# Istanbul Technical University

## Analysis of Algorithms I

BLG 335E

Taxi Simulation with Priority Queue

---

# Homework 2

---

Tevfik Özgü

150180082

19 December 2020

# CONTENT OF TABLE

# FIGURES

# TABLES

# EQUATIONS

# 1 INTRODUCTION

In this homework a taxi application simulation will be done. In this simulation there will be a priority queue with the binary heap data structure. When this application used, hotel desk clerk will be able to see which restaurant is nearest in the city to the hotel location. When a customer is leaving the hotel, hotel desk clerk will call the nearest taxi thanks to this application.

In this report, a dataset named *'locations.txt'* is used. There are locations of restaurants which are in form of longitude and latitude.

In this application. it is a probability that a new taxi can be available in the city. This property will be handled by adding a new taxi to hotel. Also, any taxi's distance can be updated. When it is wanted to update a taxi distance, its distance will be subtracted by 0.01 meter. Also, when clerk desk calls the taxi, this taxi will be deleted from the simulation.

Update and add properties will belong to a probability. This probability will be taken from user in the command window. Probability of updating will be p and adding will be 1-p. Also, when 99 operations are done with adding or updating, next operation will be deleting namely calling the nearest taxi.

At the end of the program the distance of called taxis, the number of taxi additions and distance updates and total running time in milliseconds will be given.

In the first part of this report, functions of the application will be explained. These functions will be tried to explain briefly by using flowchart.

In the second part, upper bound of each function namely operations will be found. These upper bounds will be tried to explain on the code snippets. After that, theoretical running time of simulation will be evaluated and simulation running times will be taken from application. There will be given a plot which belongs to number of operation - running times. These results will be compared with theoretical results and check whether they prove each other or not. At the end of the report, different probability values will be given and check whether running time effected or not. Also, by using these results, it will be tried to explain why or not effected from probability value.

# 2 PART 1

In this part code of the applications will be explained. Hotel location is selected as 33.40819 (longitude) and 39.190001 (latitude). At the starting point of application, probability of updating and number of operations are taken from command window.

## 2.1 Operations

When the next operation is 100th operation, nearest taxi will be called. If this condition is not hold, update or adding operations will be done according to probability value. Program starts from the main function. In this function delete, update or delete conditions are checked and applied after calculating probability. If operation is adding operation, new location is read from file and added to locations_array. Then this array is heapified again. When update operation is called, random index is updated and again this array is heapified. Only difference is that there is an adding in the add operation. When deleting a taxi location this location is saved to deleted taxis location array. Main function flowchart is given at Figure 1.

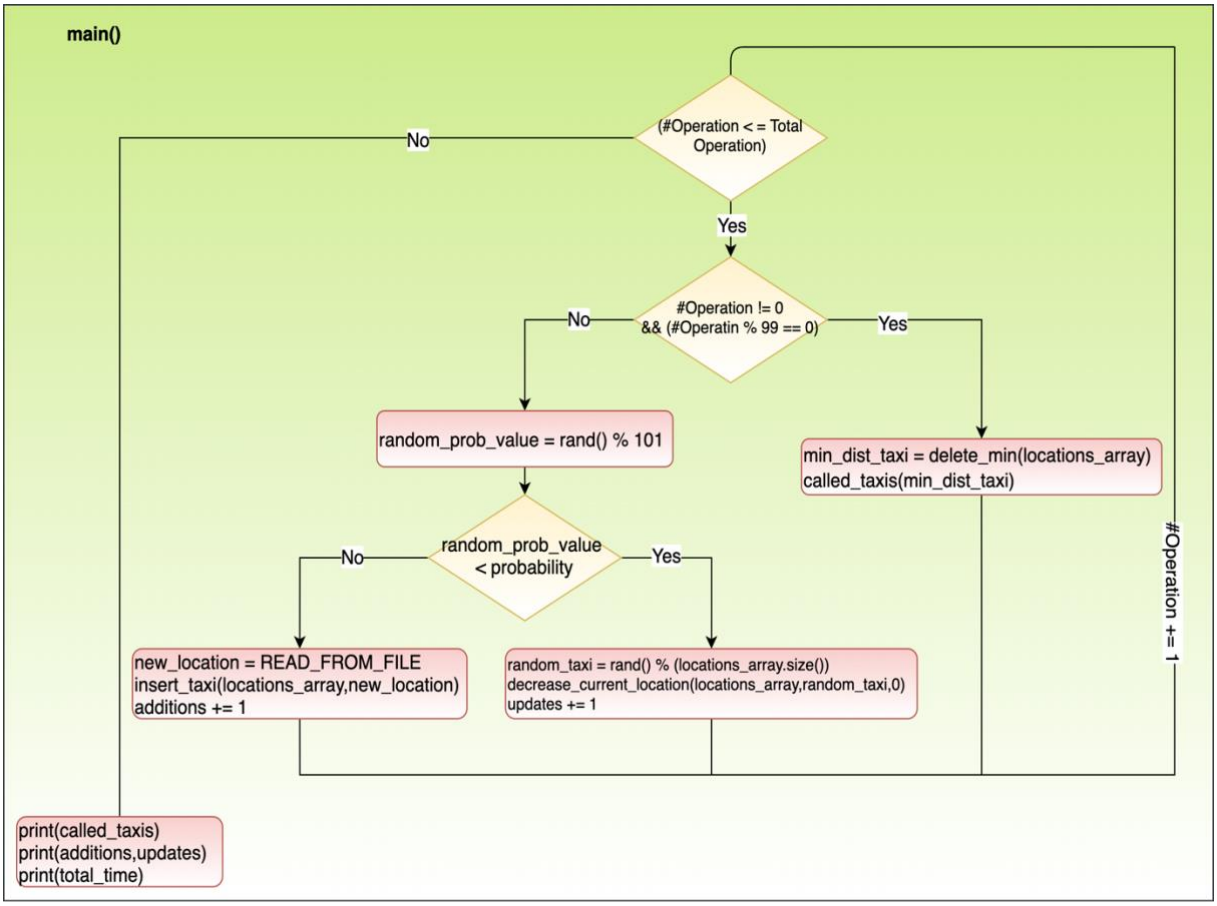


*Figure 1. Main Function Flowchart*

## 2.2 Insert Taxi Function

In this function new taxi is pushed back to the end of the array and decrease current location function is called with mode of adding (0).

## 2.3 Decrease Current Location Function

In this function initially if mode is update, distance is decremented by 0.01. Then the selected index is compared with its parent. If it is lower than its parent, they are swapped and after this point on it compared between its new parent consequently. This functions flowchart is given at Figure 2.

