COMP 352- Data Structure & Algorithms
Assignment 2

By Édouard Gagné
ID#: 40061204
Section: AA

29 May 2018

Question 1:

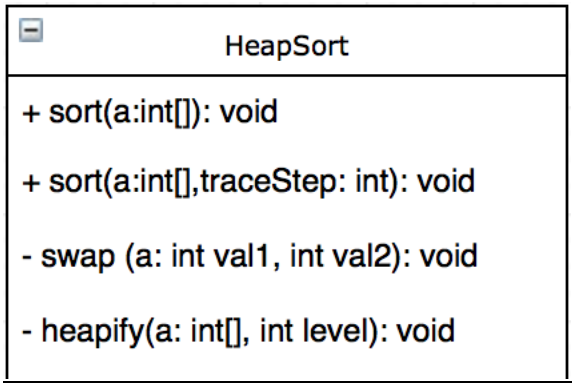

Figure 1: UML diagram of the HeapSort class

To sort the values, the class first define a method called heapify that can heapify a level of the binary tree. The sort method will start by computing the height of the tree and calling heapify on each level of the tree. Heapify checks for each node on a level if its left child is greater than itself and swap these two nodes, then call the heapify method on the level above to check if the node can still move upward. If the left child is not greater than the parent, it will check the right child and do the same processus instead. After the array is heapified, the sort method will do the following processus for each element in the heap: Store the value at the top, compare the two children and swap it with the greater child, and do the same process until it arrives to the prime leaf. Then, the last value in the heap is stored in this spot and then the old root value that was stored is stored at the right place at the end of the array. This process is done again, but the size of the heap is shrunk by one, up until the size reach 0, and a sorted array is obtained.

Pseudocode:

```
Public heapify(int[] heap, int level) {
        for (number of nodes in level) {
                if (left child greater than node) {
                        swap left child and node;
                        If (level is not root) {
                                heapify( heap, level -1);
                        }
                }
                else if (right child greater than node) {
                        swap right child and node;
                        If (level is not root) {
                                heapify( heap, level -1);
                        }
                }
        }
        Return array;
}

Public sort(int [] array) {
        For (each level of the array)
                heapify( array, level );
        For (each element in the array) {
                curr= root position;
                temp=root value;
                For (each level in the array)
                        If (left child is greater than right child)
                                Swap curr value with left child value;
                                Save curr;
                        Else If  (right child is greater than left child)
                                Swap curr value with right child value;
                                Save curr;
                If (current position not equal to end of array position)
                        Swap current position value and end of array value;
                End of array value= temp;
        }
}
```

Question 2 d)

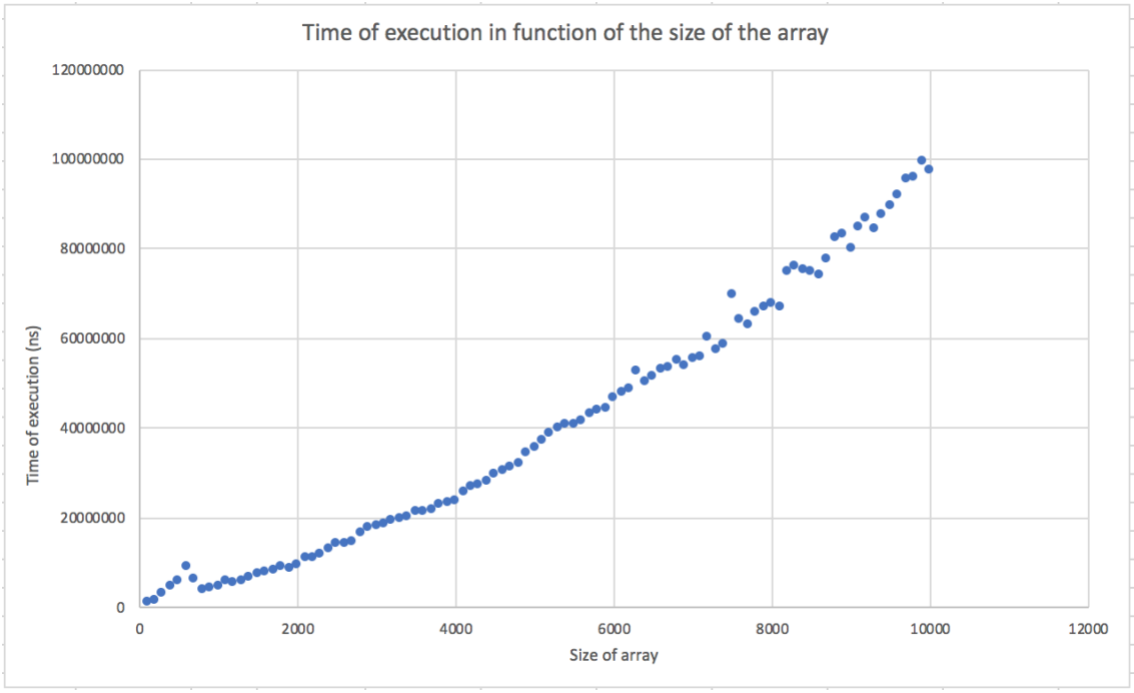

Figure 2: Time of execution in function of the size of the array

Question 3

In both top-down heapsort and bottom-up heapsort, the heapify part of the algorithm are exactly the same. However, in top-down heapsort, one would simply switch the last value and root of the heap and heapify again for all the values to sort the heap. However, for bottom up heapsort, we only "heapify" the prime path of the heap rather than the whole heap, and this results in less comparisons being made than in the top-down algorithm with an equal number of swaps.

Question 4

As stated beforehand, the heap building is implemented as a similar process to sifting down values through the heap and is the same as a top-down approach and is O(n) time complexity. Although the sorting is different than a top-down approach, because the implementation find the prime leaf by swapping nodes with its greater children to rebuild the heap after putting the root value at the end instead of heapifying, it still is O(logn) time complexity and is only faster by a constant margin. Overall, the time complexity of the algorithm is nlogn for all cases, like the top-down approach. Both algorithms are also sorted in place and therefore have a space complexity of O(1).