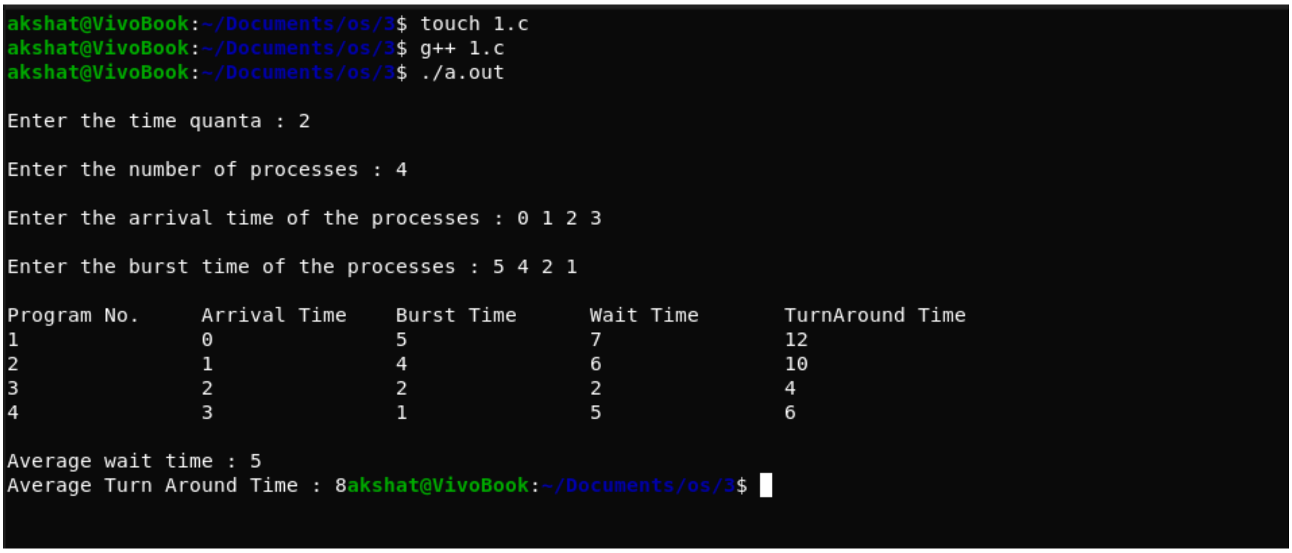
Name:Akshat Khosya
Roll No: 20106
1.

```
#include <iostream>

using namespace std;

void queueUpdation(int queue[],int timer,int arrival[],int n, int maxProccessIndex){
    int zeroIndex;
    for(int i = 0; i < n; i++){
        if(queue[i] == 0){
            zeroIndex = i;
            break;
        }
    }
    queue[zeroIndex] = maxProccessIndex + 1;
}

void queueMaintainence(int queue[], int n){
    for(int i = 0; (i < n-1) && (queue[i+1] != 0) ; i++){
        int temp = queue[i];
        queue[i] = queue[i+1];
        queue[i+1] = temp;
    }
}

void checkNewArrival(int timer, int arrival[], int n, int maxProccessIndex,int queue[]){
    if(timer <= arrival[n-1]){
       bool newArrival = false;
       for(int j = (maxProccessIndex+1); j < n; j++){
             if(arrival[j] <= timer){
              if(maxProccessIndex < j){
                 maxProccessIndex = j;
                 newArrival = true;
              }
           }
       }
       //adds the incoming process to the ready queue
       //(if any arrives)
       if(newArrival)
          queueUpdation(queue,timer,arrival,n, maxProccessIndex);
    }
}

//Driver Code
int main(){
    int n,tq, timer = 0, maxProccessIndex = 0;
    float avgWait = 0, avgTT = 0;
    cout << "\nEnter the time quanta : ";
    cin>>tq;
    cout << "\nEnter the number of processes : ";
    cin>>n;
```

```
int arrival[n], burst[n], wait[n], turn[n], queue[n], temp_burst[n];
bool complete[n];

cout << "\nEnter the arrival time of the processes : ";
for(int i = 0; i < n; i++)
    cin>>arrival[i];

cout << "\nEnter the burst time of the processes : ";
for(int i = 0; i < n; i++){
    cin>>burst[i];
    temp_burst[i] = burst[i];
}

for(int i = 0; i < n; i++){    //Initializing the queue and complete array
    complete[i] = false;
    queue[i] = 0;
}
while(timer < arrival[0])    //Incrementing Timer until the first process arrives
    timer++;
queue[0] = 1;

while(true){
    bool flag = true;
    for(int i = 0; i < n; i++){
        if(temp_burst[i] != 0){
            flag = false;
            break;
        }
    }
    if(flag)
        break;

    for(int i = 0; (i < n) && (queue[i] != 0); i++){
        int ctr = 0;
        while((ctr < tq) && (temp_burst[queue[0]-1] > 0)){
            temp_burst[queue[0]-1] -= 1;
            timer += 1;
            ctr++;

            //Checking and Updating the ready queue until all the processes arrive
