Department of Computer Engineering

BLG 335E
Analysis of Algorithms I
Project 2 Report

Fall 2018

Muhammed Raşit EROL

150150023

1 INTRODUCTION

In this project, it required that analysing the performance of a call centre's employees. For this aim, heap sort is used in this project in order to determine best/worst performance and maximum/minimum call numbers for each day.

Moreover, there is an input file called numbers.csv in our second part of project. In the second part, it is required that sorting 2 billion numbers in 10 step using extract_max() function.

In my project, there is one class called Heap which can perform both for call center employees and sorting numbers.csv.

2 IMPLEMENTATION

In this project, main class is called Heap which contains all heapsort function. In the heap class, there are two dynamic array. One is arr and is used for storing data. All heapsort functions are applied on this array; furthermore, there is an id_arr which store the id of the employees with the parallel index of the arr array. Hence, when value in arr array is swap with another index in the array, id of employees is also swap with index which is used in the previous swap. Shorty, swaps on arr is also is applied on id_arr.

For more flexible use of array, dynamic array structure is used in this project. Constructor can take parameter for determining of array size. Also, in the default constructor, initial size is set to 100. After the creating Heap objects, user can add elements which number is more than the array length. In that moment, array capacity is doubled and all elements are copied to new array. In the class, there is check_capacity() function which checks the capacity of array when adding new elements to the array.

In the class, there is heap_size variable which stores the heap number on the array and length stores added elements on the array. Heap_size is sensitive variable because methods works until heap_size. Hence, it used as private variable. Necessary getter and setter function is added to class for heap_size.

When adding new element on the array, there are two methods. One uses max/min_heap_insert() and other is direct assigning like arr[i]=value. In the second one, length must increase by hand and capacity must be checked using check_capacity() function.

In the first part of the project first day all employees are added to array. After first day, each employee is updated by searching corresponding id and updated value by using increase_key() function. If user is not in the array then that id and value is added to array. In the reading part increase_key() and insert() functions are used. For the first part, call_center() function is called in the main in order to execute days.

In the second part of the project, all numbers are read from file and build_max_heap() function is called. There are 10 step in which 200000 elements of array is extract using extract_max() function. Hence, totally 2 billion number is sorted using build_max_heap() and extract_max() functions. For the first part, number_analysis () function is called in the main in order to sorting numbers.

All function are explained below:

- **void max/min_heapify(int index)**
  This recursive function takes parameter which is called index. This index value shows us to which index will be heapifyed. Meanly, it finds the correct position of value for corresponding index. The function looks the value of both right and left child and if value bigger/smaller than index than swap operation is applied for them. After that, max/min_heapify is called for smallest or biggest element which is swaped. In that moment, ids are also swapped. The recursive part is end when left or right index is smaller than the heap_size or biggest/smallest element is equal to value of index of array. The function does not have return value.

- **void build_max/min_heap()**
  After this function is called, array is built which root element contains biggest/smallest element of array. In the function, max/min_heapfy is called for each nodes which have child. Hence, biggest or smallest element is stored in the root after function call. The function does not have return value.

- **void ascending/descending _heapsort()**
  This function calls the build_max/min_heap() function in order to build max or min heap. After that value of root is putted to the end of the array and heap_size is decreased. Hence, biggest/smallest element is putted to the end, after that value in root is placed correct position in the heap. This is made by max/min_heapify() function. End of the function call, ascending or descending array is created. The function does not have return value.

- **void max/min_heap_insert(int key, int id)**
  This function is used when adding new element on the array. If capacity is not enough for adding new element, array size is doubled and all elements are copied to the new array. If capacity is enough, maximum or minimum value of integer is assign to the new last element of the array. After that, heap_increase/decrease_key() function is called for updating the key in given index with the given id. With this function given key is placed correct position on the array with the id. The function does not have return value.

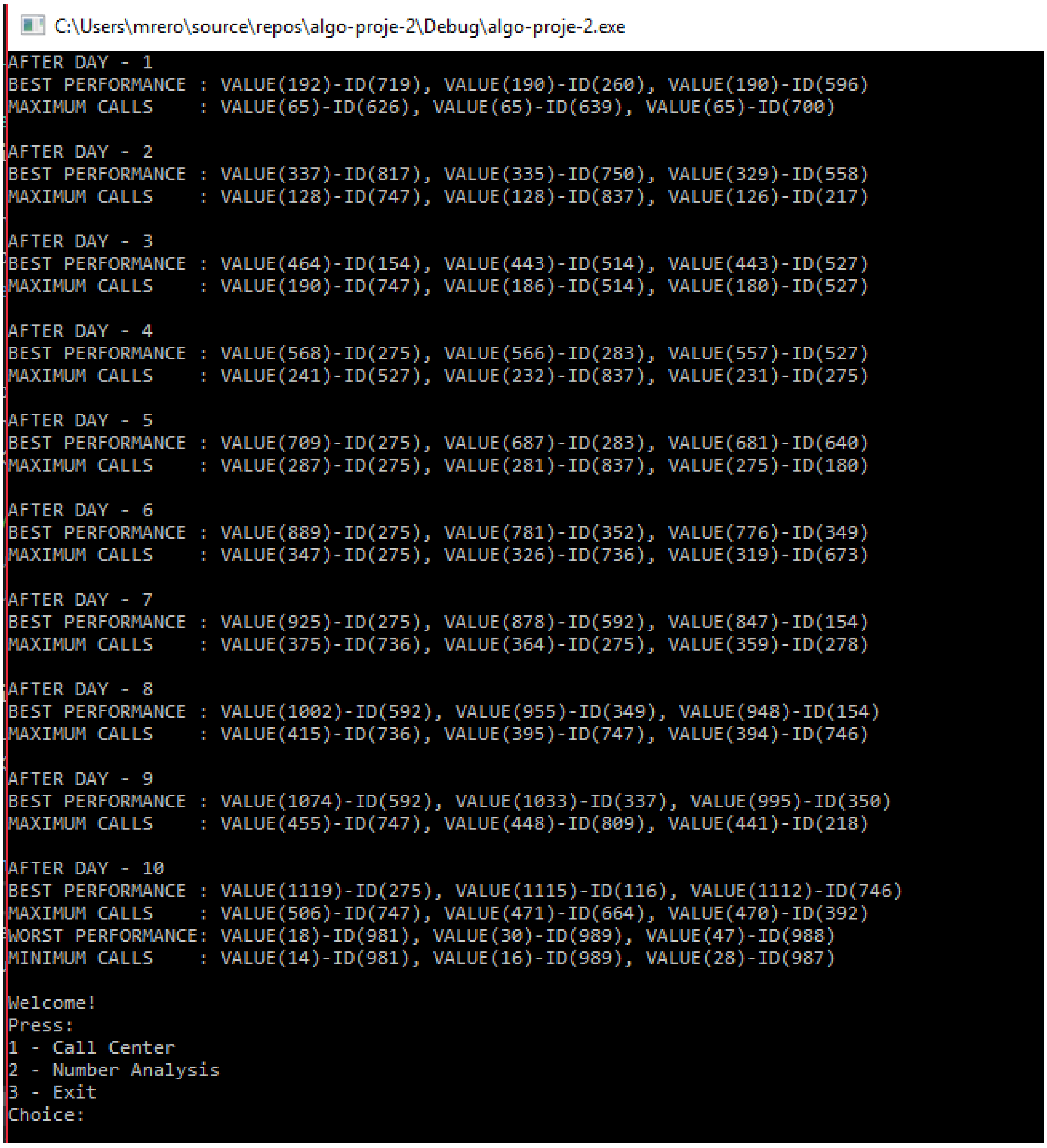
- **int heap_extract_max/min (char which_arr)**
  This function returns the root of the array and takes the last element of array and puts it into root. Then max/min_heapify() function is called for placing that element into correct position. Also, the function is take parameter for the determining which array is returned with root. In this function, length is also decreased.

- **void heap_increase/decrease_key(int index, int key, int id)**
  This function is used when updating the array elements. heap_increase key() is used on max heap and heap_decrease_key() is used on min heap. It check all value above the given index and swap if it is bigegr/smaller than itself. It does that also for id_arr. The function does not have return value.

All other functions and variables are explained in the code.

Output of the first part is shown below:

The result is also written in the “day_analysis.txt”.



```
C:\Users\mrero\source\repos\algo-proje-2\Debug\algo-proje-2.exe
AFTER DAY - 1
BEST PERFORMANCE : VALUE(192)-ID(719), VALUE(190)-ID(260), VALUE(190)-ID(596)
MAXIMUM CALLS    : VALUE(65)-ID(626), VALUE(65)-ID(639), VALUE(65)-ID(700)

AFTER DAY - 2
BEST PERFORMANCE : VALUE(337)-ID(817), VALUE(335)-ID(750), VALUE(329)-ID(558)
MAXIMUM CALLS    : VALUE(128)-ID(747), VALUE(128)-ID(837), VALUE(126)-ID(217)

AFTER DAY - 3
BEST PERFORMANCE : VALUE(464)-ID(154), VALUE(443)-ID(514), VALUE(443)-ID(527)
MAXIMUM CALLS    : VALUE(190)-ID(747), VALUE(186)-ID(514), VALUE(180)-ID(527)

AFTER DAY - 4
BEST PERFORMANCE : VALUE(568)-ID(275), VALUE(566)-ID(283), VALUE(557)-ID(527)
MAXIMUM CALLS    : VALUE(241)-ID(527), VALUE(232)-ID(837), VALUE(231)-ID(275)

AFTER DAY - 5
BEST PERFORMANCE : VALUE(709)-ID(275), VALUE(687)-ID(283), VALUE(681)-ID(640)
MAXIMUM CALLS    : VALUE(287)-ID(275), VALUE(281)-ID(837), VALUE(275)-ID(180)

AFTER DAY - 6
BEST PERFORMANCE : VALUE(889)-ID(275), VALUE(781)-ID(352), VALUE(776)-ID(349)
MAXIMUM CALLS    : VALUE(347)-ID(275), VALUE(326)-ID(736), VALUE(319)-ID(673)

AFTER DAY - 7
BEST PERFORMANCE : VALUE(925)-ID(275), VALUE(878)-ID(592), VALUE(847)-ID(154)
MAXIMUM CALLS    : VALUE(375)-ID(736), VALUE(364)-ID(275), VALUE(359)-ID(278)

AFTER DAY - 8
BEST PERFORMANCE : VALUE(1002)-ID(592), VALUE(955)-ID(349), VALUE(948)-ID(154)
MAXIMUM CALLS    : VALUE(415)-ID(736), VALUE(395)-ID(747), VALUE(394)-ID(746)

AFTER DAY - 9
BEST PERFORMANCE : VALUE(1074)-ID(592), VALUE(1033)-ID(337), VALUE(995)-ID(350)
MAXIMUM CALLS    : VALUE(455)-ID(747), VALUE(448)-ID(809), VALUE(441)-ID(218)

AFTER DAY - 10
BEST PERFORMANCE : VALUE(1119)-ID(275), VALUE(1115)-ID(116), VALUE(1112)-ID(746)
MAXIMUM CALLS    : VALUE(506)-ID(747), VALUE(471)-ID(664), VALUE(470)-ID(392)
WORST PERFORMANCE: VALUE(18)-ID(981), VALUE(30)-ID(989), VALUE(47)-ID(988)
MINIMUM CALLS    : VALUE(14)-ID(981), VALUE(16)-ID(989), VALUE(28)-ID(987)

Welcome!
Press:
1 - Call Center
2 - Number Analysis
3 - Exit
Choice:
```

Figure 1 Output of first part

The result is also written in the “day_analysis.txt”.

3 ANALYSIS

3.1 ANALYSIS OF PROCEDURES

For heapsort there are several functions which are shown below. All running time of functions is explained below:

- **void max/min_heapify(int index)**

  This function maintains the heap property. When we look the code, there is no loop in this function and number of step do not depend N. Hence, running time of this function until last line which calls the max/min_heapify() again. The recursive call implies a recurrence relation. Except max/min_heapify() line cost is Θ (1). In the case of full binary tree, about half of the tree might be involved. In a complete binary tree with 11 nodes, 7 of those nodes are leaves at the bottom level. In general, ≤ 2/3rds of the tree might be involved in the worst case. So worst case is when the last row of the tree is half full on the left side and arr[i] is their ancestor. The subtrees of the children of our current node have size at most 2n/3.
  The running time of max/min_heapify() can be described by the recurrence:
  $T(n) \leq T(2n/3) + \Theta(1)$
  This is Case 2 by the master method, so:
  $T(n) = O(\lg n)$

  We could also describe the running time of max/min_heapify() for a node of height h as O(h).

- **void build_max/min_heap()**

  This function builds a heap from an unordered array. In this function there is loop which works until N and in the loop max/min_heapify() function is called. Hence, the running time of max/min_heapify() is O(lg n) which it is in the loop works until n, then the running time of build_max/min_heap() is O(n lg n).

- **void ascending/descending _heapsort()**

  This function sorts an array in place. In this function build_max/min_heap and max/min_heapify is called. Other line's running time is $\Theta(1)$. Total running time of the function is O(n lg n).

  Shorty, build_max/min_heap() takes O(n). We have a loop which each of the n-1 calls to max/min_heapify takes O(lg n) time. Total time is O(n lg n).

- **void max/min_heap_insert(int key, int id)**

  This function insert a new element. In this function, there is no loop; however, heap_increase/decrease_key() is called. Cost is: c1*O(1) + c2*O(lg n), or just O(lg n). Due to the implementation of the project, in the insert function, capacity checks perform with O(n) complexity. But, in the sorting side, it is not affect our performance.

- **int heap_extract_max/min (char which_arr)**

  This function selects max/min element. In this function, there is no loop; however, max/min_heapify() is called. Other term's running time is O(1). With the max/min_heapify() which running time is O(lg n), total running time of this function is O(lg n).

- **void heap_increase/decrease_key(int index, int key, int id)**
  This function updates the key on the array. In this function there is a loop which works depth of the tree times. The depth of a binary tree is O(lg n). We do a constant amount of work in the loop. Cost is: c1*O(1) + c2*O(lg n), or just O(lg n).

- **int heap_maximum/minimum(char which_arr)**
  This function returns the root element of array. Runs in Θ(1) time.

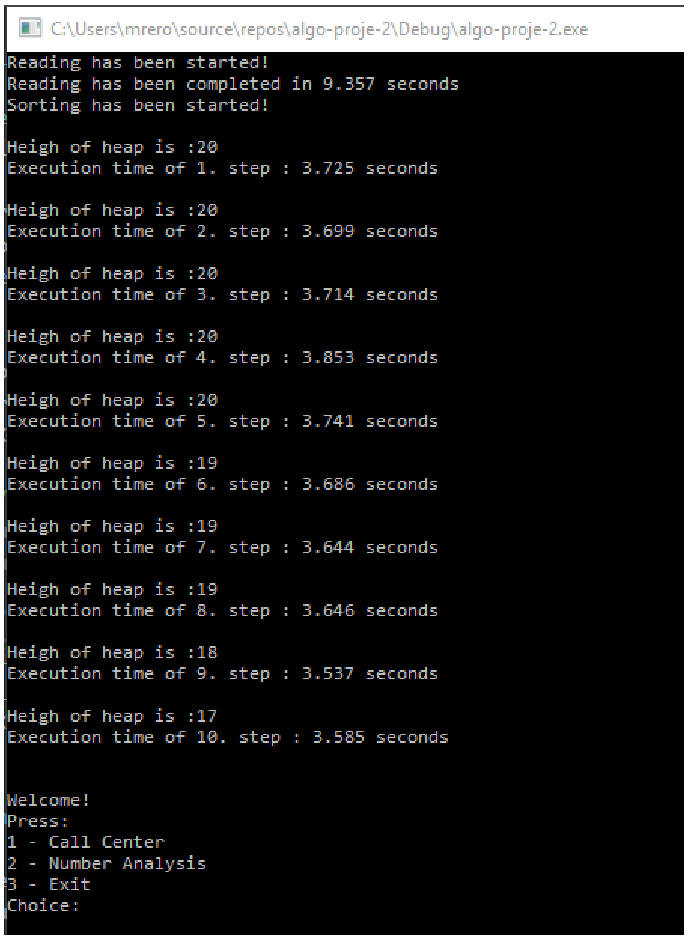
- **int Parent/Left/Right(int index)**
  This function calculates the index of parent, left child or right child of given index and returns it. Runs in Θ(1) time.

- **void swap(int*, int*)**
  This function swaps the given values. Runs in Θ(1) time.

3.2 Number Analysis

In this part, 2 billion number is sorted using Heap class mentioned above. Output of the second part is shown below:

The sorted numbers are also written in the "numbers_sorted.csv".



```
C:\Users\mrero\source\repos\algo-proje-2\Debug\algo-proje-2.exe
Reading has been started!
Reading has been completed in 9.357 seconds
Sorting has been started!

Heigh of heap is :20
Execution time of 1. step : 3.725 seconds

Heigh of heap is :20
Execution time of 2. step : 3.699 seconds

Heigh of heap is :20
Execution time of 3. step : 3.714 seconds

Heigh of heap is :20
Execution time of 4. step : 3.853 seconds

Heigh of heap is :20
Execution time of 5. step : 3.741 seconds

Heigh of heap is :19
Execution time of 6. step : 3.686 seconds

Heigh of heap is :19
Execution time of 7. step : 3.644 seconds

Heigh of heap is :19
Execution time of 8. step : 3.646 seconds

Heigh of heap is :18
Execution time of 9. step : 3.537 seconds

Heigh of heap is :17
Execution time of 10. step : 3.585 seconds


Welcome!
Press:
1 - Call Center
2 - Number Analysis
3 - Exit
Choice:
```

Figure 2 Output of second part

In this section we have an input file which consists of two million rows. We have sort the numbers in “numbers.csv” file in descending order then we have created a max-heap, and at each step, you will extract the largest 200000 numbers from the heap.

We have for loop which works ten times. In this loop, before the extracting element (it happens in inner loop) we calculate the height with this formula: ceil((log2(n + 1))) – 1.

Heights and execution time can be seen in Figure 2. In extraction function, max heapify function is called and the complexity of the extract max function is depend max/min_heapify function. In our lecture slide, there is a definition like this: “We could also describe the running time of max/min_heapify() for a node of height h as O(h). “ As can be seen in Figure 2, time is reduced logarithmically.

4 COMPILING ON SSH SERVER

- g++ *.cpp -o algo -std=c++11 –lm

In order to proper compiling use that statement.