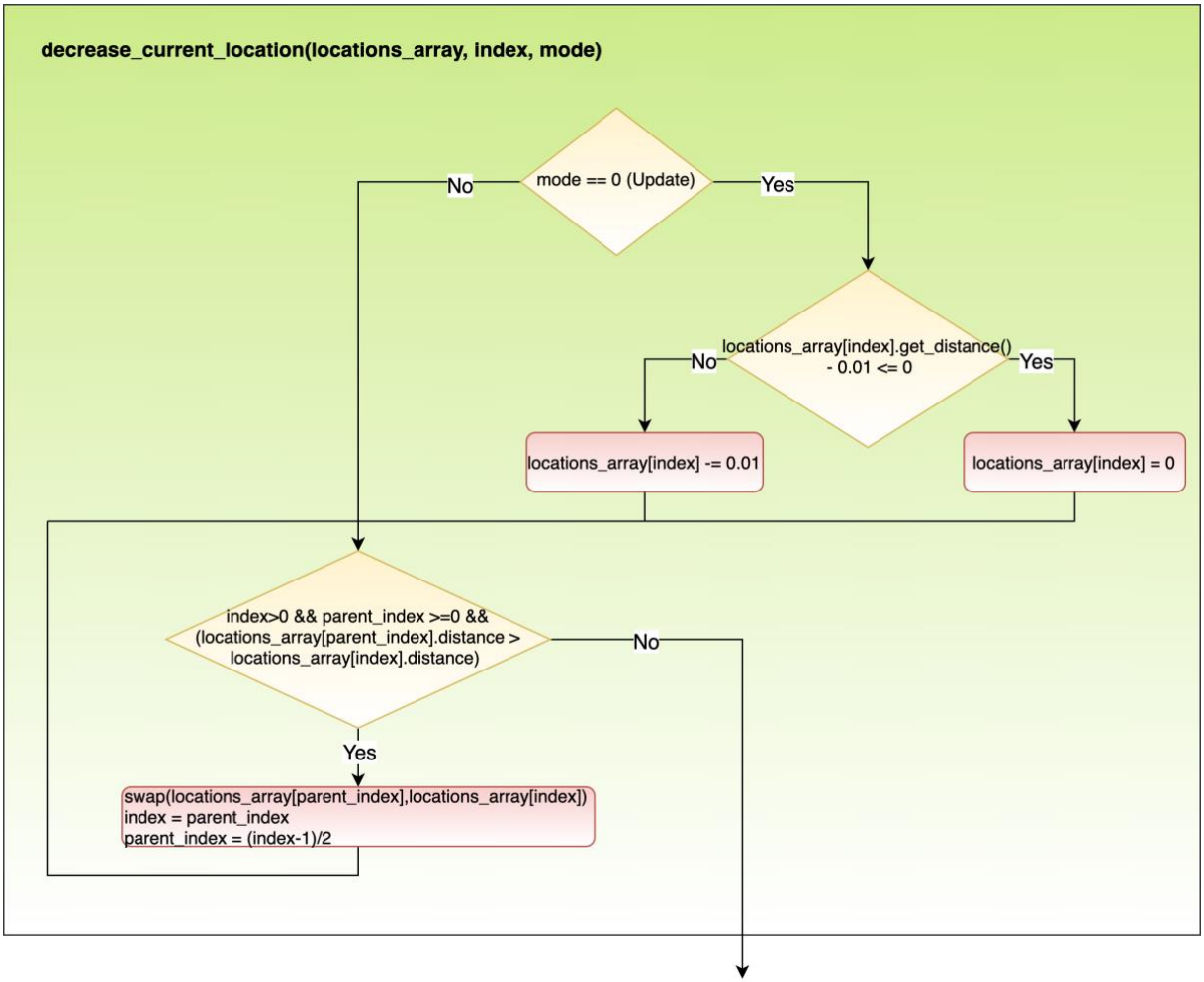


*Figure 2. Decrease Current Location Flowchart*

## 2.4 Delete Minimum Function

In this function initially checked whether array has an element or not. If there is no element function program is terminated. If there is an element, last element is put into the initial point of element and last element is deleted. From this point on array is heapified again. Flowchart of delete minimum function is given at Figure 3.

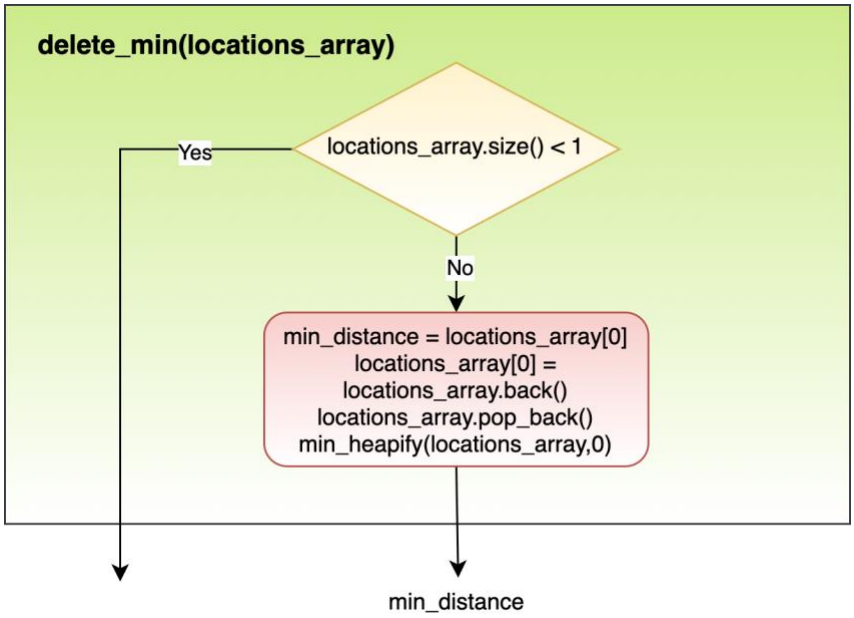


*Figure 3. Delete Minimum Function*

## 2.5 Minimum Heapify Function

In this function initially minimum index is selected as the coming index. Then left index and right index of this element is found. Initially current distance is compared with left child of this element. If it is higher than its left child, left child is compared with right child to understand which one is lower. If this index is lower than left child, it is compared with right child. After these operations, smallest indexed element between left, right and parent is found.

