SHOUZAB KHAN
SKHAN6

Theory Assignment 3

CS 452/652/752 Advanced Algorithms and Applications

Question 1 - 10 points

Simulated Robot Problem: Use dynamic programming to find the total number of distinct paths from the start to the end position. The robot can only move towards down or right direction. Also, the robot cannot go through road blockers (shaded areas). You must show how you fill the memorization table.

ANSWER:

Start	1	X	X	X	0	0	X	X	X
1	2	X	X	X	0	0	X	X	X
X	2	X	X	X	0	0	X	X	X
0	2	2	X	0	0	0	X	X	X
0	2	4	4	4	4	4	X	X	X
X	2	X	X	X	4	8	8	8	8
0	2	X	X	X	4	X	8	X	8
0	2	2	X	X	4	4	X	0	8
X	2	4	4	X	4	X	0	X	8
X	X	4	8	8	12	12	12	12	END 20

Question 2 - 10 points

Matrix Chain Product Problem: Use dynamic programming to find a order of computing the chain product of the follow 8

metrics with minimum cost. The cost is defined as total number of multiplications. You should show both what is the minimum cost and what is the order of multiplication (you may express this order as a fully parenthesized expression). You must show how you fill the memorization tables. For this question, you need to show two tables: one for finding the minimum cost and another one for finding the actual
multiplication order.

$M1 \times M2 \times M3 \times M4 \times M5 \times M6 \times M7 \times M8$

- M1 : 20×40
- M2 : 40×10
- M3 : 10×80
- M4 : 80×100
- M5 : 100×50
- M6 : 50×40
- M7 : 40×80
- M8 : 80×100

ANSWER:

M-table:

This table shows the minimum multiplication costs for different matrix combinations.

	M1	M2	M3	M4	M5	M6	M7	M8
M1	0	8000	24000	184000	190000	198000	262000	382000
M2	-	0	32000	120000	150000	166000	214000	302000
M3	-	-	0	80000	130000	150000	182000	262000
M4	-	-	-	0	400000	520000	776000	1416000
M5	-	-	-	-	0	200000	520000	1020000
M6	-	-	-	-	-	0	160000	520000
M7	-	-	-	-	-	-	0	320000
M8		-	-	-	-	-	-	0

S table

	M1	M2	M3	M4	M5	M6	M7	M8
M1	0	1	2	3	3	3	3	3
M2	-	0	2	3	3	3	3	3
M3	-	-	0	3	3	3	3	3
M4	-	-	-	0	4	5	5	5
M5	-	-	-	-	0	5	5	5
M6	-	-	-	-	-	0	6	6
M7	-	-	-	-	-	-	0	7
M8		-	-	-	-	-	-	0

(M[1][8]): 382,000 multiplications

(S[1][8]):

Question 3 - 15 points

0/1 Knapsack Problem: Given 10 items with the follow weight table and profit table and a knapsack with weight capacity K = 10. Use dynamic programming to find a set of items to put in the knapsack that yields the maximum total profit. You should show both what is the maximum profit and
which items are selected. You must show how you fill the memorization tables. For this question, you need to show two tables: one for finding the maximum profit and another one for finding the actual items selected.

weight = [1, 4, 2, 3, 2, 5, 2, 6, 1, 6]
profit = [20, 50, 30, 30, 40, 20, 10, 40, 10, 30]

ANSWER:

M	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	20	20	20	20	20	20	20	20	20	20
2	0	20	20	20	50	70	70	70	70	70	70
3	0	20	30	50	50	70	80	100	100	100	100
4	0	20	30	50	50	70	80	100	100	110	130
5	0	20	40	60	70	90	90	110	120	140	140
6	0	20	40	60	70	90	90	110	120	140	140
7	0	20	40	60	70	90	90	110	120	140	140
8	0	20	40	60	70	90	90	110	120	140	140
9	0	20	40	60	70	90	100	110	120	140	150
10	0	20	40	60	70	90	100	110	120	140	150

