Enes Bedirhan Dikmen

150180016

BLG 335E – Analysis of Algorithms I Fall2020 Homework 2

Part 1.

My code can be compiled with:

g++ -std=c++11 -Wall -Werror main.cpp

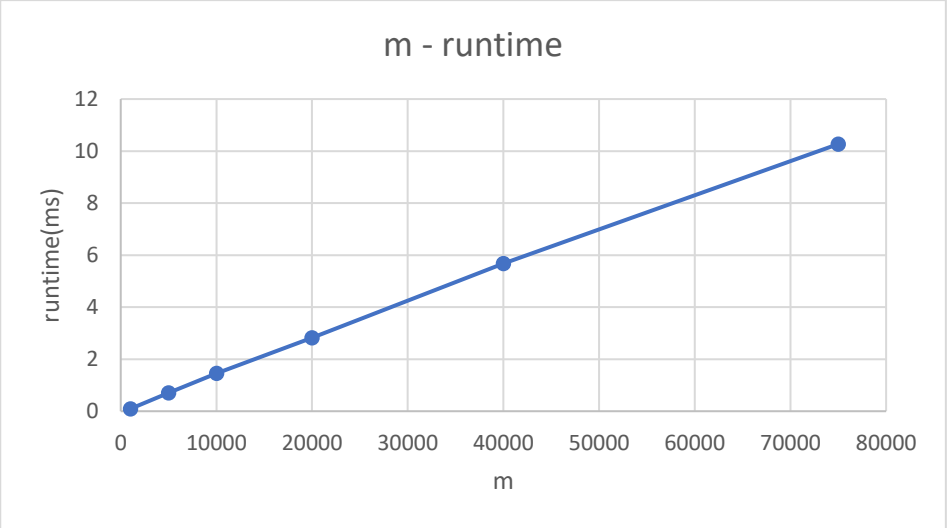
And can be run with:

./a.out m p

Part 2.

1. In my implementation for each operation either an update or an insertion operation takes place. Insert operation's theoratical run time is O(logn) since it contains a decrease-key operatin with O(logn) plus a constant number. Update operation is almost the same with decrease-key operation, just decreases with a constant 0,01.So it's run-time is O(logn) too. This means both operations run-time's are aproximately equal. So total runtime should follow a behaviour like:
   m.logn
   Considering that n's maximum value will be m.(1 – p) – (m/100). Runtime will be close to:
   O(mlogm), for small p values.

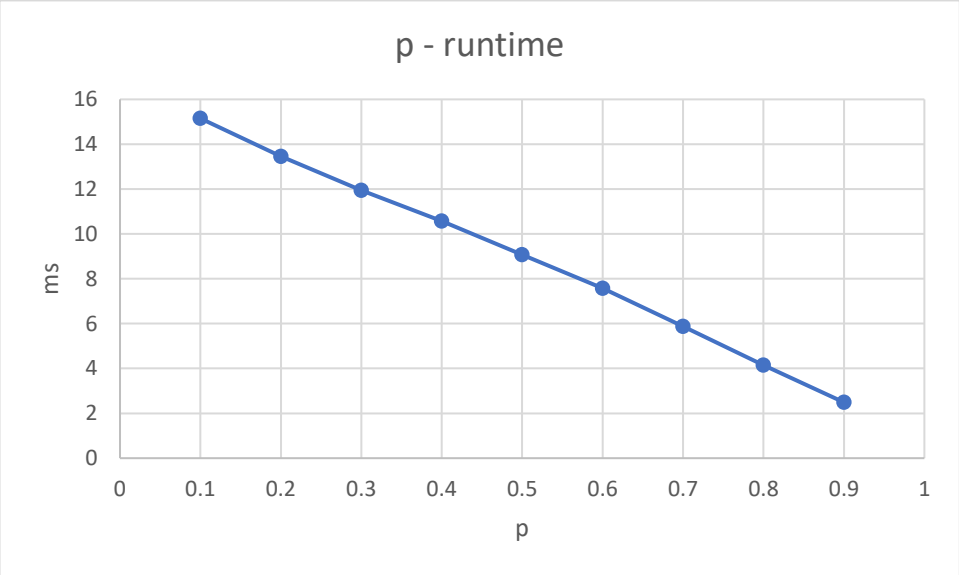
2.



m	milliseconds
1000	0.09
5000	0.7
10000	1.45
20000	2.82
40000	5.68
75000	10.27
100000	13.46

Yes, as the graph above shows runtime values show a similar pattern to a nlog(n) graph. For considerably low value of p=0.2 runtime follow O(mlogm) bound as I have mentioned at Part2.1

3. I have choosen m = 100000



p	milliseconds
0.1	15.15
0.2	13.46
0.3	11.95
0.4	10.57
0.5	9.07
0.6	7.58
0.7	5.88
0.8	4.15
0.9	2.49

Yes running time is affected by p as we can se in the graph. Because value of p determines the total number of additions to the heap. And this determines maximum size of heap. As explained at Part2.1 both update and addition operations runs in O(logn) time but depending on p, magnitude of n changes. This means in equation mlogn, as p increases n gets smaller and total runtime decreases. So the linear relationship between p and total additions leads to the linear relationship that shown in the graph between p and runtime.