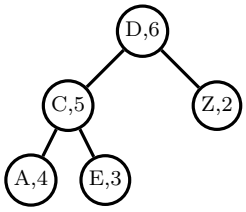
Exercise Sheet, Week 6

Question 1. Each node of the following heap tree stores a value followed by its priority:



Show how the tree changes after each of the following operations (executed in a sequence):

(i) `insert(Q,4)`
(ii) `insert(M,1)`
(iii) `insert(M,10)`
(iv) `deleteMax()`
(v) `insert(H,8)`
(vi) 5× `deleteMax()`

Question 2. Draw the corresponding (complete) trees for the following arrays (we only store priorities). Use the same convention as for heap trees, that is, node on position `i` has its left and right child stored on position `2*i` and `2*i+1`, respectively. Also, the parent of node `i` is stored on position `i div 2`. Assume that we index the arrays starting from 1.

(i) `[10,6,2,5,3,1]`

(ii) `[6,4,5,7,0,2,4,4,0]`

(iii) `[1,1,1,0,0,0,0,0,0,0,0]`

(iv) `[4,3,3,2,1,1,5,1]`

Question 3. Decide which of the trees from Question 2 are heap trees.

Question 4. `bubbleDown` keeps swapping a node with the higher priority child as long as any of its children has a higher priority. Finish the implementation.

```
void bubbleDown(int i, int[] heap, int n) {
    if (left(i) > n) {          // has no children
        return;
    } else if (right(i) > n) { // only has the left child


    } else {                    // has both children









    }
}
```

where `n` is the number of elements stored in the heap, `left(i)` returns `2*i` and `right(i)` returns `2*i+1`.

Question 5. Write `void update(int index, int priority, int[] heap, int n)` which changes the priority of the node stored on position `index` (make sure that the result is a heap tree). You can use `bubbleUp` and `bubbleDown`.