## **Assignment 1 : PS 10 – Checkin Counters**

**Problem Statement -** A shopping centre has a certain number of billing counters to assist their customers in self-checkout. They need your assistance to determine the minimum number of self-checkout counters required such that the wait time is minimum and all customers always have access to a counter when required. You are given a list of start time and amount of time (in mins) that would be required by the customer for checkout. It is represented by a pair of integers [start point, checkout time], where 0 <= start point and 0< checkout time. No two customers can use the same counter at that same time, but immediately after a customer is done checking out, another customer can use that same counter. For example, if one customer starts checking out during [2, 6], it means that the customer came at starting point 2 and would require 6 mins for checkout. Hence another customer can use the same counter at any time after starting point 8.

**Assumption** -

1. Wait time for each customer is to be 0.

2. Time entered is of numeric form. No symbols like “:”,”am”,”pm” etc. are being used to denote the time parameter.

3. There are no blank spaces before and after the data in a line and at the beginning and ending of file.

4. There is no constraint on space to be used and number of lines of code.

**Data Structure choice justification:**

We chose heap based priority queue, using arrays and tuples to implement the same, keeping start\_time as priority key.

Why heap? Because that keeps the data sorting always to O(nlogn) over other sorting algorithms that take O(n^2). And then again for priority queue as well, heap is preferred option because it keeps queue sorted.

Why Priority Queue? It is straightforward choice, because the data anyway isn’t sorted, every customer entering first should be given priority.

Why arrays over list? Because arrays are faster to access. Considering heap to always be a complete tree. Our heapify function makes it complete tree arrangement in array.

**Time Complexity of operations:**

1- To read the input file and create a array containing customers as pair of

start time, checkout time. O(n)

2- Sort the array with respect to start time. O(nlogn)

3- Declare priority queue( min-heap). O(nlogn)

4- Root of heap to store checkout time corresponding to first customer of

sorted array and declare initial counter count to 1. O(1)

5- Then, next sorted customer start time is compared with root customer

checkout time.

Case(i) – If start time of next customer is less than checkout time of root

customer.

Next customer checkout time is pushed in min heap and initial counter

count is increased by 1.

Case (ii)- If start time of next customer is greater than checkout time of root

customer.

Root item is popped out and replaced by next customer checkout time.

In this whole pop and push process, min heap condition is maintained

always. O(nlogn)

Above iteration is done for 1 to n-1 customers of sorted array.

The total complexity is O(nlogn)

It is preferred set of operations, owing to the set of data structures used.

**Alternate way of Modelling:**

1. Using Merge sort process of merge sort

2. Create two arrays of start and checkout time separately.

3. Sort both arrays.

4. Create two separate pointers for both array.

5. Now run a loop on both array.

Case 1- For every start time less than or equal to checkout time

Increase platform count by 1

Case 2 – For every start time more than checkout time

Reduce platform size by 1

After complete iteration in loop, give final result.

Above program sorting process time complexity is O(nlogn). This results in

whole program time complexity to O(nlogn).