array, leading to a seperation of the array to 2 sub arrays of size 1, and $n-1$. Which leads to the following time complexity formula:

$$T(n) = T(n-1) + n$$

Proof by induction that $T(n) \in \theta(n^2)$:

induction base: $n = 1$:

$0 \leq c_1 \cdot 1^2 \leq T(1) = 1 \leq c_2 \cdot 1^2$ so $c_1 = 1, c_2 = 1$

Induction step

$T(n) = T(n-1) - n = T(n-2) + n - 1 + n = T(n-3) + n - 2 + 2n - 1 = ... = T(n-i) + i \cdot n - \sum_{k=1}^{i-1} k = T(n-i) + i \cdot n - \frac{i \cdot (i-1)}{2}$

We want to find the i that will bring us to T(1) - the stopping condition: $i = n - 1$.

$T(n) = T(1) + n^2 - n - \frac{n^2}{2} + n + \frac{n}{2} - 1 = T(1) + \frac{n^2}{2} + \frac{n}{2} - 1 = \theta(1) + \theta(n^2) + \theta(n) = \theta(n^2)$

7 Algorithm Development

Question 5: (10 points) Describe in pseudo-code an efficient algorithm for solving the following problem, and analyze its running time and additional memory usage. The running time should be as asymptotically low as possible.
Notice: In this question you can assume that there is a **fast initializing array** that allows initialization of n cells in a constant time, that is, in $\Theta(1)$. This stands in contradiction to the RAM model displayed in class (in which allocation of n cells is done in $\Theta(1)$ and initialization of all cells is $\Theta(n)$), and can be assumed in questions 5 and 6 only.

Input: Two unsorted arrays A and B of size $N \in \mathbb{N}$. The value of each element in the arrays is a natural number in the range $[0, 2N]$.

Output: The number of elements in Array B that appear in Array A.

Examples:

Input:

A	5	2	14	2	2	7	1
B	6	9	9	10	13	0	8

Output: 0

Input:

A	5	9	7	2	2	7	7
B	6	9	9	10	7	13	8

Output: 3 (*9 appears in A and appears twice in B, 7 appears in A and appears once in B*)

Input:

A	6	9	9	10	7	13	8
B	5	9	7	2	2	7	7

Output: 4 (*9 appears in A and appears once in B, 7 appears in A and appears three times in B*)

Answer 5: Pseudocode of the algorithm:

```
def question_5(A, B):
    """
    This pseudocode uses an additional array to indicate in O(1) time
    whether a number is in array A. the algorithm then loops throw the
    numbers in array B in order to perform the counting.
    """
    mem_array = new Array[2N] # filled with zeros, takes O(1) time and O(N) space.
    for number in A: # runs N times, so this for loop is O(N) overall.
        mem_array[number] = 1 # done in O(1)

    counter = 0
    for num in B: # runs N times, so this for loop is O(N) overall.
        if mem_array[num] == 1: # done in O(1)
            counter = counter + 1

    return counter # overall, the function runs in O(N) + O(N) = O(n)
```

Question 6: (5 points) Solve question 5, where the value of each element in the arrays is a natural number in the range $[0, N^2]$ (instead of $[0, 2N]$).

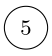
Answer 6: Pseudocode of the algorithm:

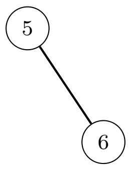
```
def question_6(A, B):
    """
    This pseudocode is sorting array A usgin merge sort, and then performs
    binary search on every element of array B to check if it exists in A.
    """
    # sort using merge sort, O(n*log(n)) time complexity and O(n) memory complexity
    sortedA = sort(A)

    counter = 0
    # runs n times, so this for loop is O(n*log(n)) time complexity overall.
    for number in B:
        # done in O(log(n)) time complexity
        if binary_search(sortedA, number) == true:
            counter = counter + 1

    # overall, this function runs in O(n*log(n)) time complexity and O(n) memory complexity
    return counter
```

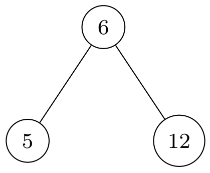
8 AVL tree

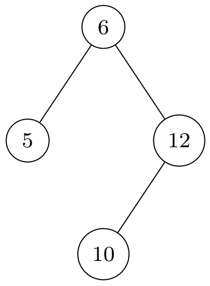
Question 7: (5 points) The following sequence of keys has been added to an AVL tree. $< 5,\ 6,\ 12,\ 10, 8 >$.

