BLG 335E – Analysis of Algorithms I
Fall2020 Homework 2

Ekin Çelebi
150160152

2.1

void minHeapify(**int** i);
Each nodes values should be smaller than its child nodes to keep the heap property. This function swaps incorrect nodes with a smaller valued child until heap is correct. Complexity is related with the depth of the tree.
Complexity O(logN)

void insert(**double** p);
Insert a value as the last node then shift up the value until its parent is smaller than the inserted node.
Complexity O(logN)

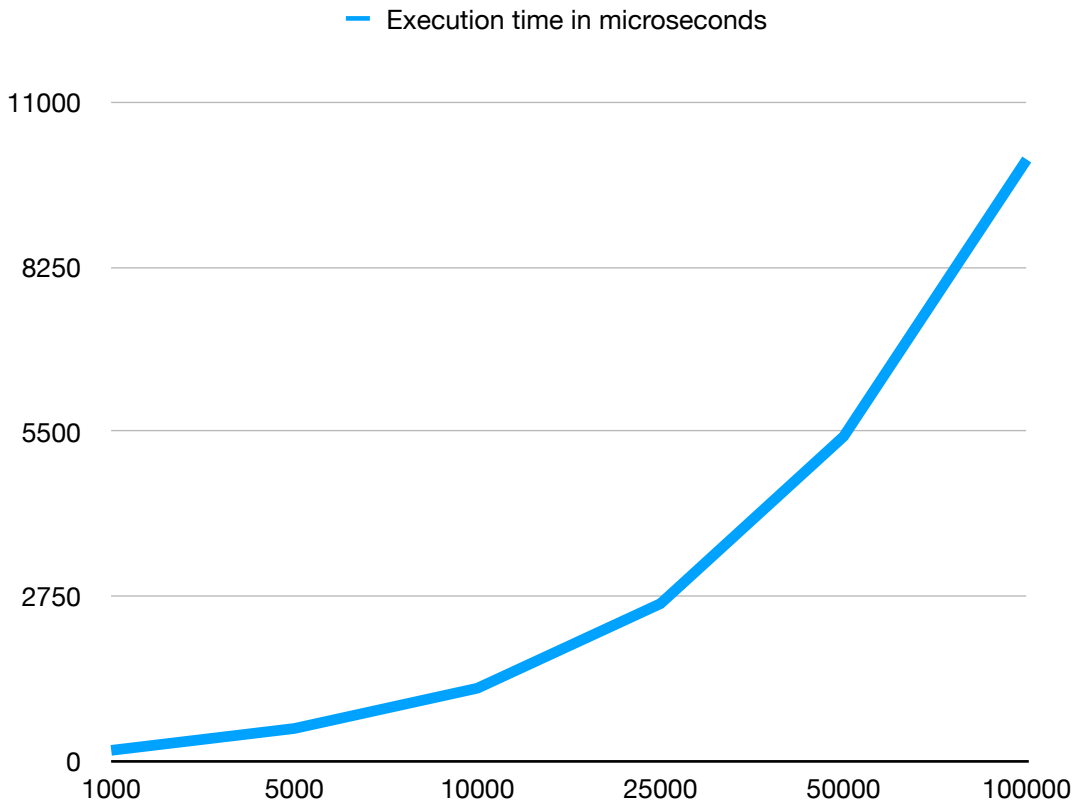
double extractMin();
The minimum value is always the first element of the array, The function returns the index at 0, replaces the 0 index value with the last value of the array then applies minHeapify to shift it down until correct heap is obtained.
Complexity O(logN)

void decreaseValue(**int** i, **double** p);
Changes the value at index I with the given value p.
To keep the heap property, untill we get a higher value on parent node, the nodes value is swapped with the parents value. We are only descreasing the values for the simulation therefore we are only comparing the changed value at index I with its parents.
Complexity O(log N)

2.2

p is chosen to be 0.2

M	1000	5000	10000	25000	50000	100000
Execution time in microseconds	179	544	1217	2630	5422	10048