MAX PROFIT = 150

S	0	1	2	3	4	5	6	7	8	9	10
0	E	E	E	E	E	E	E	E	E	E	E
1	E	I	I	I	I	I	I	I	I	I	I
2	E	E	E	E	I	I	I	I	I	I	I
3	E	E	I	I	I	E	I	I	I	I	I
4	E	E	E	E	I	E	I	E	I	I	I
5	E	E	I	I	I	I	I	I	I	I	I
6	E	E	E	E	E	E	E	E	E	E	E
7	E	E	E	E	E	E	E	E	E	E	E
8	E	E	E	E	E	E	E	E	E	E	E
9	E	E	E	E	I	E	I	E	I	E	I
10	E	E	E	E	E	E	E	E	E	E	E

Items Selected=[1,1,1,0,1,0,0,0,1,0]

Question 4 - 15 points

0/1 Knapsack Problem: Use contradiction to prove that optimal sub-structure property exists in the 0/1 Knapsack Problem.

ANSWER:

To show that the 0/1 Knapsack Problem has an "optimal substructure" property, we can use a proof by contradiction. This means we'll assume the opposite of what we want to prove and see if it leads to something that doesn't make sense.

Problem Setup

Suppose we have a knapsack with a maximum weight capacity K=50. We also have a set of items, each with a specific weight and profit value. Our goal is to maximize the total profit without exceeding the knapsack's capacity.

Proof by Contradiction

Let's assume that the 0/1 Knapsack Problem does **not** have an optimal substructure property. If this were true, it would mean that the best (optimal) solution for a knapsack of capacity K=50 could not be built from the best solutions of smaller capacities.

Now, let's say that the best possible solution for K=50K=50 includes an item, **Item 1**, with a weight of 10 and a profit of 80. By including this item, we use up 10 units of capacity, leaving us with a remaining capacity of K−10=40.

If we have the optimal solution for this smaller capacity K=40, let's call it P40, then we could add the profit of **Item 1** (which is

80) to P40P40 to get the optimal solution for the total capacity K=50. This would mean the optimal solution at K=50 depends on the best solution for K=40.

This contradicts our assumption that the problem doesn't have an optimal substructure! Since we can actually build the best solution for K=50 by combining **Item 1** and the optimal solution for K=40, it shows that the 0/1 Knapsack Problem does have an optimal substructure.

Conclusion
So, because we could use the solution to a subproblem (capacity K=40) to build the best solution for the larger problem (capacity K=50), we see that the 0/1 Knapsack Problem has an optimal substructure.

Therefore, the solution to the main problem at K=50 contains within it an optimal solution for the subproblem at K=40

Question 5 - 10 points
Explain what is stable sort and what is an unstable sort. You may use examples to explain.

ANSWER:

Stable sorting:
Is a sorting algorithm that keeps the relative order of elements with the same key or value. In other words, if two elements are equal, their original order is preserved in the sorted list. Stability

can be important in cases where we want to maintain the initial arrangement of items that have the same value.

Unstable sorting:
is a sorting algorithm that does not guarantee the preservation of the relative order of elements with equal keys. In an unstable sort, elements with the same key might end up in a different order than in the input. Unstable sorting algorithms can sometimes be faster or require fewer operations than stable ones.

Example:

Consider a list of students with their names and marks:

Input Sequence:
(Liam, 90) (Noah, 85) (Emma, 90) (Olivia, 75) (Ava, 85)

n a stable sort (e.g., Merge Sort), if we sort the students by their marks, the output will keep the original order of students with the same marks.

Sorted Output:
(Olivia, 75) (Noah, 85) (Ava, 85) (Liam, 90) (Emma, 90)

Notice that students with the same marks, like Liam and Emma (both 90), stay in the same order as they were in the original list.

Unstable Sort Example:

In an unstable sort (e.g., Quick Sort), sorting the same list by marks might result in a different order for students with the same marks:

Unstable Sorted Output

(Olivia, 75) (Ava, 85) (Noah, 85) (Emma, 90) (Liam, 90)

Liam and Emma (both with 90) don't maintain their original order, showing the instability of the sorting process.

Question 6 - 20 points

The quick sort algorithm discussed in the lecture is not a stable sort algorithm because the partition operation does not preserve the order of elements with same key value. Design an improved linear time partition operation that preserves the order so it can enable stable, quick sort. You may use extra memory in your design. Explain why it runs in linear time.

ANSWER:

A stable quicksort keeps the original order of elements with the same key. To make quicksort stable, we modify the partition operation using extra memory.

Stable Partition Operation

1. **Three-Part Partitioning**: We split elements into:
    - **Left Partition**: Elements less than the pivot.
    - **Middle Partition**: Elements equal to the pivot.
    - **Right Partition**: Elements greater than the pivot.

2. **Auxiliary Arrays**: We use two arrays:
    - **Left Array** for elements less than the pivot.
    - **Right Array** for elements greater than the pivot.
3. **Collecting and Reconstructing**: We go through the array once (O(n) time) to place elements in these arrays. Then, we combine the left, middle, and right partitions back into the original array.

Why This Runs in Linear Time:

Since we only make one pass through the array to organize elements into auxiliary arrays and then merge them back, this modified partition operation runs in **O(n)** time. By applying this recursively, we achieve a stable quicksort.

Question 7 - 10 points

Use Radix sort to sort the following sequence. Please show intermediate sorting results (after sorted by each digit) and final sorting results.

Input: [3453, 2675, 1211, 7642, 1352, 121, 8642, 6263, 135, 12]

ANSWER:

Initial	Count 1	Count 2	Count 3	Count 4
3453			12	**12**
2675	1211,121	1211,121	121,135	**121**
1211	7642,1352,8642,12	12	1211,6263	**135**
7642	3453,6263	135	1352	**1211**
1352		7642,8642	3453	**1352**

121	2675,135	3453,1352		**2675**
8642		6263	2675,7642,8642	**3453**
6263		2675		**6263**
135				**7642**
12				**8642**

