**Runtime analysis, Classes, and functions’ documentation:**

**Class:** BinomialHeap

Description**:**

Functions:

**public HeapItem insert***(int key, String info)* ***|***

First, we will create a HeapItem and HeapNode with the input elements. Then, we will check the current state of the heap, and act accordingly:

1. If the heap is an empty heap, or there is no subtree in the size of 1- we will add the HeapNode as the first element in the subTreesArray.
2. The heap is not an empty heap, and there is a subtree in the size of 1- we will call the helper function singleMeld with this new HeapNode and the index 0.

Then, we will increase the size of the heap by one, and update the min pointer if needed.

In the worst case, the heap is not empty, and all of it’s subtrees are full, and therefore we will have to call the singleMeld function. In this case we have subtrees, so the runtime of singleMeld will be . All of the other actions in this function are conditions and pointers update, which takes time. In conclusion, summing up all the actions will give us the runtime analysis of **.**

**public void deleteMin***()* ***|***

First, we will create a new subtreesArray consists of the children of the minimal node of the current heap. Then, we will create a new heap element using this array, noticing this is not a real heap representation- so we won’t update all of the pointers.

Then, we will check if the minimum node is the root of the largest subtree. If so, we will remove this node. Otherwise we will set the element in this index to be null.

After disconnecting the minimum node, we will call the updateMin, updateLast in order to update the heap’s pointers. In the end we will meld the original heap without the minimum node with the new heap that we created from it’s children.

In the worst case, the minimum node is the root of the largest subtree, the new minimum will be the second largest subtree, and we have all of the subtrees. In this case, updateMin will take because we will iterate over all the roots of the other subtrees until we will reach the minimum.

Then, we will call the function meld with two heaps, which are the same sized. Therefore we will have to meld all of the subtrees. We will meld subtrees, and each meld takes **.** This two parts happens one after the other, so we will sum it up, and will get runtime.

**public void updateLast***()* ***|***

We will check if the heap is an empty tree or not. If not we will return the last root in the subTreesArray.

**public void updateMin***()* ***|***

We will iterate over all of the roots of the subtrees in the subTreeArray, and will compare their value until we will find the minimal root.

If the minimal node is the root of the largest tree- we will iterate over all of the subtrees. If all of the subtrees of the heap exist- we will loop iterations, so the runtime of the function will take **.**

**public HeapItem findMin***()* ***|***

Returning the value of the min pointer.

**Class:** HeapNode

Description**:**

Functions:

**Class:** HeapItem

Description**:**

Functions: