Written Assignment

Q1. Answer the following questions:

a) Let T be an ordered tree (i.e., the children of each node are ordered) with more than one node. Is it possible that the in-order traversal of T visits the nodes in the same order as the pre-order traversal? If so, give an example; otherwise argue why it cannot happen. Likewise, is it possible that post-order traversal of T visits the nodes in the reverse order as pre-order traversal? If so, give an example; otherwise argue why it cannot happen.

Either in postorder or preorder traversal, it is not possible to visit the nodes in the same order. Because for postorder case, first visiting case is always external node but for preorder case, first visiting case is always root. Since the question indicates node has more than one case so it is impossible.

However, it is possible to visit reverse order. Assume that we have 2 nodes in the tree. Then it only exist parent node (called A, root) and child node (called B, leaf). So for the postorder case, it will visit B and then A which is reverse order of preorder, which will visit A and then B.

b) Answer the previous questions if T is a proper binary tree with more than one node.

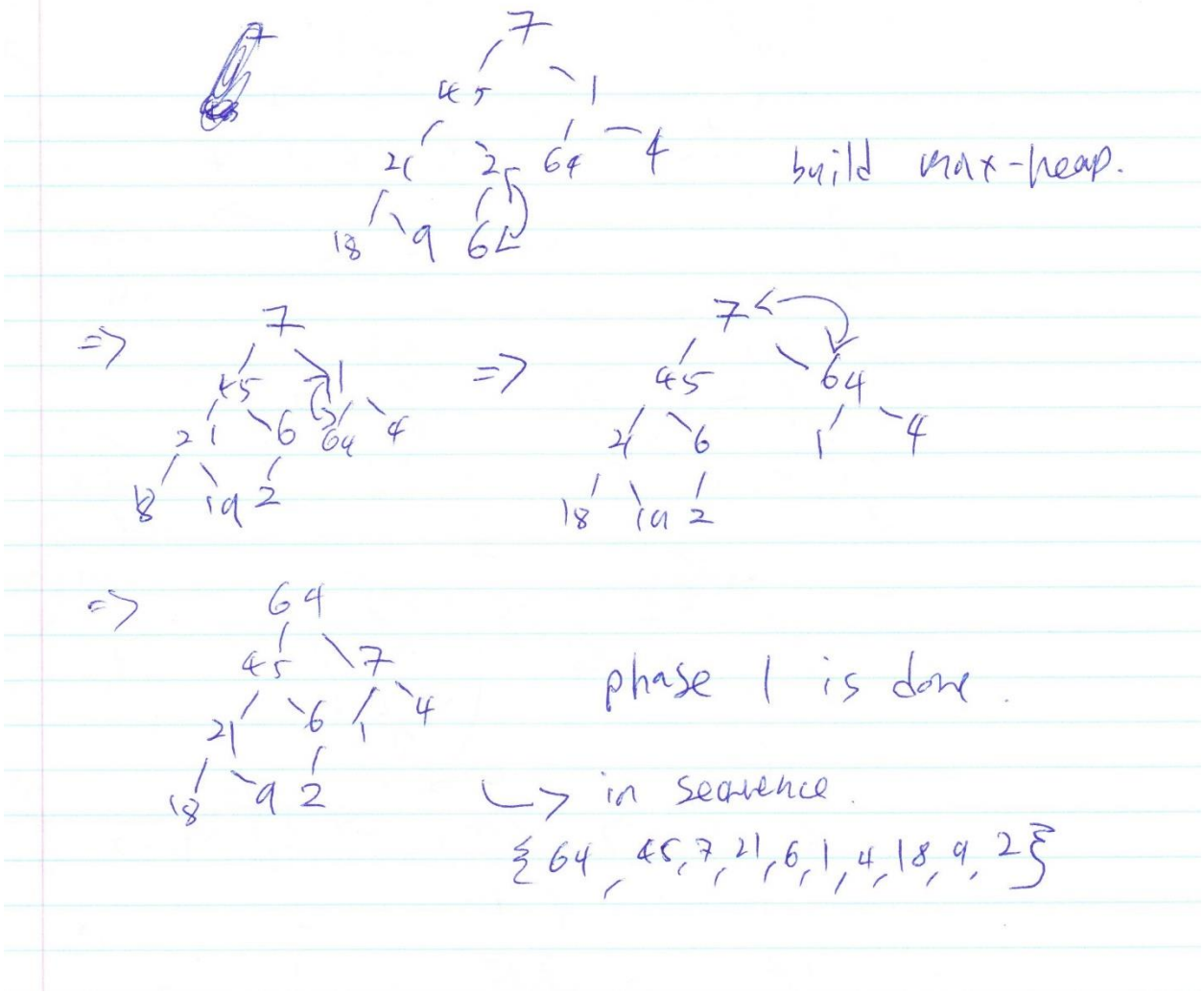
As the same reason as previous question, it is not possible to visit the nodes in the same order in binary tree since it indicates that it has more than one node.

