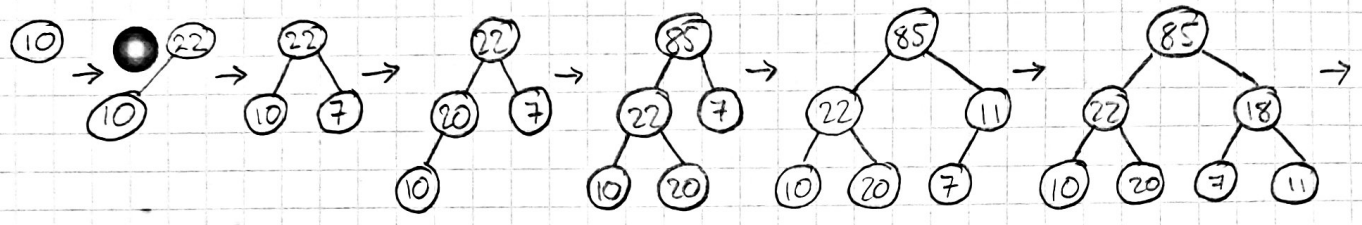
Abraham Murciano - Data Structures - HW 8

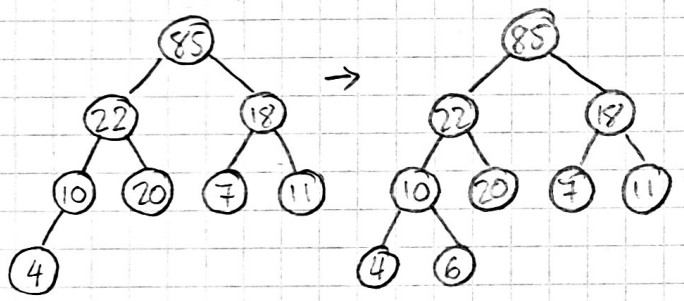
1) a) The array would hold the nodes in order of height, from top to bottom, and within each level, nodes would be stored from right to left. (same as binary heap)
Let i denote the index of any node N in the array A. (assuming i starts at 0)
The left child of N can be accessed with $A[3*i+1]$.
The middle 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).

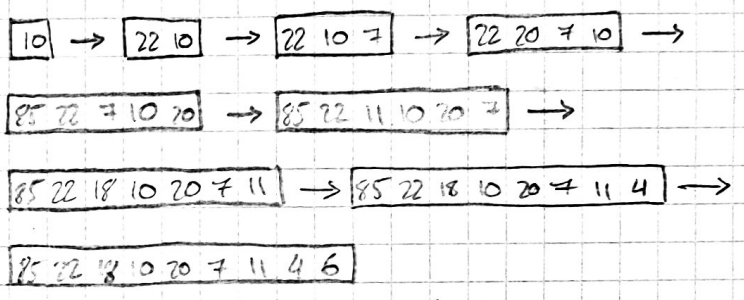
b)
```
int* build3heap(int* arr, int n){
    int* heap = new int[n];
    for(int i=0; i<n; i++){
        heap[i] = arr[i];
        for(int j=i; heap[(j-1)/3] < heap[j]; j=(j-1)/3){
            swap(heap[(j-1)/3], heap[j]);
        }
    }
}
```

c) $O(n \log_3 n)$

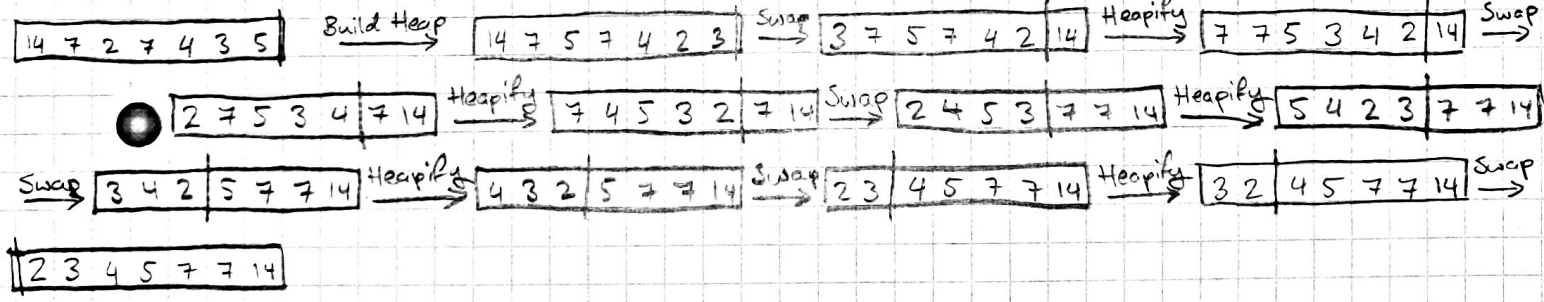
2)







3)



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 mergedList;
}
```

5)
```
void help(heap& h, int i, int m){ // recursively add m to each node below i
    if(h[i]){
        h[i]+= m;
        help(h, h[i]->left->index, m);
        help(h, h[i]->right->index, m);
    }
}
heap add(heap h, int i, int m){
    help(h, i, m);  // O(n)
    for(; i!= 0; i = h[i]->parent->index){  // O(lg(n))
        h[i]->parent->heapify;  // O(lg(n))
    }
}
```

Complexity = $O(n + lg^2 n) = O(n)$

6) Given a max heap implemented in an array, the maximal value of the heap would be at index 0, because in a max heap the root is the maximal value, and when implemented in an array, the values are stored in order from top to bottom, from right to left. Since the root is the top-most value, it will be the first value of the array

7) Yes. Keep a variable, initialise 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 removeHighestPriority() from the priority queue.