ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 335E

ANALYSIS OF ALGORITHMS I

PROJECT II REPORT

DATE: 23.12.2020
STUDENT NAME: ABDULKADİR PAZAR
STUDENT NO: 150180028

AUTUMN 2020

Project II Report

1)

minHeapify(): The worst case for this function occurs when last row is half full.

The run time of minHeapify() can be described by this recurrence relation:

$T(n) = T(2n/3) + O(1)$ $a = 1$ $b = 3/2$ $d = 0$ $a = b^d$

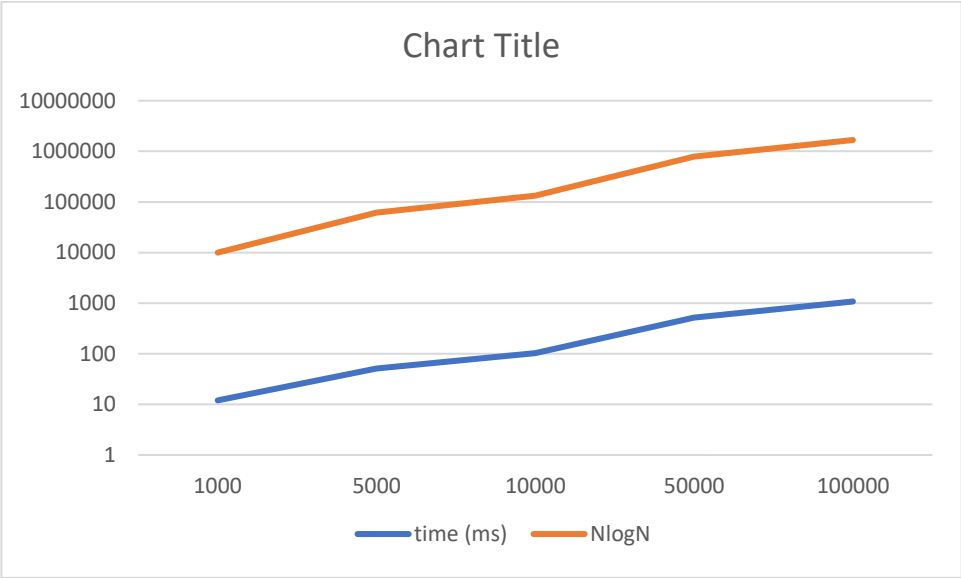
$T(n) = O(\log N)$

decreaseKey(): After the update, the updated element is moved to the correct position in the heap. Because this operation can take at most logN steps, the time complexity of the function is O(logN).

insert(): The inserted element is added to the end of the heap. Because of the percolate up movement, (swapping with parent node) this operation takes logN steps. Its time complexity is O(logN).

popMin(): After removing the root, the last element of the heap is placed to the root. Then the minHeapify() function is called to make sure the array satisfies the minheap property. Because of this, time complexity of popMin() is O(logN).

2)



The graph of computing time for different m values vs NlogN

For the worst-case running time of the program, we want the heap to be large as possible. For that all operations must be addition operations. This would result in following run time relation:

$T(n) = \log(1) + \log(2) + \log(3) + \ldots \log(n)$

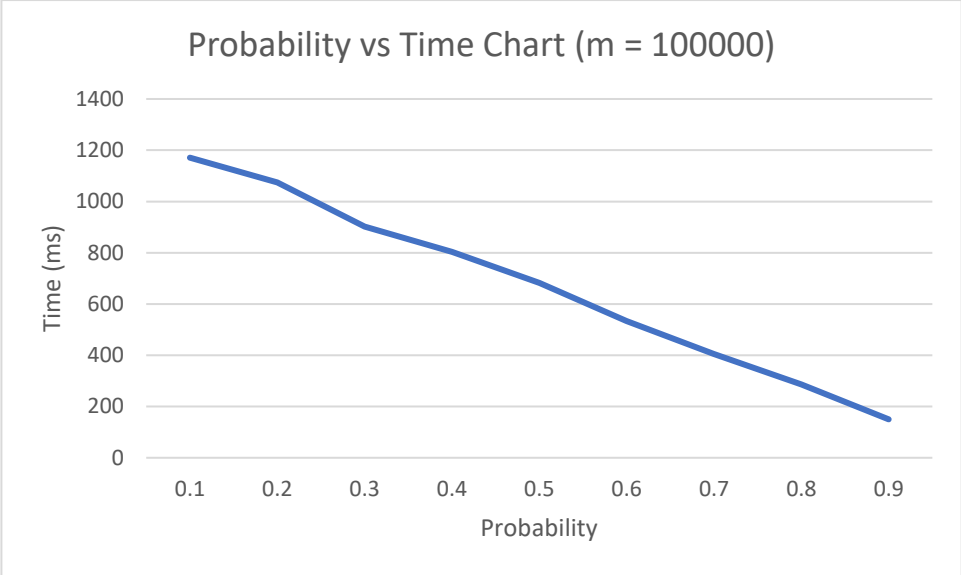
$T(n) = \log(n!)$

$\log(1) + \log(2) + \ldots \log(n) \leq \log(n) + \log(n) + \log(n) \ldots$ (n times)

$\log(n!) \leq n \log n$

nlogn is the theoretical upper bound for run time of the program. The chart above shows the relation between run time of the program for different values and value of NlogN for the same values

3)



Effect of p choice in run time

For small values of p, probability of operation being an addition is larger. As a result, size of the heap (N) gets larger and that causes computing times (N * logN) to become larger. Also, since the decrease in the decreaseKey() function is quite small, the number of swaps to get the updated value to the correct position is not likely to be as large as the insert() operation. This also contributes to the decrease in computing time as probability of operation being an update increases.