Unlike previous question, it is also not possible to visit in reverse order. Proper binary tree has 2 child nodes except leaves. Assume that we have 3 nodes, root, called A, and 2 child B (left) and C (right), leaves. For the postorder case, it will visit A -> B -> C and for the preorder case, it will visit B -> C -> A. However, postorder's reverse is not preorder. The reason is that both of them visit the left one first. To be satisfied, one of them should visit right child first but this is not the case.

Q2. Answer the following question:

a) Show (diagrammatically) all the steps involved in the heap sort on the following input sequence: 7, 45, 1, 21, 2, 64, 4, 18, 9, 6. You should sort the sequence in the increasing order.

To have increasing order, we need to build a max-heap first. (It is 1st phase)



7	45	1	21	2	64	4	18	9	6
7	45	64	21	6	1	4	18	19	2
64	45	7	21	6	1	4	18	19	2
64	45	7	21	6	1	4	18	9	2

Using removeMin(), we can do following, (2nd phase)

Sorted sequence	Heap
{}	{64,45,7,21,6,1,4,18,9,2}
{1}	{64,45,7,21,6,4,18,9,2}
{1,2}	{64,45,7,21,6,4,18,9}
{1,2,4}	{64,45,7,21,6,18,9}

{1,2,4,6}	{64,45,7,21,18,9}
{1,2,4,6,7}	{64,45,21,18,9}
{1,2,4,6,7,9}	{64,45,21,18}
{1,2,4,6,7,9,18}	{64,45,21}
{1,2,4,6,7,9,18,21}	{64,45}
{1,2,4,6,7,9,18,21,45}	{64}
{1,2,4,6,7,9,18,21,45,64}	{}

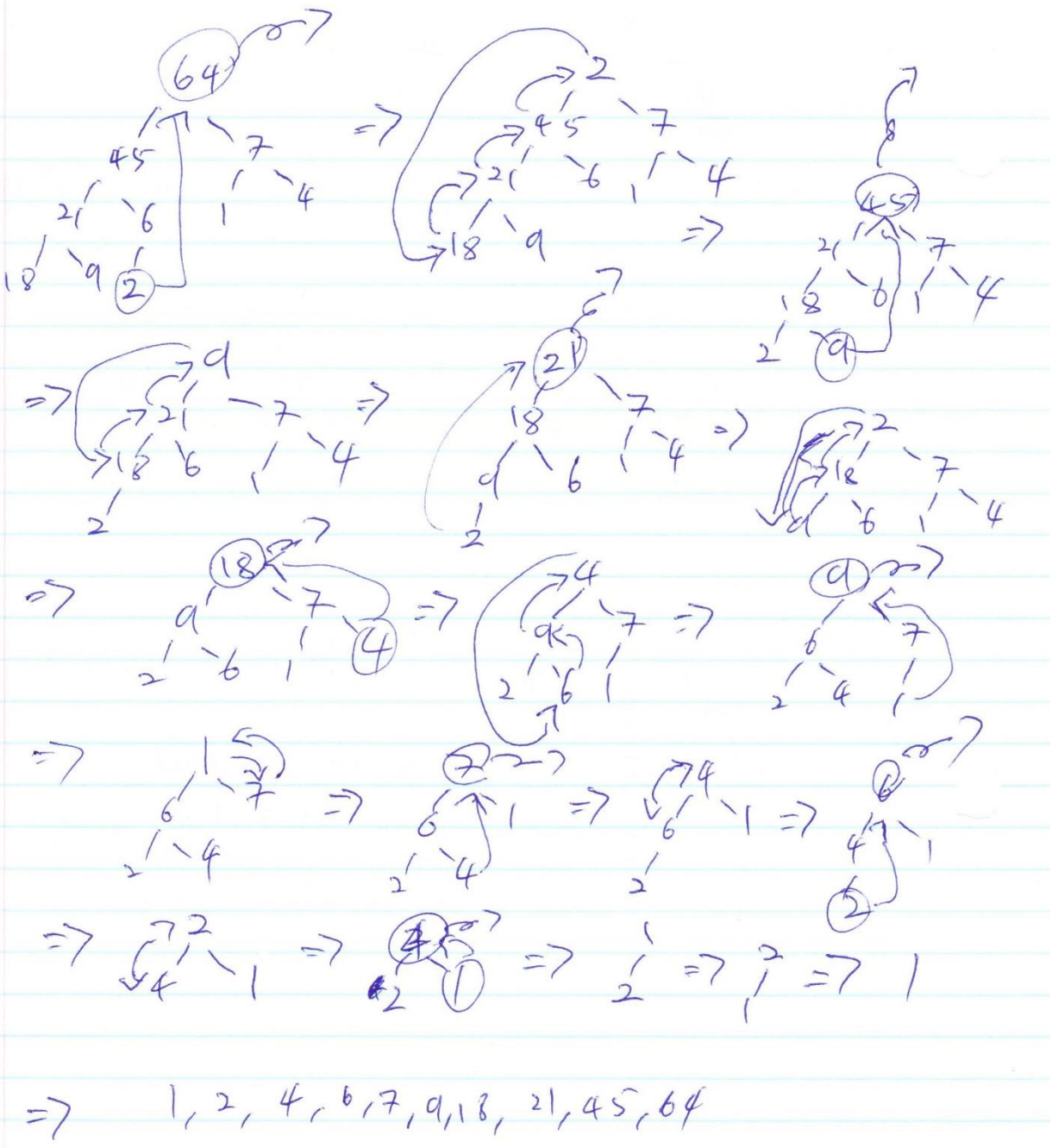
b) Repeat part a) if the heap sort is performed in place.

