Some points that may help you in coding the solution of Assignment 7

General

- Name your variables meaningfully, do not just i,j,k,x,y,z for everything. It helps in debugging faster
- Think what functions you will write and maybe write English steps before you start coding. Spending 10-15 minutes before starting the assignment can help save a lot more time debugging later.
- If stuck, debug with small inputs that you can work out by hand and compare with what the program is giving at each step (put printfs in middle). Do not try to debug with a large input directly.

Part 1

First test out the heap functions thoroughly before going to Part 2. **Do not even read Part 2 and 3 until you have coded and tested Part 1.**

After you write the heap functions:

- Write a printheap() function to print the heap nicely
- In the main() function, as you read in each job, Insert it in the heap, with remLength set to jobLength. Print the heap after each insert and verify that the heap is correct for the input. Testing with a 7-8 size input should be ok (you can just test with arbitrary input, not necessary to stick to sample input given)
- After all the inserts, call n extractMin's in a loop. Again print out the heap after each extractMin and verify it is the correct heap.

Remember carefully that for jobs with same remLength, the one with the lower jobId should be chosen as the lower value. So you have to take care of this properly in insert and extractMin. (while checking for <, also check an additional condition (== and lower id)

Note that structures can be assigned to each other directly just like assigning one int to another int, there is no need to copy field by field. Use this to reduce code size while swapping etc.

Part 2

Do not even read Part 3 until you have coded and tested Part 1 and 2.

First sort the jobs in jobList array in non-descending order of their start times. Print the jobs in job list before and after sorting to verify your sorting works.

Then loop until all jobs are finished, with each iteration of the loop corresponding to one timestep.

At each timestep

- Keep track the job id of the current job with a variable (initialized to 0 before the loop)
- Print the current job
- Find if any new jobs have arrived at the beginning of this timestep (use the sorted job list, remember till how long you have already looked); if so insert in heap
- Find if the current job has finished after the last timestep. If so, reset current job to 0
- Now apply the policy to decide which job should be run on the processor at this timestep

Note carefully that timestep 1 starts from time 0, so at any timestep *i*, new jobs mean jobs that have start time (*i*-1). Code this carefully and test, this can mess up things. Also, heap index starts from 1, not 0.

First try to print the correct schedule, and not worry about computing the Turnaround time. Add it later. If the schedule is not correct, turnaround time will not be correct.

For checking Condition 1 in Part 2, from an implementation point of view, it is probably easier to add *x* to the scheduling queue anyway, and then extract the minimum *remLength* job from the queue and check for condition 1. If Condition 1 is not satisfied, insert the job back in the queue again (note that the job you get after insert may not be x, as some other job may have a lower remLength value than x. But it won't matter, as in that case, the remLength value of current running job must be lower, or it would not be running. This makes handling the case of new jobs arrival and running job finishing the same.

Part 3

This part is just incremental over Part 2. Do not start Part 3 before finishing Part 2, it won't work.

Note: For this assignment, you have to submit a separate C file with its own main() function for Part 3.

Start with making a copy of the C file of Part 2, and naming it as asked for in problem. Then modify the code:

- Add anything you want to the definition of heap.
- Change insert and extractMin
- Write decreaseKey. After decreasing key, remember to re-heapify (this is just part of the insert code starting from the node that changes, so borrow from insert code and

change). Note that you need to check whether a job has already started, can do by checking if jobLength = remLength

- Check the heap part again as for Part 1
- Make a copy of the scheduler() function written earlier and modify it to get the new function.

The code change in Part 3 from Part 2 is not much if you do it right. Use the fact that the job ids are unique and range from 1 to *n* for the *n* jobs. If you get the idea correct, it should take you very little time to change the code if done carefully (took us less than 15 minutes).