When found the minimum indexed number, it is checked whether this element is lower than left or right child. If it is lower than its children', they are swapped and recursive function is called back. From this point on function is proceed with lowest indexed value. At the end of the function lowest element has become the first element of locations array. Flowchart of this function is given at Figure 4..

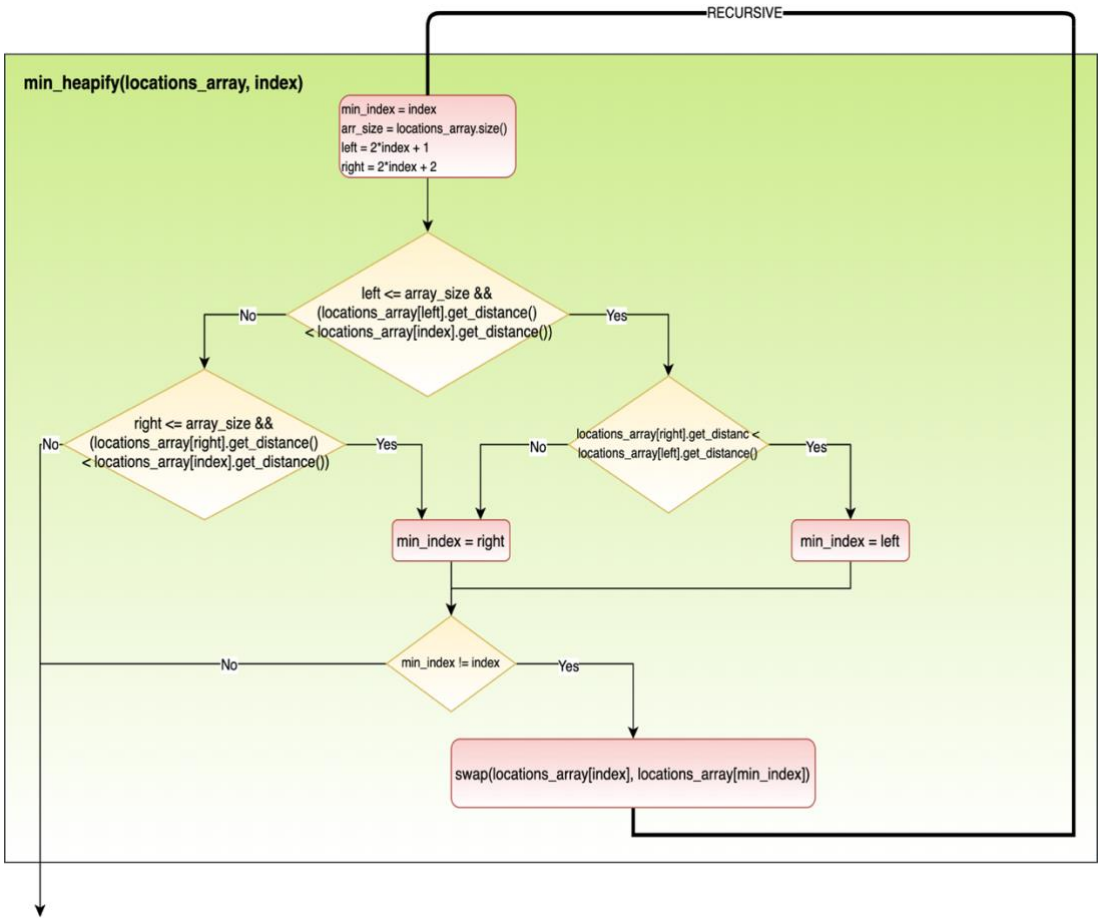


*Figure 4. Minimum Heapify Function*

At the end of the function, called taxis' locations, total addition and update operations and total amount of time passed are printed.