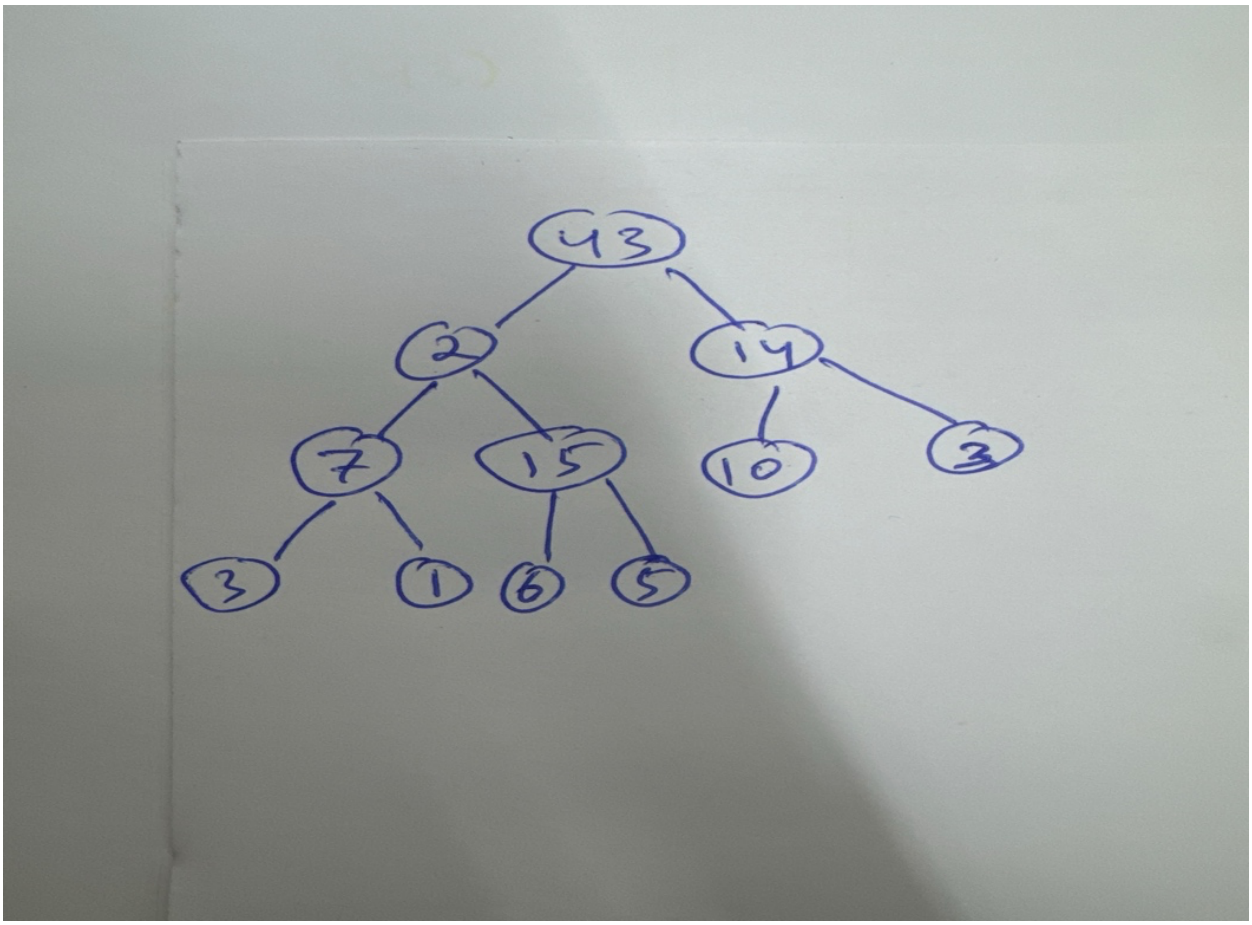
Question 8 - 10 points

The follow nodes satisfy the max-heap property except for one node. (1) Identify which node does not satisfy the max-heap property. (2) Show step by step how MAX-HEAPIFY operation fix this max-heap.