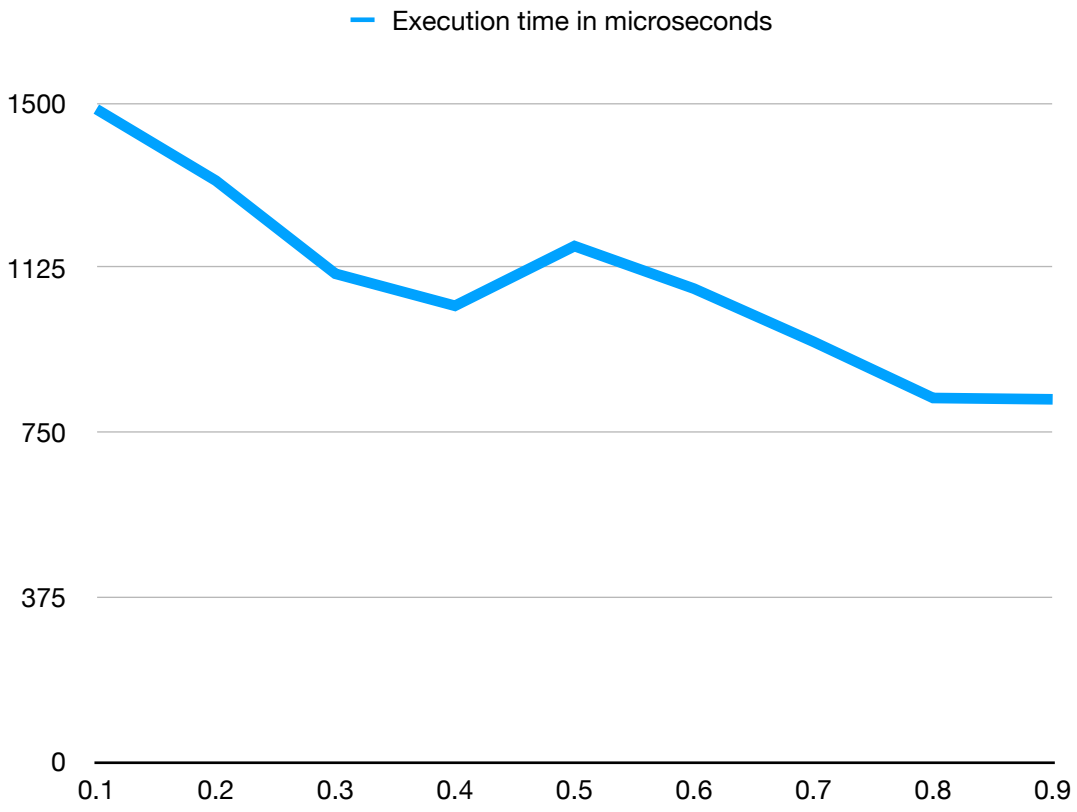


To build a heap, we start with an empty array and keep inserting elements. Heap is a complete binary tree with n nodes and has the smallest possible height of logn.With n elements building the heap takes O(n) time, traversing the heap takes O(logn) time, overall complexity of O(nlogn) time is obtained. We can see the increase related with the element count n. For the small values the change is smaller and for the big values the difference increases faster. It is closer to the O(n) than O(n^2).
In our simulation the graph fits to theoretical running time, because of the fact that we mostly apply insertion(O(logn) complexity), in addition to this the update and extracting minimum have the same complexity.Under these circumstances, I expect the simulation to fit the theoretical running time.

2.3

m is chosen to be 10000

P	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Execution time in microseconds	1486	1323	1111	1038	1174	1077	956	828	825



Both insertion and update alters the heap. For the heap to restore itself shifting/traversing up is necessary. The number of operations depends on how many levels the inserted or the updated value shifted up.The depth of the tree is logN. Thus for both of the insertion and the update operations worst case time complexity is O(logN). As the p value increases the update count increases, and in our case we expect the updated nodes to move less levels than the inserted nodes(needs less operations). Because we only change the value by 0.1, however for the insertion the values generally shifted by more levels to find its parents due to inserted values have more varying values. Our time plots still keeps it O(logN) complexity and the change is minimal. Execution time decreases as the update count increases.