# 3 PART 2

In the first section of this part, running times will be evaluated. After that, effect of the number of operations will be shown and results will be demonstrated. These results will be compared with the theoretical results. Then in the last section effect of the probability value on the running time will be discussed whether it is affected or not.

## 3.1 Running Times

For evaluation of running times, every function's code will be given and some calculations will be done. Initially delete operation will be explained.

### 3.1.1 Delete Function Running Time

As explained earlier, in the delete function, last element is being carried to first index of array. Then minimum heapify function is called. Since there is only deletion operation in delete function, calculations will be done only based on minimum heapify function. This function is given at Table 1.

```
1 void min_heapify(vector<Locations> &inc_vec, int index){
2     int left, right,min_index=index;
3     int arr_size = inc_vec.size();
4     left = 2*index+1;
5     right = 2*index+2;
6
7     if (left <= arr_size && inc_vec[left].get_distance() < inc_vec[index].get_distance()){
8         if (inc_vec[right].get_distance() < inc_vec[left].get_distance()){
9             min_index = right;
10        } else {
11            min_index = left;
12        }
13
14    }
15    else if (right <= arr_size && inc_vec[right].get_distance() < inc_vec[index].get_distance()){
16        min_index = right;
17    }
18
19    if (min_index != index){
20        swap(inc_vec[index],inc_vec[min_index]);
21        min_heapify(inc_vec,min_index);
22    }
23 }
```

*Table 1. Minimum Heapify Function Codes*

In this function, from line 1 to 20 there is only one operation. No recursive or no loop is concurred. So, their running times can be thought at $\Theta(1)$. Thinks will be change at line 21 since it is recursive function. Actually, how many times this function will be called is not known. So, there will be make some assumptions. Since purpose is to find the upper limit, every time operations will be supposed as being on the left side of array and tree is not complete binary tree, elements are stacked onto the left side of array. This scenario is simulated at Figure 5.

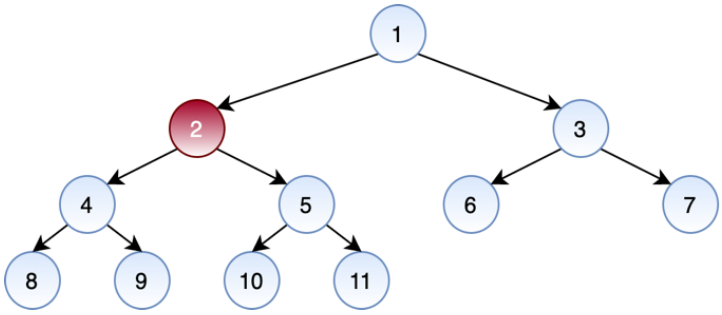


*Figure 5. Worst Case Recursive Function*

As show at Figure 5, more child is located on the left side of tree. 7 out of 11 elements are located on the left side of the tree. So, in general it is said that 2n/3 elements are involved. From these calculations on upper case can be described as in Equation 1.

$$\Theta(n) \leq \Theta(2n/3) + \Theta(1)$$

*Equation 1. Upper Bound of Minimum Heapify Algorithm*

In this equation a=1 and b = 3/2, k=0. Since a = $b^k = 1$ then upper bound of min heapify which is T(n) = $\Theta(\log n)$.

### 3.1.2 Addition and Update Worst Case Running Times

In the addition operation, as explained before only difference was that an extra element to the last is being added to the array. So, these functions upper bounds will be same each other. These both functions were calling decrease key function which is given at Table 2.

```
1 void decrease_current_location(vector<Locations> &dec_vec, int index,int mode){
2     if (mode == 0){
3         if (dec_vec[index].get_distance() - 0.01 <= 0){
4             dec_vec[index].set_distance(0);
5         } else {
6             dec_vec[index].set_distance(dec_vec[index].get_distance() - 0.01);
7         }
8     }
9     int parent_index = 0;
10    double parent_index_double = (index-1)/2;
11    if (parent_index_double < 0){
12        parent_index = 0;
13    } else {
14        parent_index = floor(parent_index_double);
15    }
16    while((index > 0) && (parent_index >= 0) && (dec_vec[parent_index].get_distance() > dec_vec[index].get_distance())){
17        swap(dec_vec[index],dec_vec[parent_index]);
18        index = parent_index;
19        parent_index = (index-1)/2;
20    }
21 }
```