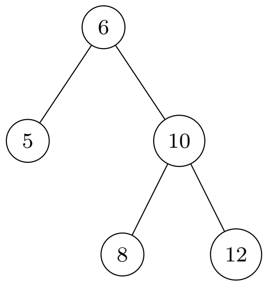
Answer 7:











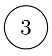
Question 8: (10 points) Consider the following set S of keys: $\{1, 2, 3, 4, 5, 6, 7\}$

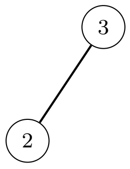
Answer 8: Write an insertion sequence of the keys in S that has **at least two rotations overall** and results in an AVL tree of

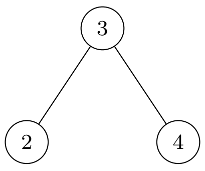
(i) maximum height.

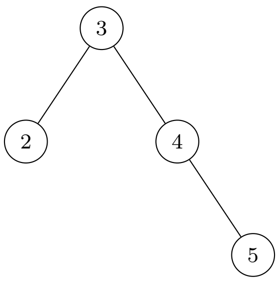
(ii) minimum height.

(i) maximum height. $< 3,\ 2,\ 4,\ 5,\ 6,\ 7,\ 1 >$

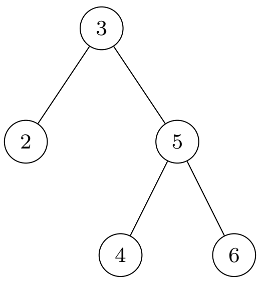




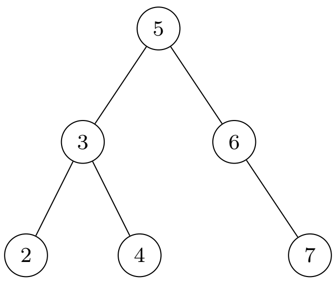


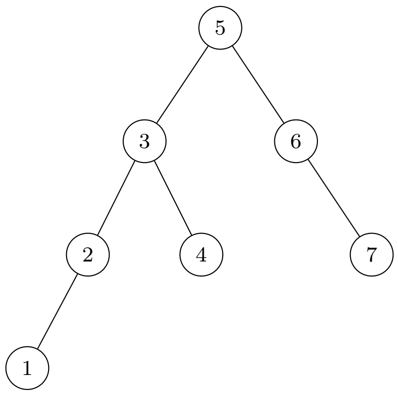


First rotation:

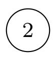


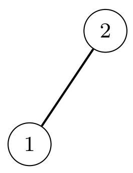
Second rotation:

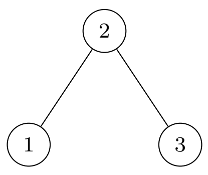


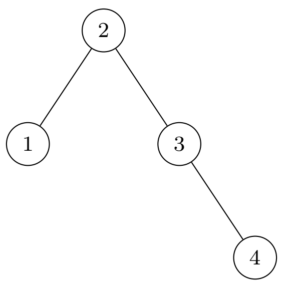


(ii) minimum height. $< 2,\ 1,\ 3,\ 4,\ 6,\ 5,\ 7 >$

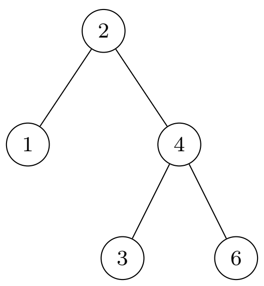








First rotation:



Second rotation: