DSA Assignment: Assignment 1 – PS1

1)Kumar Abhimanyu(2021sp93079)

2)Prateek Ram(2021sp93032)

3)Rishabh Sharma(2021sp93019)

4)Samee Sheikh(2021sp93020)

Input/Output File Description: -

The input mainly consists of 2 Files: [inputPS1a.txt, inputPS1b.txt]

- inputPS1a.txt : - It consists of the initial input given by the user at the start of the program. It contains the patient records consisting of name and age separated with comma. Each new line consists of new patient record.

- inputPS1b.txt : - Thereafter, new patients will be input through another file. It contains the data of new patient in the form of '*newPatient: [patient_name], [patient_age]*'. It also consists of how many patients records we need to display who are next for testing in the form of '*nextPatient: [No. of Patients]*'.

The output consists of a single File: [outputPS1.txt]

- outputPS1.txt: - This file consists of list of registered patients and, also if a new patient gets added after a while, then it gives the new refreshed list of patients. Also, it contains the records of next 'N' patients to be next for testing.

Sample Input inputPS1a.txt:

Surya, 60
Ajay, 54
Rishi, 57

Sample Input inputPS1b.txt:

newPatient: John, 55
nextPatient: 1

Sample Output outputPS1.txt

---- registered Patient ----------
No of patients added: 3
Refreshed queue:
100160, Surya
100357, Rishi
100254, Ajay

---- new patient entered----------
Patient details: John, 55, 100455
Refreshed queue:

100260, Surya
100357, Rishi
100455, John
100554, Ajay

---- next patient: 1 ----------
Next patient for testing is: 100260, Surya

Actual Input inputPS1a.txt :

Surya, 60
Ajay, 54
Rishi, 57
Ajay, bca
Zane, -99
Samee ,
Brooks, 99
Cole, 13
David, 12
Evan, 50

Actual Input inputPS1b.txt :

newPatient: John, 55
newPatient: Sam ,
nextPatient: 1
fgsdhqjkdbkew
newPatient:Abhimanyu,67
nextPatient: 14

Actual Output outputPS1.txt

---- registered Patient ----------

No of patients added: 7
Refreshed Queue:
000399, Brooks
000060, Surya
000257, Rishi

000154, Ajay
000650, Evan
000413, Cole
000512, David

---

3 Invalid Input
Ajay, bca
Zane, -99
Samee ,

---- new Patient entered ----------

Patient Details: John, 55, 000755
Refreshed Queue:
000399, Brooks
000060, Surya
000257, Rishi
000755, John
000154, Ajay
000650, Evan
000413, Cole
000512, David

Enter Valid Age : 'newPatient: Sam ,'

--------next Patient: 1----------

Next Patient for Testing is: 000399, Brooks

Check Input: 'fgsdhqjkdbkew'

---- new Patient entered ----------

Patient Details: Abhimanyu, 67, 000867
Refreshed Queue:
000867, Abhimanyu
000060, Surya
000257, Rishi
000755, John
000154, Ajay
000650, Evan
000413, Cole
000512, David

--------next Patient: 14----------

Only 8patients left
Next Patient for Testing is: 000867, Abhimanyu
Next Patient for Testing is: 000060, Surya
Next Patient for Testing is: 000257, Rishi
Next Patient for Testing is: 000755, John
Next Patient for Testing is: 000154, Ajay
Next Patient for Testing is: 000650, Evan
Next Patient for Testing is: 000413, Cole
Next Patient for Testing is: 000512, David

Program Description :-

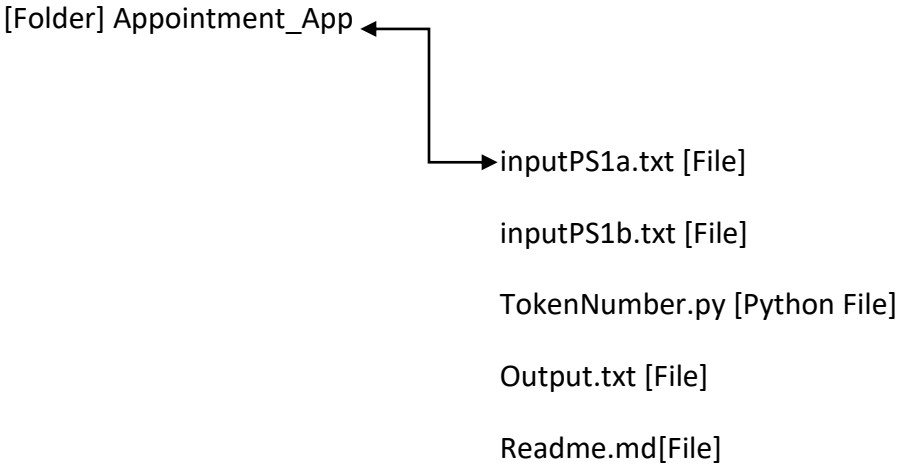
In this program we are trying to develop an appointment software which uses a token system that gives priority to senior citizens to get tested before other patients. It selects next patient for the appointment based on the priority.

Function Name	Description
registerPatient	This function registers the name and age of the patient entering the hospital and assigns them an ID that is then used to capture the details of the patient in the Patient Record.
enqueuePatient	This function assigns the patient a place in the max heap depending on their age. The patient id is inserted into the max heap.
nextPatient	This function prints the next x number of patient IDs and their names that are next in line for testing.
_dequeuePatient	This function removes from the queue the patient ID that has completed the testing and updates the queue.
swapNodeData	This function swaps the data between two nodes.
setTail	This function modifies the Tail reference where the new node during enqueue is added.
revMaxHeapify	This function is used to heapify the tree when we swap the maximum value root node with the smallest value child node.
maxHeapify	This function heapify the binary tree and creates max heap for patient record.
isValidInputPatient	This function validates the input given and verify whether it follows the constraints.

isValidInputnewPatient	This function validates the record of new patient given and verify whether it follows the constraints.
isValidInputnextPatient	This function validates the record of next patient given and verify whether it follows the constraints.

Folder Structure :-

We should have the following files in a folder before executing the program.



Run time Complexity :-

registerPatient:

It has constant time and space complexity since we make a new patient object with the provided details.

Time Complexity: O(1)

Space Complexity:O(1)

enqueuePatient:

In the worst case, we will need to upshift to the root node , since we're bubbling up.And since we know that the number of levels in the Heap is logN, i.e. ,height of the tree, then, we will require to check and bubble up only a maximum of logN times.

So, the time complexity for worst-case insertion will be O(logN).

The space complexity would be O(N) .

nextPatient:

It has a constant space time complexity.

Time Complexity: O(1)

Space Complexity:O(1)

_dequeuePatient:

In the worst case, we will need to downshift from the root node ,since we're downShifting therefore the worst case will be to downshift till the bottom of the Heap. And since we know that the number of levels in the Heap is logN then, we will require to check and shift only a maximum of logN times.

So, the time complexity for worst-case deletion will be O(logN),where logN is height of tree.

The space complexity would be O(height of Tree)

swapNodeData:

It has constant space time complexity.

Time Complexity: O(1)

Space Complexity:O(1)

setTail:

Tail references to the node where the new node during enqueue is added.

In worst case the tail may need to be modified from the right-most node of the nth level to the leftmost node of (n+1)th level.

The worst-case time complexity of setting tail is 2* height of tree, or 2*O(logN).

The space complexity of this operation is O(1).

revMaxHeapify:

This is used in conjunction with _dequeuePatient in which the root node is swapped with the last node of the tree in order to delete the root node. Since after this swap the root node is the smallest node in the tree, we need to heapify it in order to satisfy the condition of a max-heap. We start from the root node and travel down the tree unless we find any children which is larger than it, in which case we swap the nodes.

Time complexity: O(1) + O(logN),where logN is height of the Tree.

The space complexity is O(height of Tree).

maxHeapify:

This is used in conjunction with enqueuePatient in which the new node is added at the end of the tree and we bubble up until the condition of a max-heap is satisfied.

Time complexity: O(1) + O(logN),where logN is height of the Tree.

The space complexity is O(height of Tree).

Overall Complexity:

Let us assume the file inputPS1a.txt has N records to be enqueued to the heap. So, in this case the time complexity would be O(N*log(N)) and space complexity would be O(N).

Now, in order to print this queue we need auxiliary heap. In this process we dequeue from the main heap one by one and enqueue in the auxiliary heap. Since initially we have N elements in the main heap the time complexity required for this operation would be

O(N*log(N))+O(1*N)+O(N*log(N)), we assume that the write operation would take constant time.

Now let us assume the file inputPS1b.txt has M records to be enqueued and n dequeue operations with parameters x_1,$x_{2,}$….x_n. After each of the M enqueue operation, a print operation is performed which prints the whole refreshed heap. Thus the total complexity of this operation,

O(M*log(M+N)) + M*(O((M+N)*log(M+N))+O(1*(M+N))+O((M+N)*log(M+N)))+ O((x_1+$x_{2+,}$….x_n)*log(M+N))

Therefore, the total complexity is:

O(N*log(N))+ O(N*log(N))+O(1*N)+O(N*log(N))+ O(M*log(M+N))+

M*(O((M+N)*log(M+N))+O(1*(M+N))+O((M+N)*log(M+N)))+ O((x_1+$x_{2+,}$….x_n)*log(M+N))