*Table 2. Decrease Key Function Codes*

In this function there is only one loop which is at line 16. So, it can be said that only this line specifies the upper limit since rest of them are Θ(1). In this line which is 16, it is clear that worst case happens when current index is the last element of the tree. From reaching lower element to the root, there must be a number of operations which is equal to depth of the tree. Depth of the tree is lgn as usual. So, upper bound of addition and update operations are Θ(lgn).

## 3.2 Effect of Operation Amount

In this section, effects of operation amount on running time will be discussed. Operation amount will be changed from 1000 to 100K. For all simulations, probability value will be selected as 0.2. After finding running times, plot will be given. These results will be compared with the theoretical expectations. Now general equation will be found. As explained before delete operation takes logn, update and adding operations take lgn times. The probability that is the operation is adding is 0.8 as said before. So, in general in 100 operation there will be 1 deletion, 80 adding and approximately 20 update operations will be done. This means that there will be n/100 deletion, 80n/100 adding and 19n/100 update operations. In this case upper bound of all operations can be expressed as in Equation 2.

$$T(n) = \frac{80n}{100}\Theta(\lg n) + \frac{19n}{100}\Theta(\lg n) + \frac{n}{100}\Theta(\log n) = \frac{99n}{100}\Theta(\lg n) + \frac{n}{100}\Theta(\log n)$$

*Equation 2. Upper Bound of Taxi Simulation Application*

When simulation is done, running times that was handled is given at Table 3.

| Operation Amount | Running Times (ms) |
|---|---|
| **1000** | 1.420 |
| **5000** | 6.71 |
| **10000** | 12.78 |
| **20000** | 24.819 |
| **50000** | 58.917 |
| **75000** | 80.927 |
| **100000** | 106.911 |

*Table 3. Running Times with p=0.2*

From these calculations, running time plot is drawn. This plot is given at Figure 6.

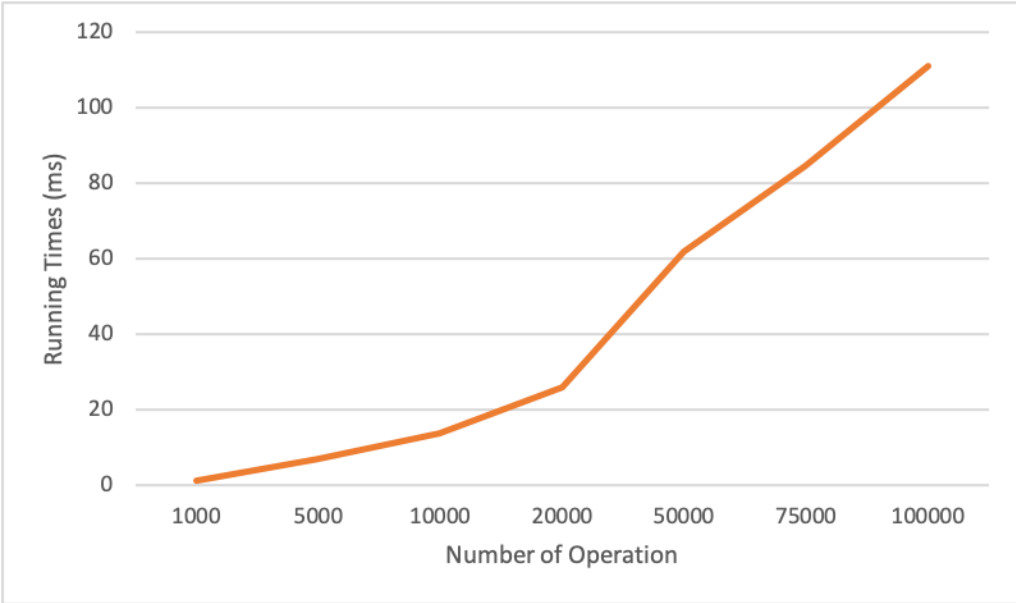


*Figure 6. Running Time Plot*

This is the graph that was expected by comparing with the equation at Equation 2. When same calculations are done with same number of operation and with the theoretical value which is given at Equation 2, comparison table is given at Table 4.

