OS_Lab3

K. Ram Mohan

COE19B055

Q1) Priority Scheduling

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 100
#include "queue.h"

void swap(int *p,int *q)
{
   int temp;
   temp=*p;
   *p=*q;
   *q=temp;
}

//function to sort based on arrival time
void sort_arr(int id[], int arr[], int burst_time[],int temp_b_time[], int priority[], int len)
{
   int i, j;
   int go=1;

   while(go)
   {
      for(i=0; i<len-1; i++)
      {
         go=0;
         for(j=0; j<len-1-i; j++)
```

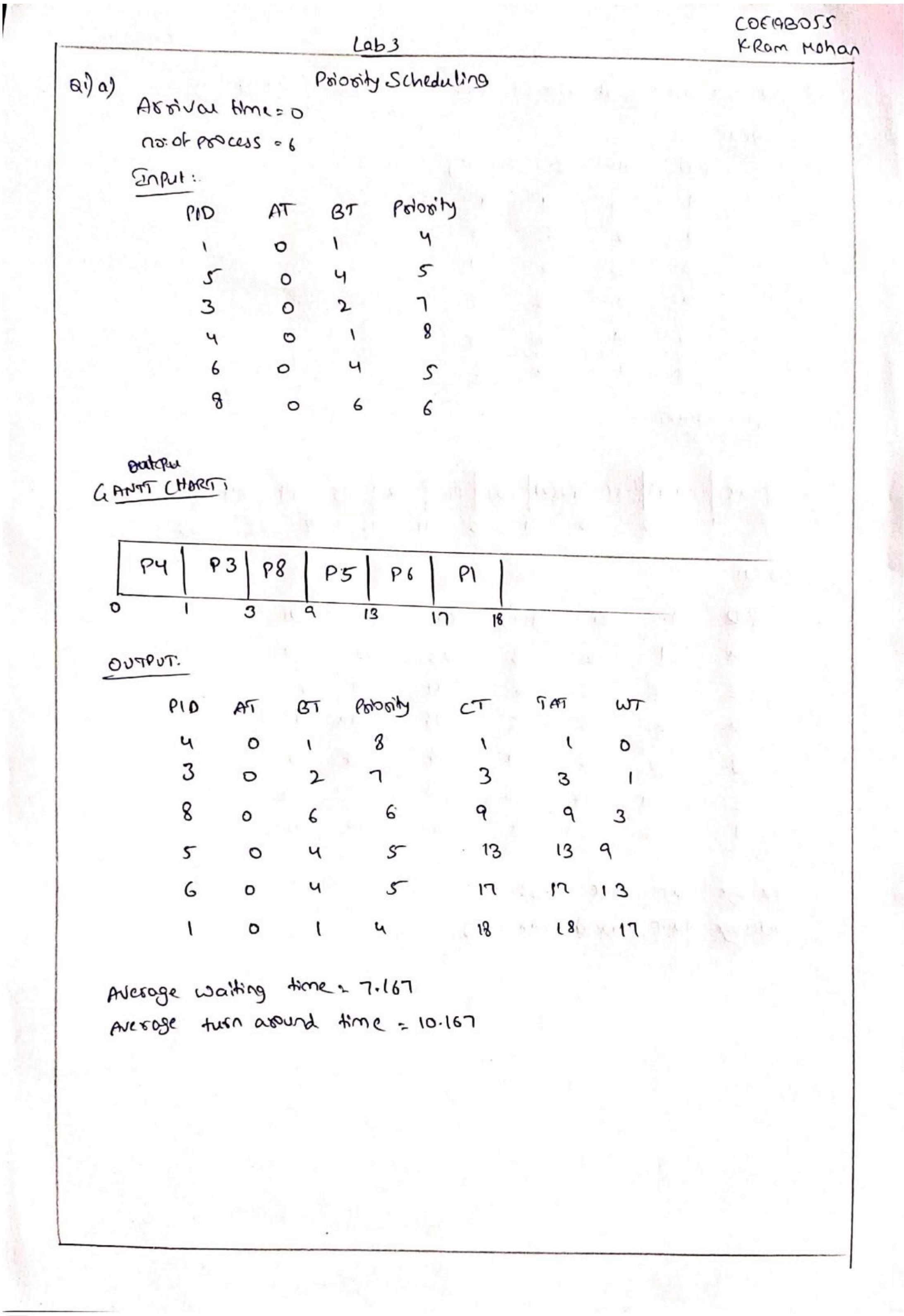
```
{
   if(arr[j]>arr[j+1])
   {
      swap(&arr[j], &arr[j+1]);
      swap(&id[j], &id[j+1]);
      swap(&burst_time[j], &burst_time[j+1]);
      swap(&priority[j], &priority[j+1]);
      swap(&temp_b_time[j], &temp_b_time[j+1]);
      go=1;
   }

   if(arr[j]==arr[j+1])
   {
      if(priority[j]<priority[j+1])
      {
         swap(&arr[j], &arr[j+1]);
         swap(&id[j], &id[j+1]);
         swap(&burst_time[j], &burst_time[j+1]);
         swap(&priority[j], &priority[j+1]);
         swap(&temp_b_time[j], &temp_b_time[j+1]);
         go=1;
      }

      if(priority[j]==priority[j+1])
      {
         if(id[j]>id[j+1])
         {
            swap(&arr[j], &arr[j+1]);
            swap(&id[j], &id[j+1]);
```

```
                    swap(&burst_time[j], &burst_time[j+1]);
                    swap(&temp_b_time[j], &temp_b_time[j+1]);
                    swap(&priority[j], &priority[j+1]);
                    go=1;
                }
            }
        }
    }
  }
 }
}

int index_id(int id, int process_id_arr[], int len)
{
    int i;
    for(i=0; i<len; i++)
    {
        if(process_id_arr[i]==id)
        {
            return i;
        }
    }
    return -1;
}

int pre(int arr[], int len, int val)
{
    int j;
    for(j=0; j<len; j++)
    {
        if(arr[j] == val)
        {
            return 1;
        }
    }
    return 0;
}

int main()
{
    int total_process, i, j;
    int process_id_arr[SIZE], arr_time[SIZE], burst_time[SIZE], temp_b_time[SIZE],
                       in_queue[SIZE], in_queue_len=0, priority[SIZE], c_time=0, element;
    int index, index1, start=0;
    float tot_waiting_time=0, tot_trt=0;

    printf("Enter no of process: ");
    scanf("%d", &total_process);

    for(i=0; i<total_process; i++)
    {
        printf("Process_id of process %d: ", i+1);
        scanf("%d", &process_id_arr[i]);
        printf("Arrival time of process %d: ", i+1);
        scanf("%d", &arr_time[i]);
        printf("Burst time of process %d: ", i+1);
        scanf("%d", &element);
        burst_time[i]=element;
        temp_b_time[i]=element;
        printf("Priority of process %d: ", i+1);
        scanf("%d", &priority[i]);
    }

    sort_arr(process_id_arr, arr_time, burst_time, temp_b_time, priority, total_process);

    int completion_time[SIZE], waiting_time[SIZE], turn_arnd_time[SIZE];

    //adding the first available process into in_queue array
    in_queue[in_queue_len++]  = process_id_arr[0];

    int total = total_process;

    while(total!=0)
    {
        int temp=0;
        //sorting the current available process according to their priority
        for(i=0; i<in_queue_len-1; i++)
        {
            for(j=start; j<in_queue_len-1-i; j++)
            {
                index  = index_id(in_queue[j], process_id_arr, total_process);
                index1 = index_id(in_queue[j+1], process_id_arr, total_process);

                if(priority[index]<priority[index1])
                {
                    swap(&in_queue[j], &in_queue[j+1]);
                }
            }
        }
```

```
element = in_queue[start];
index = index_id(element, process_id_arr, total_process);

if(c_time==0)
{
    c_time = arr_time[index];
}

if(temp_b_time[index]>1)
{
    temp_b_time[index] = temp_b_time[index]-1;
    c_time++;
    temp=1;
}
else if(temp_b_time[index]==1)
{
    temp_b_time[index] = temp_b_time[index]-1;
    c_time++;
    start++;
    completion_time[index] = c_time;
    total--;
    temp=1;
}

//if cpu  is in idle state
if(temp==0)
{
    c_time++;
```

```
        }

        for(i=0; i<total_process; i++)
        {
            if(arr_time[i]<=c_time && !(pre(in_queue, in_queue_len, process_id_arr[i])))
            {
                in_queue[in_queue_len++]=process_id_arr[i];
            }
        }
    }


    for(i=0; i<total_process; i++)
    {
        turn_arnd_time[i] = completion_time[i] - arr_time[i];
        tot_trt = tot_trt + turn_arnd_time[i];
    }

    for(i=0; i<total_process; i++)
    {
        waiting_time[i] = turn_arnd_time[i] - burst_time[i];
        tot_waiting_time = tot_waiting_time + waiting_time[i];
    }

    printf("Processes   arrival_time   Burst time      priority  completion time   Turn around
time   Waiting time\n");
    for(i=0; i<total_process; i++)
    {
        printf("%d", process_id_arr[i]);
        printf("\t \t %d", arr_time[i]);
        printf("\t \t %d", burst_time[i]);
        printf("\t \t %d", priority[i]);
        printf("\t \t %d", completion_time[i]);
        printf("\t \t %d", turn_arnd_time[i]);
        printf("\t \t %d \n", waiting_time[i]);
    }
    float avg_wait = tot_waiting_time/total_process;
    float avg_trt = tot_trt/total_process;
    printf("Average waiting time is : %f\n", avg_wait);
    printf("Average turn around time is: %f", avg_trt);
}
```

a) All inputs with arrival time 0



COE19B055
K.Ram Mohan

Lab 3

Q1) a) Priority Scheduling

Arrival time = 0

no: of process = 6

Input:

PID	AT	BT	Priority
1	0	1	4
5	0	4	5
3	0	2	7
4	0	1	8
6	0	4	5
8	0	6	6

Output

GANTT CHART:

P4	P3	P8	P5	P6	P1

0 1 3 9 13 17 18

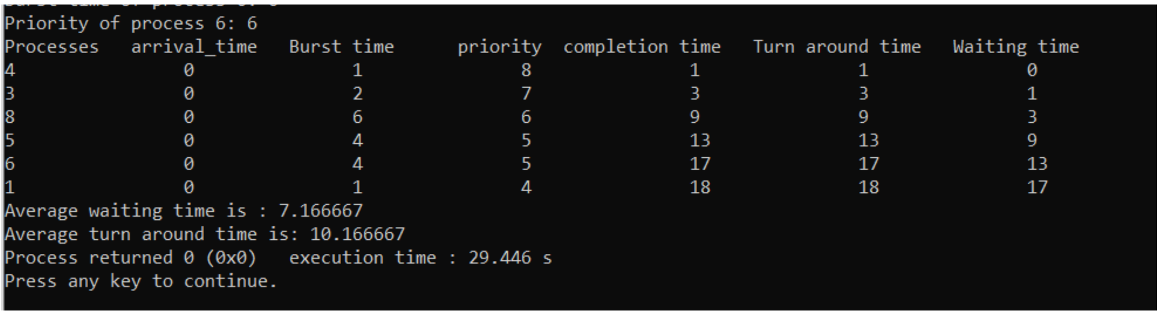
OUTPUT:

PID	AT	BT	Priority	CT	TAT	WT
4	0	1	8	1	1	0
3	0	2	7	3	3	1
8	0	6	6	9	9	3
5	0	4	5	13	13	9
6	0	4	5	17	17	13
1	0	1	4	18	18	17

Average waiting time = 7.167

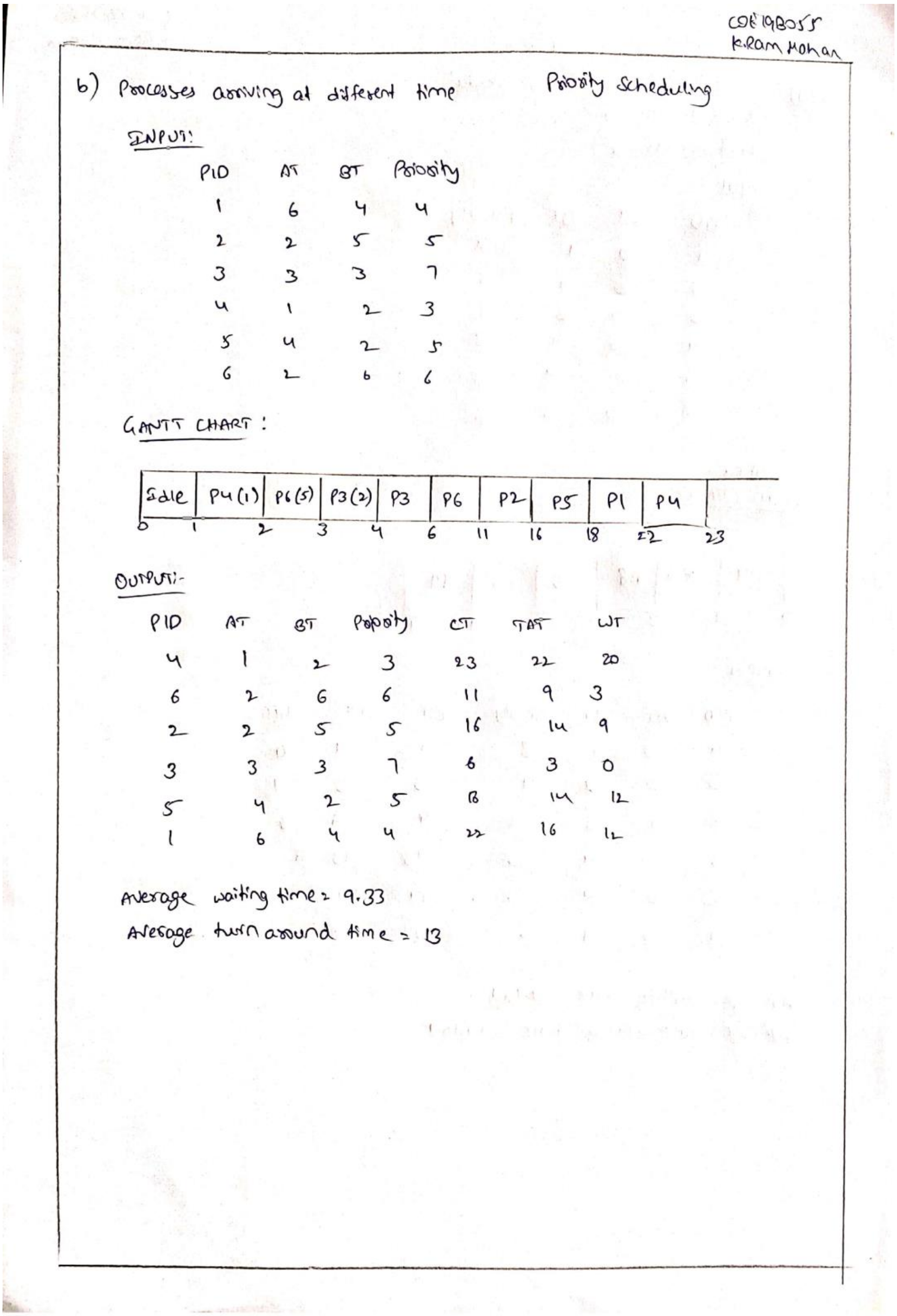
Average turn around time = 10.167

Output:



```
Priority of process 6: 6
Processes   arrival_time   Burst time      priority  completion time   Turn around time   Waiting time
4                0              1              8             1                1                0
3                0              2              7             3                3                1
8                0              6              6             9                9                3
5                0              4              5             13               13               9
6                0              4              5             17               17               13
1                0              1              4             18               18               17
Average waiting time is : 7.166667
Average turn around time is: 10.166667
Process returned 0 (0x0)   execution time : 29.446 s
Press any key to continue.
```

b) Inputs with different arrival time



COE19B055
K.Ram Mohan

b) Processes arriving at different time Priority Scheduling

INPUT:

PID	AT	BT	Priority
1	6	4	4
2	2	5	5
3	3	3	7
4	1	2	3
5	4	2	5
6	2	6	6

GANTT CHART :

Idle	P4(1)	P6(5)	P3(2)	P3	P6	P2	P5	P1	P4

0 1 2 3 4 6 11 16 18 22 23

OUTPUT:-

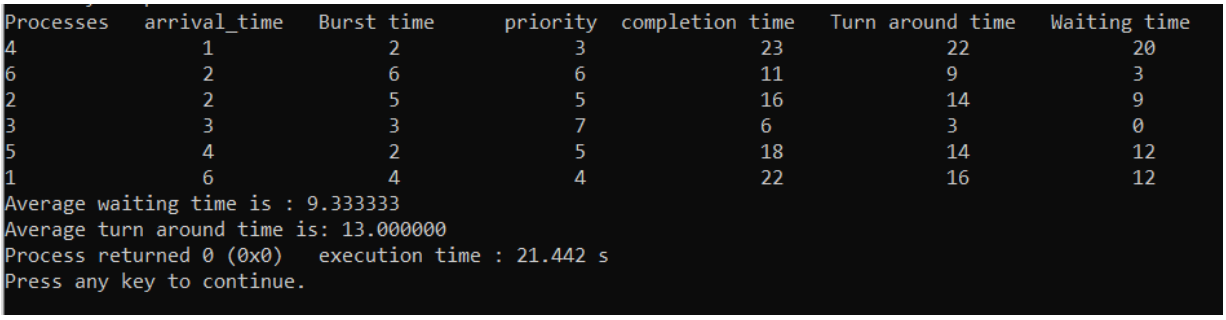
PID	AT	BT	Priority	CT	TAT	WT
4	1	2	3	23	22	20
6	2	6	6	11	9	3
2	2	5	5	16	14	9
3	3	3	7	6	3	0
5	4	2	5	13	14	12
1	6	4	4	22	16	12

Average waiting time = 9.33

Average turn around time = 13

Scanned with CamScanner

Output:



```
Processes   arrival_time   Burst time      priority  completion time   Turn around time   Waiting time
4                1               2               3              23               22                20
6                2               6               6              11               9                 3
2                2               5               5              16               14                9
3                3               3               7              6                3                 0
5                4               2               5              18               14                12
1                6               4               4              22               16                12
Average waiting time is : 9.333333
Average turn around time is: 13.000000
Process returned 0 (0x0)   execution time : 21.442 s
Press any key to continue.
```

Q2) Round Robin

```
#include<stdio.h>

#include<stdlib.h>

#include "queue.h"


void swap(int *p,int *q)

{

   int temp;

   temp=*p;

   *p=*q;

   *q=temp;

}


//function to sort based on arrival time

void sort_arr(int id[], int arr[], int burst_time[], int temp_b_time[], int len)

{

   int i, j;

   int go=1;


   while(go)

   {

      for(i=0; i<len-1; i++)

      {

         go=0;

         for(j=0; j<len-1-i; j++)

         {
                if(arr[j]>arr[j+1])
                {
                    swap(&arr[j], &arr[j+1]);
                    swap(&id[j], &id[j+1]);
                    swap(&burst_time[j], &burst_time[j+1]);
                    swap(&temp_b_time[j], &temp_b_time[j+1]);
                    go=1;
                }

                if(arr[j]==arr[j+1])
                {
                    if(id[j]>id[j+1])
                    {
                        swap(&arr[j], &arr[j+1]);
                        swap(&id[j], &id[j+1]);
                        swap(&burst_time[j], &burst_time[j+1]);
                        swap(&temp_b_time[j], &temp_b_time[j+1]);
                        go=1;
                    }
                }
            }
        }
    }
}

int index_id(int id, int process_id_arr[], int len)
{
    int i;
    for(i=0; i<len; i++)
    {
        if(process_id_arr[i]==id)
        {
            return i;
        }
    }
    return -1;
}

int pre(int arr[], int len, int val)
{
    int j;
    for(j=0; j<len; j++)
    {
        if(arr[j] == val)
        {
            return 1;
        }
    }
    return 0;
}

int main()
{
    int total_process, i, j;

    struct queue process_id;
    process_id.enqueue=enqueue1;
    process_id.dequeue=dequeue1;
    process_id.display=display1;

    process_id.empty=empty1;

    process_id.front=-1;

    process_id.rear=-1;

    int process_id_arr[100], arr_time[100], burst_time[100], temp_b_time[100],
in_queue[100], in_queue_len=0, time_q, c_time=0, element, index;

    float tot_waiting_time=0, tot_trt=0;


    printf("Enter no of process: ");

    scanf("%d", &total_process);


    printf("Enter time quantum: ");

    scanf("%d", &time_q);


    for(i=0; i<total_process; i++)

    {

        printf("Process_id of process %d: ", i+1);

        scanf("%d", &process_id_arr[i]);

        printf("Arrival time of process %d: ", i+1);

        scanf("%d", &arr_time[i]);

        printf("Burst time of process %d: ", i+1);

        scanf("%d", &element);

        burst_time[i]=element;

        temp_b_time[i]=element;

    }


    sort_arr(process_id_arr, arr_time, burst_time, temp_b_time, total_process);


    int completion_time[100], waiting_time[100], turn_arnd_time[100];
    //adding the first available process into queue
    process_id.enqueue(process_id_arr[0], &process_id);
    in_queue[in_queue_len++] = process_id_arr[0];

    i=0;
    int total=total_process;
    while(!(process_id.empty(&process_id)) || total!=0)
    {
        //There are process that did not come to queue which means their arrival time is more
than current c_time
        if(process_id.empty(&process_id))
        {
            c_time++;
            for(i=0; i<total_process; i++)
            {
                if(arr_time[i]<=c_time && !(pre(in_queue, in_queue_len, process_id_arr[i])))
                {
                    in_queue[in_queue_len++] = process_id_arr[i];
                    process_id.enqueue(process_id_arr[i], &process_id);
                }
            }
        }
        else
        {
            int temp=0, temp_1=0;
            //getting the process id to be executed
            element = process_id.array[process_id.front];
            //getting the index of the process id
            index = index_id(element, process_id_arr, total_process);
            //dequeuing the process for execution
```

```
process_id.dequeue(&process_id);

//if it is the first process available or if arr_time is more than c_time
if(c_time==0 || arr_time[index]>c_time)
{
   c_time = arr_time[index];
}

//if burst time is more than time slice
if(temp_b_time[index]>time_q)
{
   temp_b_time[index] = temp_b_time[index]-time_q;
   c_time = c_time + time_q;
   temp=1;
   temp_1=1;
}
else
{
   c_time = c_time + temp_b_time[index];
   temp_b_time[index] = 0;
   completion_time[index] = c_time;
   //printf("%d - %d\n", index, completion_time[index]);
   total--;
   temp_1=1;
}

//enqueue the process arrived by the current c_time
for(i=0; i<total_process; i++)
{
```

```
          if(arr_time[i]<=c_time && !(pre(in_queue, in_queue_len, process_id_arr[i])))
          {
            in_queue[in_queue_len++] = process_id_arr[i];
            process_id.enqueue(process_id_arr[i], &process_id);
          }
        }

        if(temp==1)
        {
          process_id.enqueue(element, &process_id);
        }
      }
    }

    for(i=0; i<total_process; i++)
    {
      turn_arnd_time[i] = completion_time[i] - arr_time[i];
      tot_trt = tot_trt + turn_arnd_time[i];
    }

    for(i=0; i<total_process; i++)
    {
      waiting_time[i] = turn_arnd_time[i] - burst_time[i];
      tot_waiting_time = tot_waiting_time + waiting_time[i];
    }

    printf("Processes   arrival_time   Burst time  completion time   Turn around time   Waiting time\n");
    for(i=0; i<total_process; i++)
    {
        printf("%d", process_id_arr[i]);

        printf("\t \t %d", arr_time[i]);

        printf("\t \t %d", burst_time[i]);

        printf("\t \t %d", completion_time[i]);

        printf("\t \t %d", turn_arnd_time[i]);

        printf("\t \t %d \n", waiting_time[i]);

    }

    float avg_wait = tot_waiting_time/total_process;

    float avg_trt = tot_trt/total_process;

    printf("Average waiting time is : %f\n", avg_wait);

    printf("Average turn around time is: %f", avg_trt);

}
```

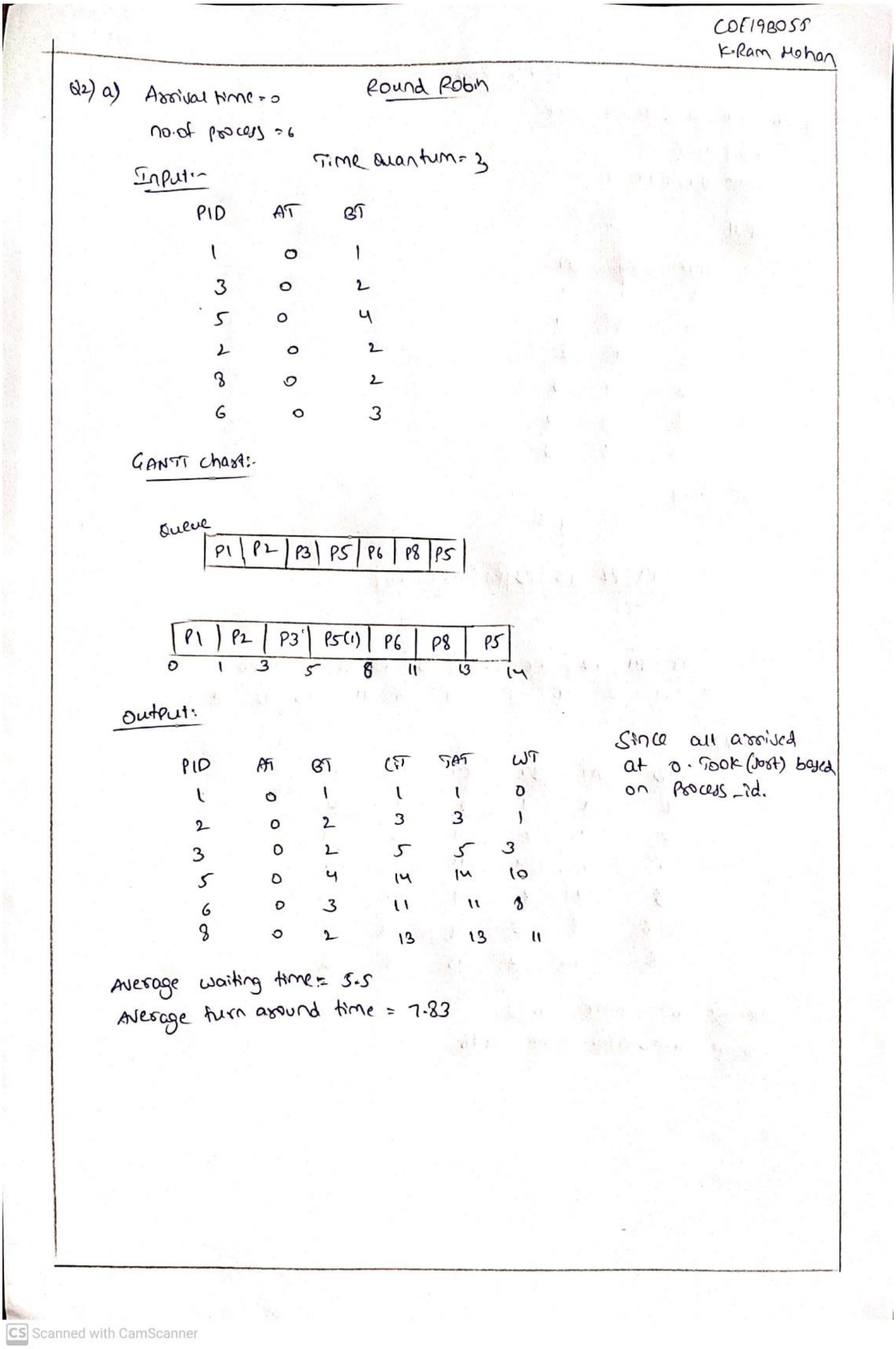
Queue(Header file)

```
#define size 100
struct queue
{
   int front,rear;
   int array[size];
   void (*enqueue)(int ,struct queue* );
   void (*dequeue)(struct queue* );
   void (*display)(struct queue* );
   int (*empty) (struct queue* );
};
void enqueue1(int ,struct queue* );
void dequeue1(struct queue* );
void display1(struct queue* );
void enqueue1(int item,struct queue *que)
{
   if((que->rear==size-1 && que->front==0) || (que->front==que->rear+1))
   {
      printf("queue is full\n");
   }
   else
   {
      if(que->front==-1)
      {
         que->front=0;
      }
      que->rear=(que->rear+1)%size;
```