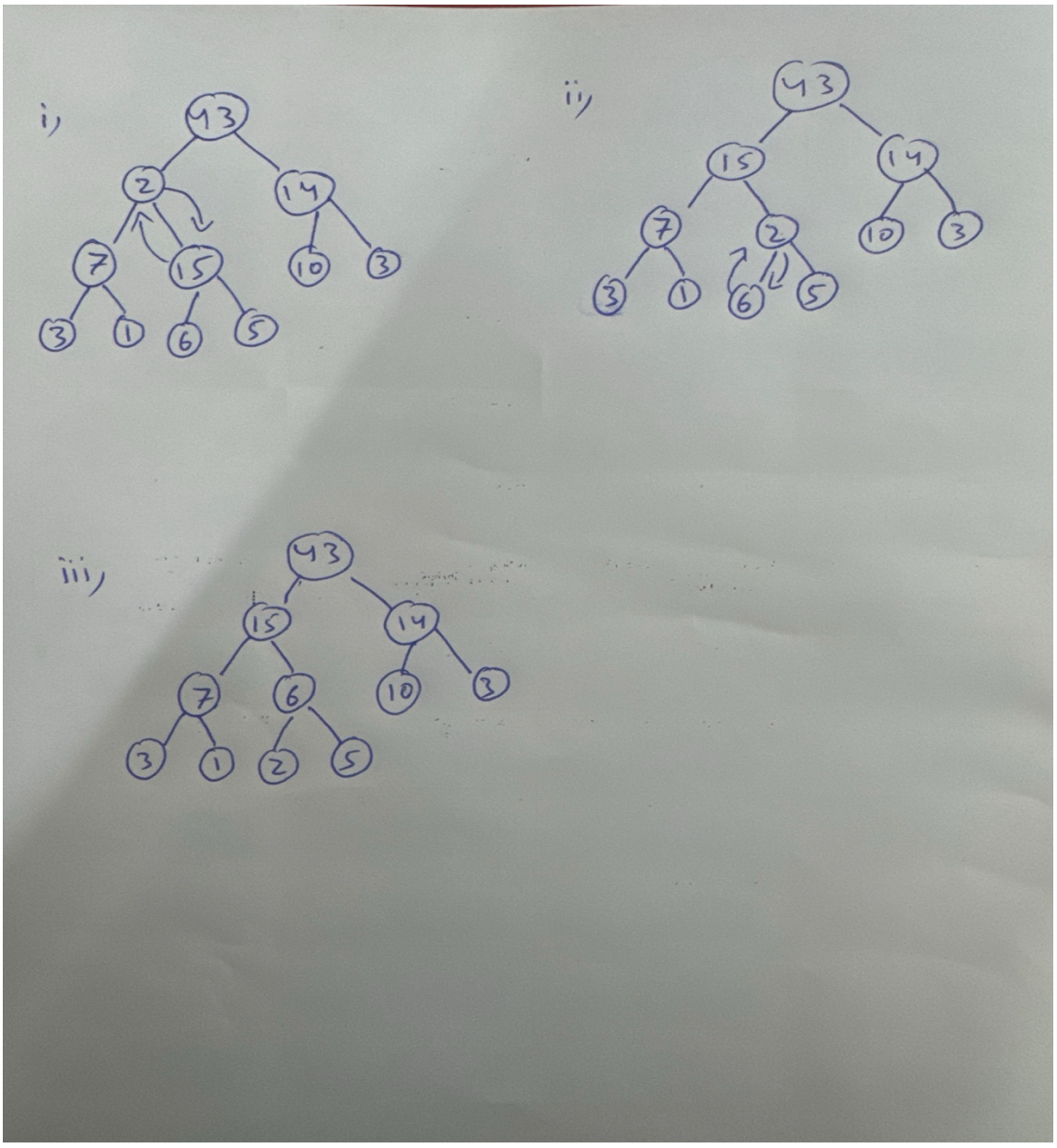
ANSWER:

STEP 1:

Node 1 does not fully satisfy the max heapify operation.
we use the following max heapify operations:


43
2
14
7
15
10
3
3
1
6
5

STEP 2:



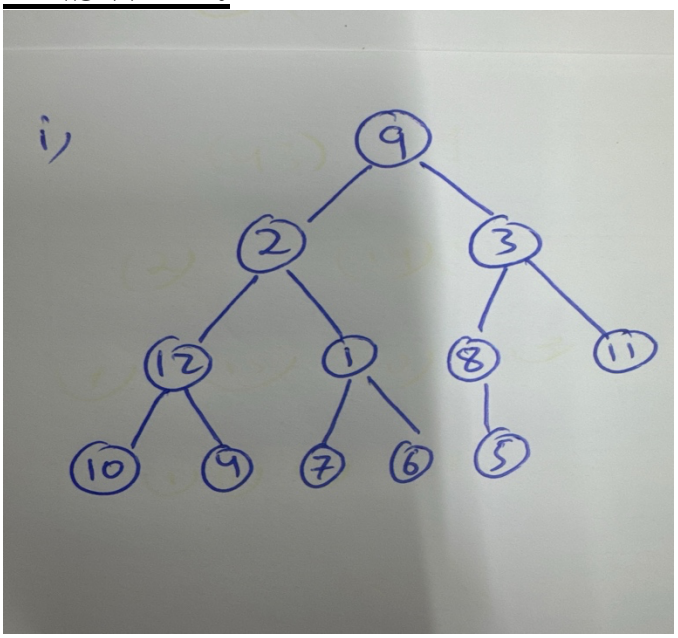
Now, All nodes satisfy the properties of max heapify.