            checkNewArrival(timer, arrival, n, maxProccessIndex, queue);
        }
        //If a process is completed then store its exit time
        //and mark it as completed
        if((temp_burst[queue[0]-1] == 0) && (complete[queue[0]-1] == false)){
            //turn array currently stores the completion time
            turn[queue[0]-1] = timer;
            complete[queue[0]-1] = true;
        }

          //checks whether or not CPU is idle
        bool idle = true;
```

```
            if(queue[n-1] == 0){
                for(int i = 0; i < n && queue[i] != 0; i++){
                    if(complete[queue[i]-1] == false){
                        idle = false;
                    }
                }
            }
            else
                idle = false;

            if(idle){
                timer++;
                checkNewArrival(timer, arrival, n, maxProccessIndex, queue);
            }

            //Maintaining the entries of processes
            //after each premption in the ready Queue
            queueMaintainence(queue,n);
        }
    }

    for(int i = 0; i < n; i++){
        turn[i] = turn[i] - arrival[i];
        wait[i] = turn[i] - burst[i];
    }

    cout << "\nProgram No.\tArrival Time\tBurst Time\tWait Time\tTurnAround Time"
          << endl;
    for(int i = 0; i < n; i++){
        cout<<i+1<<"\t\t"<<arrival[i]<<"\t\t"
          <<burst[i]<<"\t\t"<<wait[i]<<"\t\t"<<turn[i]<<endl;
    }
    for(int i =0; i< n; i++){
        avgWait += wait[i];
        avgTT += turn[i];
    }
    cout<<"\nAverage wait time : "<<(avgWait/n)
      <<"\nAverage Turn Around Time : "<<(avgTT/n);

    return 0;

}
```



```
akshat@VivoBook:~/Documents/os/3$ touch 1.c
akshat@VivoBook:~/Documents/os/3$ g++ 1.c
akshat@VivoBook:~/Documents/os/3$ ./a.out

Enter the time quanta : 2

Enter the number of processes : 4

Enter the arrival time of the processes : 0 1 2 3

Enter the burst time of the processes : 5 4 2 1

Program No.     Arrival Time    Burst Time      Wait Time       TurnAround Time
1               0               5               7               12
2               1               4               6               10
3               2               2               2               4
4               3               1               5               6

Average wait time : 5
Average Turn Around Time : 8akshat@VivoBook:~/Documents/os/3$ 
```

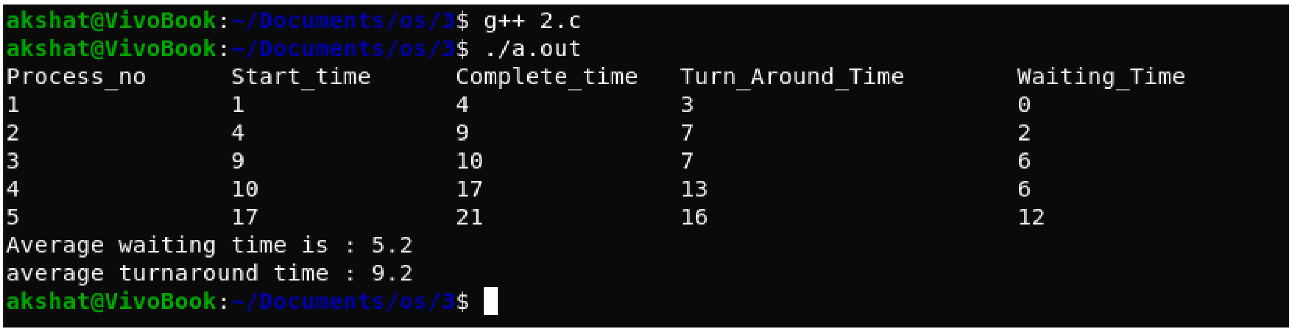
2.

```
#include <bits/stdc++.h>
using namespace std;

#define totalprocess 5

// Making a struct to hold the given input

struct process
{
int at,bt,pr,pno;
};

process proc[50];

/*
Writing comparator function to sort according to priority if
arrival time is same
*/

bool comp(process a,process b)
{
if(a.at == b.at)
{
return a.pr<b.pr;
}
else
{
   return a.at<b.at;
}
}

// Using FCFS Algorithm to find Waiting time
void get_wt_time(int wt[])
{
// declaring service array that stores cumulative burst time
```

```
int service[50];

// Initialising initial elements of the arrays
service[0] = proc[0].at;
wt[0]=0;


for(int i=1;i<totalprocess;i++)
{
service[i]=proc[i-1].bt+service[i-1];

wt[i]=service[i]-proc[i].at;

// If waiting time is negative, change it into zero

   if(wt[i]<0)
   {
   wt[i]=0;
   }
}

}

void get_tat_time(int tat[],int wt[])
{
// Filling turnaroundtime array

for(int i=0;i<totalprocess;i++)
{
   tat[i]=proc[i].bt+wt[i];
}

}

void findgc()
{
//Declare waiting time and turnaround time array
int wt[50],tat[50];

double wavg=0,tavg=0;

// Function call to find waiting time array
get_wt_time(wt);
//Function call to find turnaround time
get_tat_time(tat,wt);

int stime[50],ctime[50];

stime[0] = proc[0].at;
ctime[0]=stime[0]+tat[0];

// calculating starting and ending time
```

```
for(int i=1;i<totalprocess;i++)
   {
      stime[i]=ctime[i-1];
      ctime[i]=stime[i]+tat[i]-wt[i];
   }

cout<<"Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaiting_Time"<<endl;

   // display the process details

for(int i=0;i<totalprocess;i++)
   {
      wavg += wt[i];
      tavg += tat[i];

      cout<<proc[i].pno<<"\t\t"<<
         stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<<
         tat[i]<<"\t\t\t"<<wt[i]<<endl;
   }

      // display the average waiting time
      //and average turn around time

   cout<<"Average waiting time is : ";
   cout<<wavg/(float)totalprocess<<endl;
   cout<<"average turnaround time : ";
   cout<<tavg/(float)totalprocess<<endl;

}

int main()
{
int arrivaltime[] = { 1, 2, 3, 4, 5 };
int bursttime[] = { 3, 5, 1, 7, 4 };
int priority[] = { 3, 4, 1, 7, 8 };

for(int i=0;i<totalprocess;i++)
{
   proc[i].at=arrivaltime[i];
   proc[i].bt=bursttime[i];
   proc[i].pr=priority[i];
   proc[i].pno=i+1;
   }

   //Using inbuilt sort function

   sort(proc,proc+totalprocess,comp);

   //Calling function findgc for finding Gantt Chart

   findgc();
```

```
    return 0;
}
```



```
akshat@VivoBook:~/Documents/os/3$ g++ 2.c
akshat@VivoBook:~/Documents/os/3$ ./a.out
Process_no      Start_time      Complete_time   Turn_Around_Time        Waiting_Time
1               1               4               3                       0
2               4               9               7                       2
3               9               10              7                       6
4               10              17              13                      6
5               17              21              16                      12
Average waiting time is : 5.2
average turnaround time : 9.2
akshat@VivoBook:~/Documents/os/3$ 
```