```
        que->array[que->rear]=item;
    }
}
void dequeue1(struct queue *que)
{
    if(que->front==-1)
    {
        printf("queue is empty\n");
    }
    else
    {
        if(que->front==que->rear)
        {
            que->rear=-1;
            que->front=-1;
        }
        else
        {
            que->front=(que->front+1)%size;
        }
    }
}
void display1(struct queue *que)
{
    int i;
    if(que->rear==-1 && que->front==-1)
    {
        printf("The queue is empty\n");
    }
    else
    {
        if(que->front < que->rear)
        {
            for(i=que->front;i<=que->rear;i++)
            {
                printf("%d ",que->array[i]);
            }
        }
        else if(que->front == que->rear)
        {
            printf("%d ",que->array[que->front]);
        }
        else
        {
            for(i=que->front;i<size;i++)
            {
                printf("%d ",que->array[i]);
            }
            for(i=0;i<=que->rear;i++)
            {
                printf("%d ",que->array[i]);
            }
        }
        printf("\n");
    }
}

int empty1(struct queue *que)
```

```
{
   if(que->front==-1)
   {
      return 1;
   }
   return 0;
}
```

a) All inputs with arrival time 0



COE19B055
K.Ram Mohan

Q2) a) Arrival time = 0 Round Robin

no.of process = 6

Time Quantum = 3

Input:-

PID	AT	BT
1	0	1
3	0	2
5	0	4
2	0	2
8	0	2
6	0	3

GANTT chart:-

Queue

P1	P2	P3	P5	P6	P8	P5

P1	P2	P3'	P5(1)	P6	P8	P5

0 1 3 5 8 11 13 14

Output:

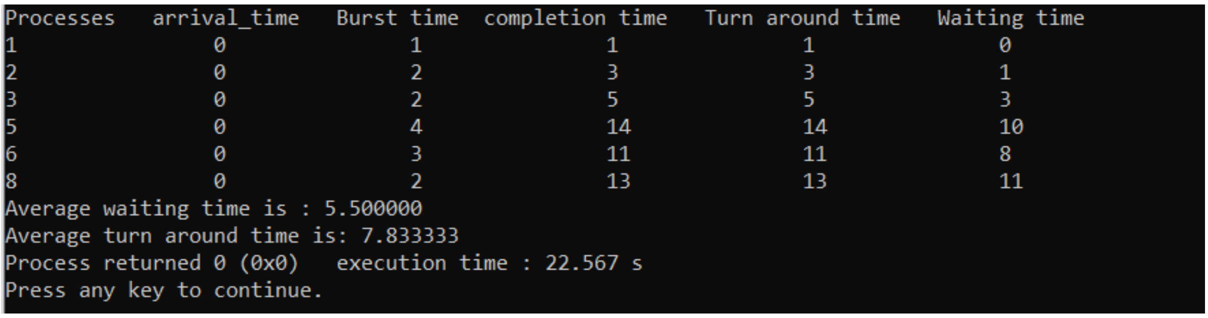
PID	AT	BT	CT	TAT	WT
1	0	1	1	1	0
2	0	2	3	3	1
3	0	2	5	5	3
5	0	4	14	14	10
6	0	3	11	11	8
8	0	2	13	13	11

Since all arrived at 0. Took (sort) based on Process_id.

Average waiting time = 5.5
Average turn around time = 7.83

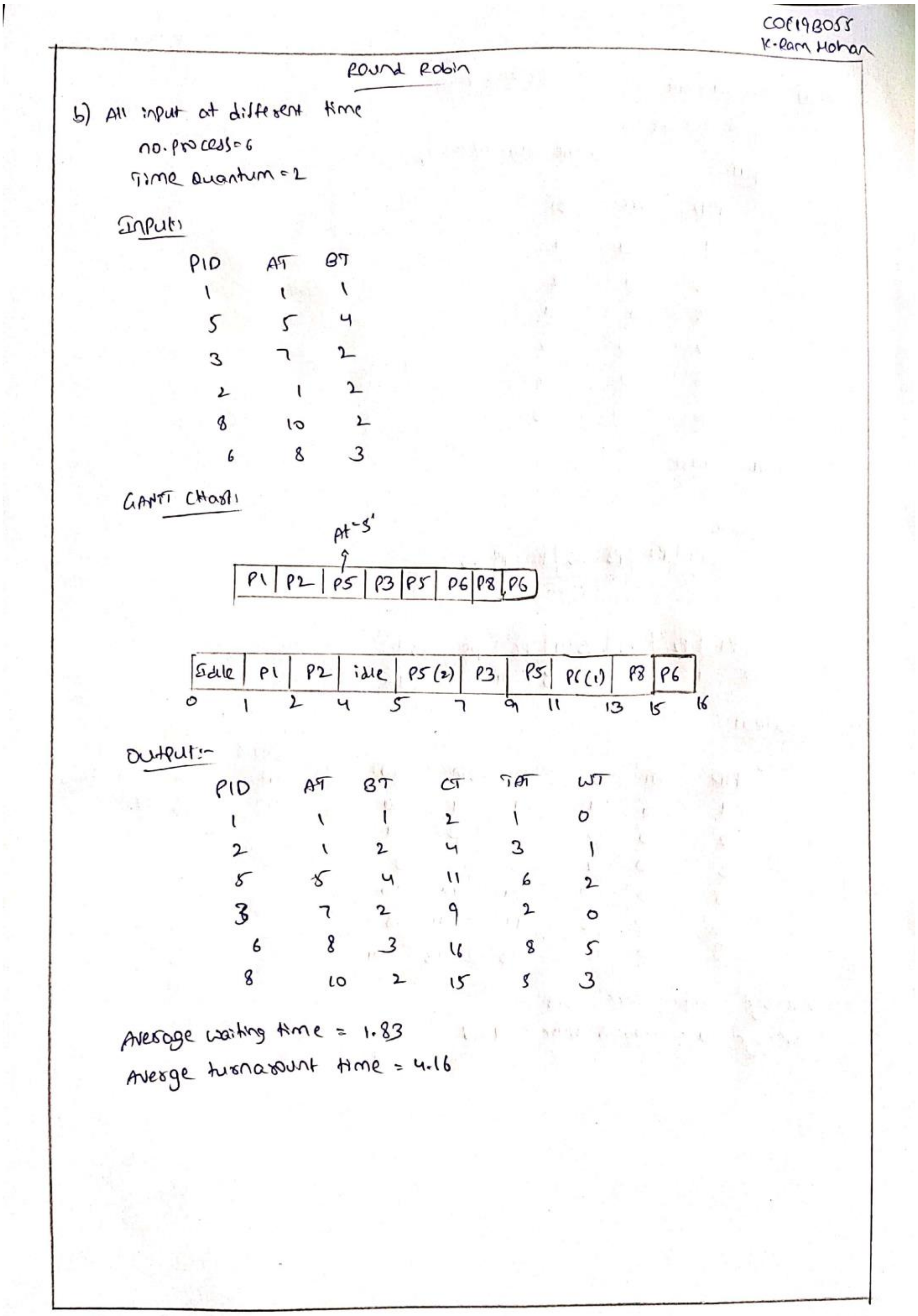
Scanned with CamScanner

Output:



```
Processes   arrival_time   Burst time  completion time   Turn around time   Waiting time
1                0               1               1               1               0
2                0               2               3               3               1
3                0               2               5               5               3
5                0               4               14              14              10
6                0               3               11              11              8
8                0               2               13              13              11
Average waiting time is : 5.500000
Average turn around time is: 7.833333
Process returned 0 (0x0)   execution time : 22.567 s
Press any key to continue.
```

b) Inputs with different arrival time



COE19B055
K-Ram Mohan

Round Robin

b) All input at different time

no. process = 6

Time Quantum = 2

Inputs

PID	AT	BT
1	1	1
5	5	4
3	7	2
2	1	2
8	10	2
6	8	3

GANTT Chart:

At-5'

P1	P2	P5	P3	P5	P6	P8	P6

Idle	P1	P2	idle	P5 (2)	P3	P5	P6 (1)	P8	P6

0 1 2 4 5 7 9 11 13 15 16

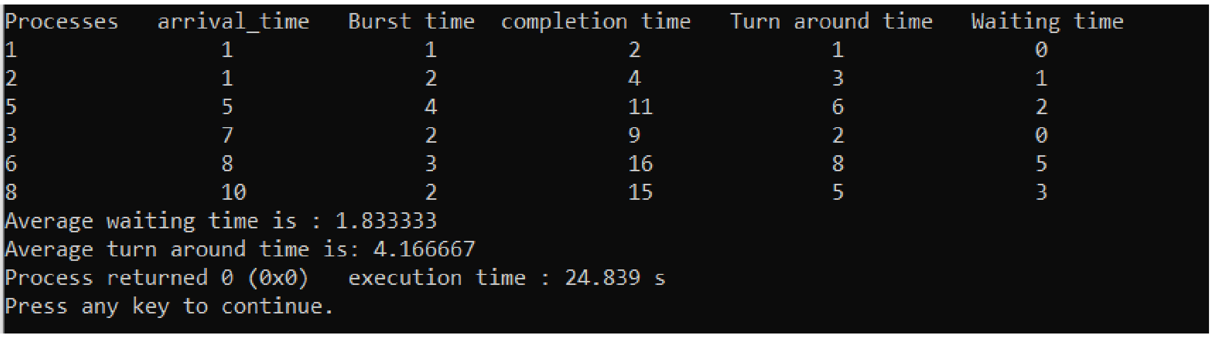
Output:-

PID	AT	BT	CT	TAT	WT
1	1	1	2	1	0
2	1	2	4	3	1
5	5	4	11	6	2
3	7	2	9	2	0
6	8	3	16	8	5
8	10	2	15	5	3

Average waiting time = 1.83

Averge turnaround time = 4.16

Output:



```
Processes   arrival_time   Burst time  completion time   Turn around time   Waiting time
1                1               1               2               1               0
2                1               2               4               3               1
5                5               4               11              6               2
3                7               2               9               2               0
6                8               3               16              8               5
8                10              2               15              5               3
Average waiting time is : 1.833333
Average turn around time is: 4.166667
Process returned 0 (0x0)   execution time : 24.839 s
Press any key to continue.
```