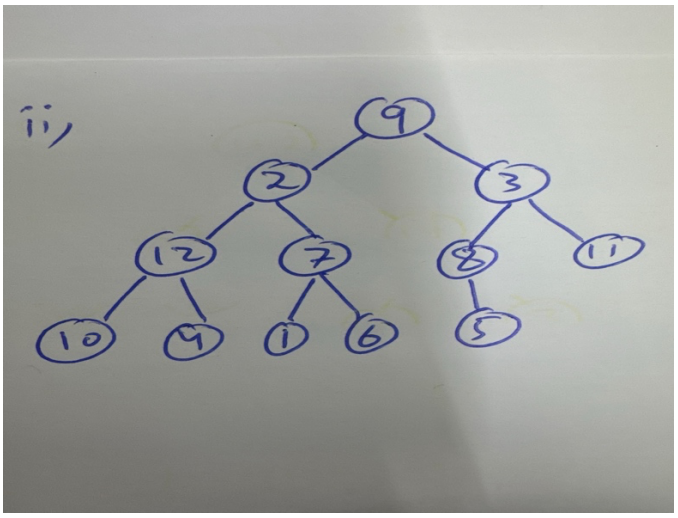
Question 9 - 20 points

Given an unordered input array. Show step by step how the BUILD-MAX-HEAP operation build a max-heap from this array.

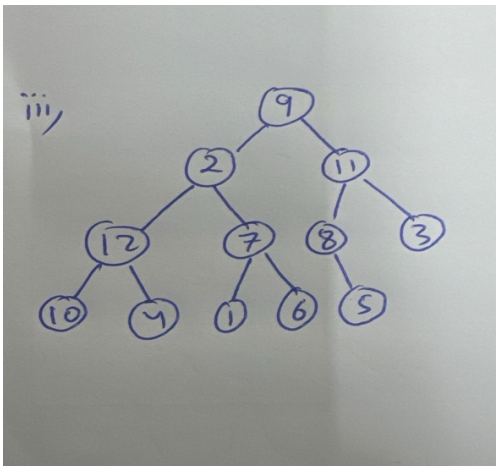
A = [9, 2, 3, 12, 1, 8, 11, 10, 4, 7, 6, 5]

ANSWER:

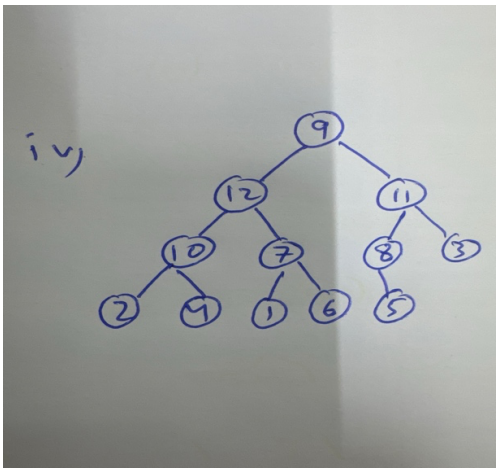




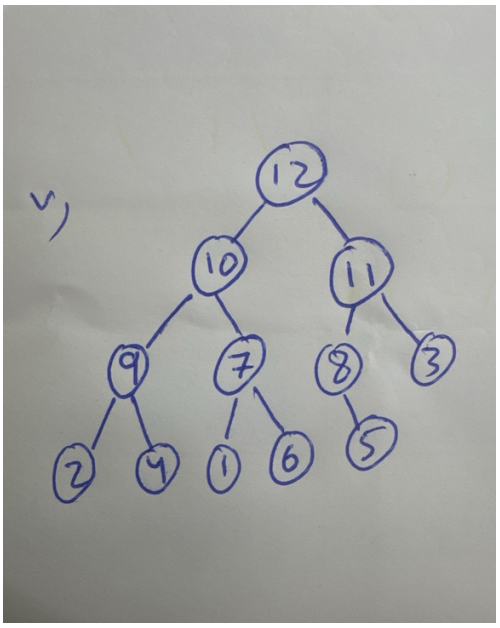
iii)