| **Operation Amount** | **Running Times (ms)** | **T(n)** | $\frac{T(n)}{\text{Running Time}}$ |
|---|---|---|---|
| **1000** | 1.420 | 9896.126442 | 6969 |
| **5000** | 6.71 | 61009.12478 | 9102 |
| **10000** | 12.78 | 131948.3526 | 10324 |
| **20000** | 24.819 | 283756.9111 | 11440 |
| **50000** | 58.917 | 775026.6885 | 13101 |
| **75000** | 80.927 | 1206105.567 | 14800 |
| **100000** | 106.911 | 1649354.407 | 15420 |

*Table 4. Ratio Simulation*

It is understandable that ratios between different operation amounts are close to each other. Since 1000 and 5000 is very low, they are slightly different but it is not problem at all. For the higher operation amounts, it must not be forgotten that there is a probability according to p value. So, these deviations is normal I think. So, it can be said that theoretical and experimental results are proving each other.

## 3.3 Effect of Probability Value

In this section of part 2, effect of p value on the running time will be observed. Different p values from 0.1 to 0.9 will be given and operation amount value which is called m will be 75000 as a constant. When inputs are given, handled running times are given at Table 5.

| Probability Value | Running Times (ms) |
|---|---|
| **0.1** | 90.2 |
| **0.2** | 82.1 |
| **0.3** | 75.5 |
| **0.4** | 66.4 |
| **0.5** | 60.1 |
| **0.6** | 49.5 |
| **0.7** | 40.3 |
| **0.8** | 28.55 |
| **0.9** | 18.95 |

*Table 5. Effect of p Value to Running Time*

When these values are put into the graph it is clear that plotting is like at Figure 7.

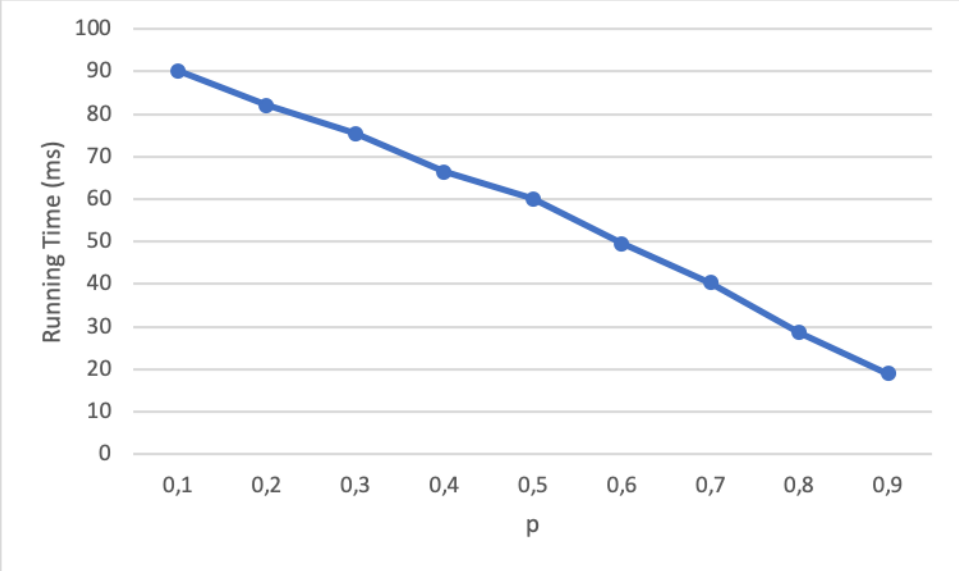


*Figure 7. p-Running Time Graph*

### 3.3.1 Effected or not Effected?

So, it is understandable that running time **DECREASES** while probability value is **INCREASING.** It can be said that probability value **effects** running time.

### 3.3.2 Reason Why Effected?

When probability value increases, it is clear that there becomes more update operation so adding operation decreases. When there is less adding operation, tree becomes having **less depth** so namely n is **decreasing**. In other words, since tree depth is **decreasing**, running time **decreases** too.