For the first phase, it is the same procedure which we need to build a max-heap.

64	45	7	21	6	1	4	18	9	2

However, we are required to perform in place, Therefore,

2	45	7	21	6	1	4	18	9	64
45	21	7	18	6	1	4	2	9	64
9	21	7	18	6	1	4	2	45	64
21	18	7	9	6	1	4	2	45	64
2	18	7	9	6	1	4	21	45	64
18	9	7	2	6	1	4	21	45	64
4	9	7	2	6	1	18	21	45	64
9	6	7	2	4	1	18	21	45	64
1	6	7	2	4	9	18	21	45	64
7	6	1	2	4	9	18	21	45	64
4	6	1	2	7	9	18	21	45	64
6	4	1	2	7	9	18	21	45	64
2	4	1	6	7	9	18	21	45	64
4	2	1	6	7	9	18	21	45	64
2	1	4	6	7	9	18	21	45	64
1	2	4	6	7	9	18	21	45	64



As shown in the drawing, increasing order sequence is placed.

Q.3. Answer the following questions:

a) Bill suggested that he can implement a queue ADT using a priority queue and a constant amount of additional space. Recall that a queue supports enqueue and dequeue operations, and a priority queue supports min(), removeMin and insert operations. Explain how Bill can achieve this by showing the necessary pseudo code for implementing the queue ADT.

Assume that each entry of priority queue has key k and value v.

Using priority queue called pq as a global.

The priority queue has interface for entry (key and value) which is

Interface entry<K,V>

K getKey();

V getValue();

Algorithm EnQueue(val)

Input: a input which want to added in queue

Output: void

If pq.min() is null then

```
    pq.insert (1, val);
```

Else

```
    temp = pq.min();

    pq.insert(temp.getKey()+1,val);
```

Algorithm DeQueue()

Input: void

Output: dequeued value // note that we do not require key for normal queue.

If pq.min() is null then

    Return null;

Else

    Return pq.removeMin().getValue();

In this way, first in and first out since we are using removeMin and the key is increasing by 1 and prior key is always larger than previous key.

b) Referring to a) above, what are the tight big-Oh time complexities of the enqueue and dequeue operations of the newly designed queue given that the priority queue is implemented using a heap? Explain your answer.

While using heap, the new entries will be always new leaf of heap and there will not have any arrangement. The rearrangement will occur only when dequeue operation is called.

As we use priority queue, we know where is last node (or insertion node) so enqueue time complexity is **O(1)**. Note that normal heap insertion complexity is O(log(n)) due to upheap occurs; however, for implementing queue, upheap will never occurs because last node has the largest key all the time based on pseudo code written in (a).

For dequeue, we remove the root (which is the minimum key) and the last node becomes root node. As we discussed, last node has largest key and it becomes the root, so it has to be rearranged (with downheap) . Therefore, dequeue time complexity is **O(log(n)).**