Data structures HW #8

① (a) The array would hold the nodes in order of height, from top to bottom, and within each level, nodes would be stored from right to left. (Same as binary heap) let it denote the index of any Node N in the array A. (assuming i starts at 0).

The left child of N can be accessed with $A[3*i+1]$

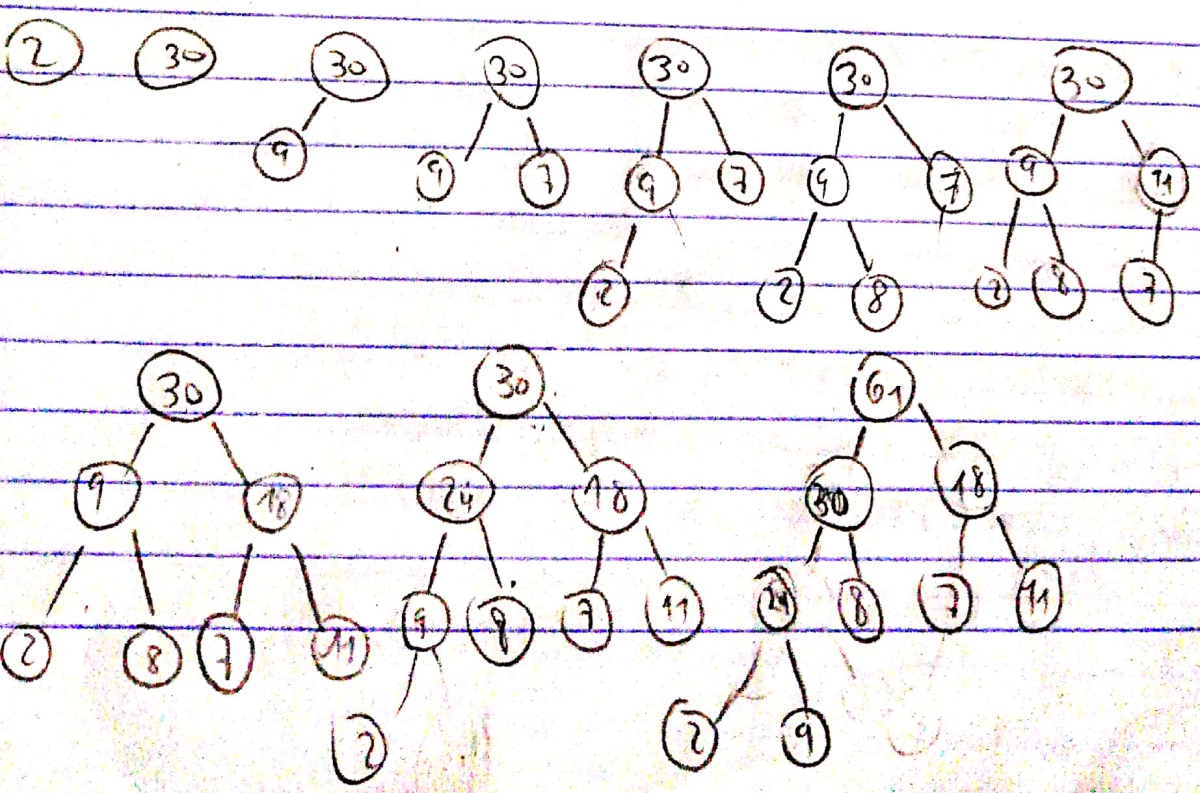
The middle child of N can be accessed with $A[3*i+2]$

The right child of N can be accessed with $A[3*i+3]$ or $A[4*i]$.

The parent of N can be accessed with $A[(i-1)/3]$ (where / performs floor division).

(b)
```
int* build3heap(int* arr, int n)
{   int* heap = new int[n]
    for(int i=0 ; i<n ; i++)
    {   heap[i] = arr[i]
        for(int j=i , heap[(j-1)/3] < heap[j] , j=(j-1)/3)
            { swap (heap[(j-1)/3] , heap[j]) }
    }
}
```

(c) $O(n \log_3 n)$

② 

30 → 30 9 → 30 9 7 → 30 9 7 2 → 30 9 7 2 8

→ 30 9 11 2 8 7 → 30 9 18 2 8 7 11

→ 30 24 18 9 8 7 11 2 → 61 30 18 24 8 7 11 2 9

③ A = < 4, 8, 22, 17, 4, 33, 15 >

4 8 22 17 4 33 15 → 33 17 22 8 4 15 4 → 4 17 22 8 4 15 | 33

22 17 15 8 4 4 | 33 → 17 15 8 4 4 | 22 33 → 15 8 4 4 | 17 22 33

→ 8 4 4 | 15 17 22 33 → 4 4 | 8 15 17 22 33 → 4 | 4 8 15 17 22 33

4 4 8 15 17 22 33

④ Take the first node from each of the k lists and insert them into a min-heap, while the heap is not empty, extract the minimum value from the heap and insert it to the end of the result list and replace the node with the next node if it exists, otherwise, simply remove the node from the heap

```
list <int> MergeSortedLists (List<int>* arrayofLists, int k)
{   MinHeap<List::node*> heap                    //O(1)
    for (int i=0, i<k, i++) heap.insert (arrayofLists [i] -> head)    //O(k lg(k))
    list <int> mergedlist
    while (!heap.isEmpty())        //O(kn)
      { list::node * tmp = heap.pop ();  //O(lg(k))
        mergedlist.append (tmp -> val)  //O(1)
        if (tmp -> next) heap.insert (tmp -> next)  //O(lg(k))
      }
    return merged list
}
```

(5) ① Delete a node from the array
(this creates a "hole" and the tree is no longer "complete")
② Replace the deletion node
with the "farthest right node" on the lowest level of the Binary tree
(This step makes the tree into a "complete binary tree")
③ Heapify (fix the heap):
if (value in replacement node < its parent node)
Filter the replacement node UP the binary tree
Use
Filter the replacement node down the binary tree.

(6)

you can find the second greatest value between the indexes $\frac{n}{4}$ to n because the value would be found either on the last level or one of the last level's parents

(7) yes, keep a variable initialize it to 0, every time you insert into the priority queue pass the variable as the priority then decrement the variable. To dequeue an element just remove (highest priority) from the priority queue.