CS5800: Algorithms Spring 2018
Assignment 5.1

Saptaparna Das

March 16, 2018

(1) Algorithm of priority queue:

```
Let Node consists of an item and priority of that item
Also, we have an array items[] to hold list of items, which
acts as a max heap. I have assumed the root of the max heap is the item
with highest priority.
The array items is a zero based array, in which the items are stored from index 1,
to maintain index relationship of parent/child and heapsize is total number of present items
in the heap.

add(item, priority)
1. Create a new Node consisting the given item and it's priority
2. Store the new Node after the last occupied space in items array
3. Store this index of new item
4. Till index is not 1 and priority of new item is greater than priority of its parent
    (item at index/2 position),
      repeat the following steps:
        a) Send parent item in place of child item/new item
        b) Make index half
5. Store new item in current position



getItemWithHighestPriority()
1. Store root of the heap or first element of the array
2. Store the last node in heap.
3. Start from parent =1 and child =2
4. Till child is less than or equal to total heapsize, repeat following steps:
        a) get the index of child with max priority (priority of item at child and child +1)
           and store in child
        b) If priority of last is greater than or equals to child, then go to step 5
        c) Store item at child in parent position
        d) Take the value of child in parent
        e) Make child double
5. Store last in parent index of heap
6. return the root



changePriority(item,new priority)

To change the priority of an item we take another data structure say AVL tree, to store the
index of an item in heap.

1. Given the item, get the index of it in the heap from tree
2. Get the old priority of the item from heap array
3. If old priority is same as new priority, do nothing
4. If old priority is more than new priority, then only the tree below it needs to be fixed.
Perform the same operation as getItemWithHighestPriority to sink down the item.
(i.e. keep swapping it with the children having more priority till the priority is more than
 max priority of its children)
5. If old priority is less than new priority, then the tree above it, needs to be fixed.
```

```
Perform same operation as add(item, priority) to bubble up the item.
(i.e. keep swapping with parent till priority of the item is greater than or
equal to its parent)
```

Running time: The add operation takes time proportional to height of the tree in worst case. So its done in O(log n) time. The remove operation also takes same time since extracting root is constant time and rearrange other elements take O(log n). The change priority operation use AVL(self-balanced tree) to store index of each item.So searching in the tree takes log n time and then changing priority of the item in heap and rearranging the heap takes another log n operations. But for the rearrangement in heap all the corresponding items AVL need to be updated. Each of this operations takes log n time so total time should be $(logn)^2$

Another implementation:
If the jobs to be inserted in priority queue are known before then we can achieve O(log n) time complexity of changePriority(item,new priority) function. We need to use a HashTable as secondary datastructure instead of AVL tree. The rest of the process is similar to what has been explained before. Only change is the item and its position in heap is now stored in hashtable. Since, its proven that if keys in a hashtable are static and finite, we can achieve perfect hashing by creating hashtables in every slot of primary hashtable in case of collision. This also ensures O(1) worst case time complexity for hashtable operations. So, the changePriority(item,new priority) can be performed in O(log n) in worst case.