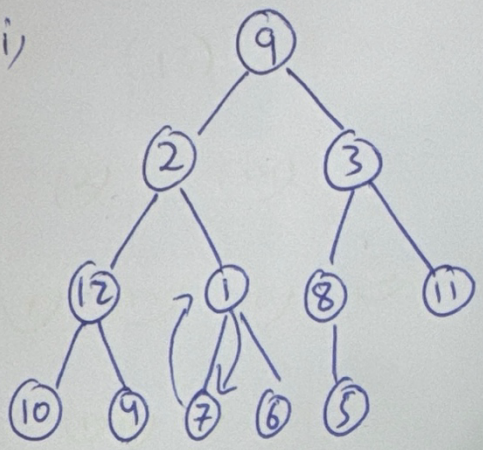
iv)



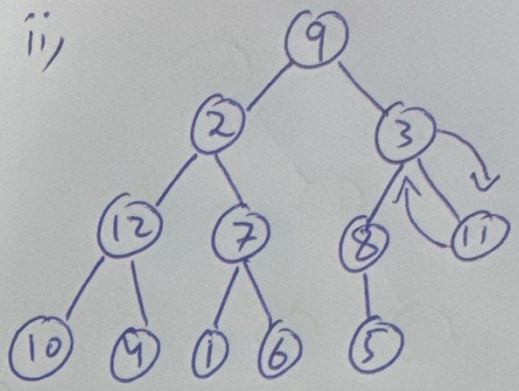
v)



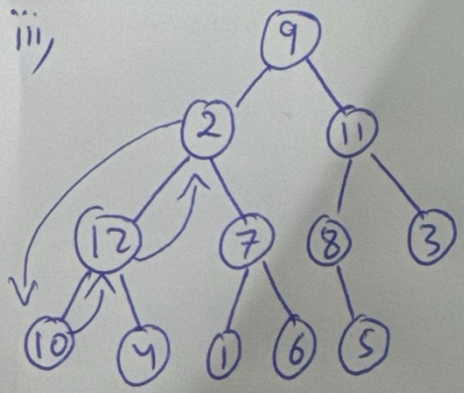
i)


9
2
3
12
1
8
11
10
4
7
6
5

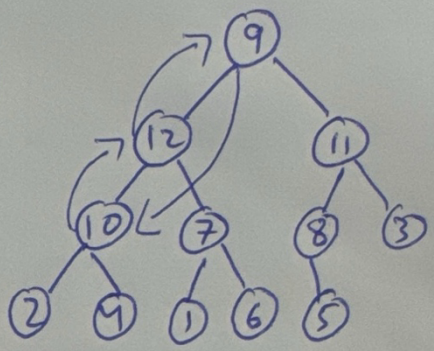
ii)


9
2
3
12
7
8
11
10
4
1
6
5

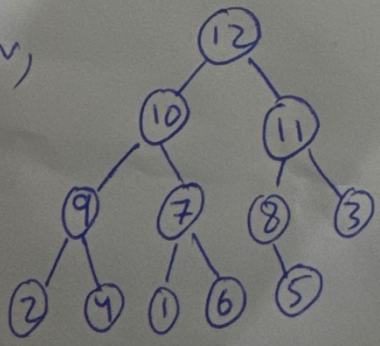
iii)


9
2
11
12
7
8
3
10
4
1
6
5

iv)


9
12
11
10
7
8
3
2
4
1
6
5

v)


12
10
11
9
7
8
3
2
4
1
6
5