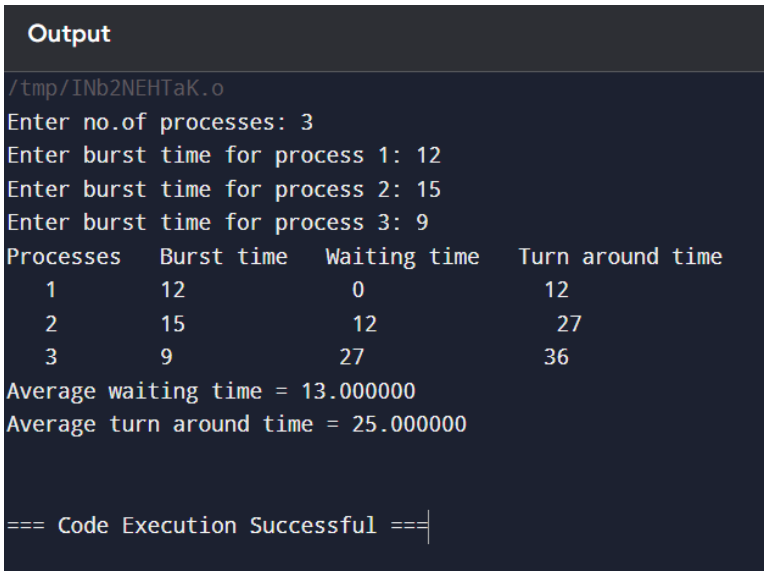
LAB PERFORMANCE

Kadiyam venkata bhargav
AP22110010051
CSE-Q

1. FCFS

```
#include<stdio.h>
void findingwt(int p[], int n, int bt[], int wt[]) {
   wt[0] = 0;
   for (int i = 1; i < n; i++)
      wt[i] = bt[i-1] + wt[i-1];
}
void findingtat(int p[], int n, int bt[], int wt[], int tat[]) {
   for (int i = 0; i < n; i++)
      tat[i] = bt[i] + wt[i];
}
void findingavgtime(int p[], int n, int bt[]) {
   int wt[n], tat[n], totalwt = 0, totaltat = 0;
   findingwt(p, n, bt, wt);
   findingtat(p, n, bt, wt, tat);
   printf("Processes Burst time Waiting time  Turn around time\n"); for
   (int i = 0; i < n; i++) {
      totalwt += wt[i];
      totaltat += tat[i];
      printf("   %d        %d            %d            %d\n", (i+1), bt[i], wt[i], tat[i]);
   }
   printf("Average waiting time = %f\n", (float)totalwt / n);
   printf("Average turn around time = %f\n", (float)totaltat / n);
}
int main() {
   int n;
   printf("Enter no.of processes: ");
   scanf("%d", &n);
   int p[n], bursttime[n];
   for (int i = 0; i < n; i++) {
      p[i] = i + 1;
      printf("Enter burst time for process %d: ", i + 1);
      scanf("%d", &bursttime[i]);
   }
   findingavgtime(p, n, bursttime);
   return 0;
}
```

OUTPUT:

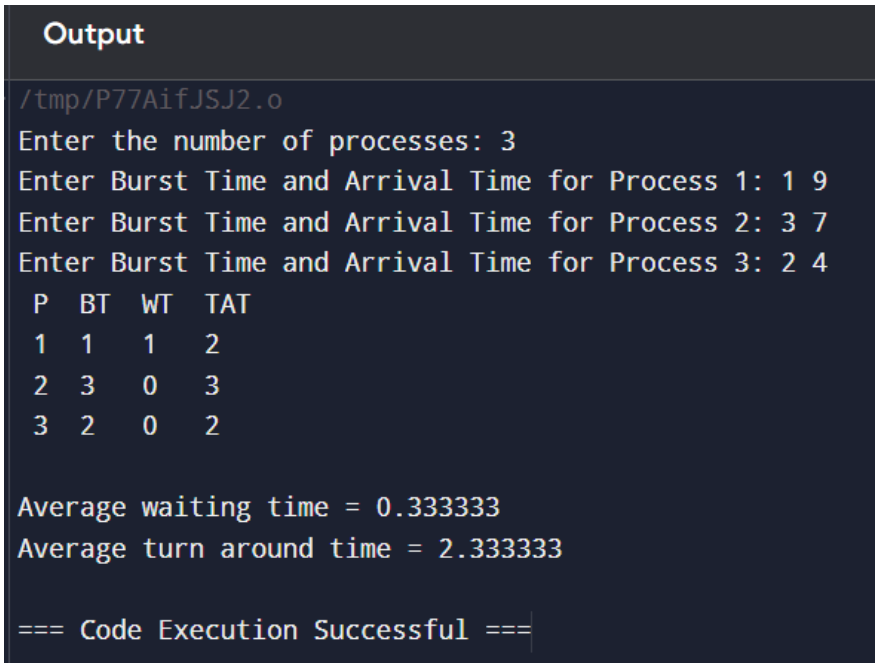


2.SHORTEST JOB FIRST(SJF) - PREEMPTIVE/SRTF

```
#include <stdio.h>
#include <limits.h>
struct Process {
   int pid;
   int bt;
   int art;
};
void findWaitingTime(struct Process proc[], int n, int wt[]) {
   int rt[n];
   for (int i = 0; i < n; i++)
      rt[i] = proc[i].bt;
   int complete = 0, t = 0, minm = INT_MAX;
   int shortest = 0, finish_time;
   int check = 0;
   while (complete != n) {
      for (int j = 0; j < n; j++) {
         if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0) {
            minm = rt[j];
            shortest = j;
            check = 1;
         }
      }
      if (check == 0) {
         t++;
         continue;
      }
```

```
        rt[shortest]--;
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;
        if (rt[shortest] == 0) {
            complete++;
            check = 0;
            finish_time = t + 1;
            wt[shortest] = finish_time - proc[shortest].bt - proc[shortest].art;
            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        t++;
    }
}
void findTurnAroundTime(struct Process proc[], int n, int wt[], int tat[]) {
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
void findavgTime(struct Process proc[], int n) {
    int wt[n], tat[n], total_wt = 0, total_tat = 0;
    findWaitingTime(proc, n, wt);
    findTurnAroundTime(proc, n, wt, tat);
    printf(" P\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        total_wt += wt[i];
        total_tat += tat[i];
        printf(" %d\t%d\t%d\t%d\n", proc[i].pid, proc[i].bt, wt[i], tat[i]);
    }
    printf("\nAverage waiting time = %f", (float)total_wt / n);
    printf("\nAverage turn around time = %f", (float)total_tat / n);
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    struct Process proc[n];
    for (int i = 0; i < n; i++) {
        proc[i].pid = i + 1;
        printf("Enter Burst Time and Arrival Time for Process %d: ", i + 1);
        scanf("%d %d", &proc[i].bt, &proc[i].art);
    }
    findavgTime(proc, n);
    return 0;
}
```

OUTPUT:



3.SHORTEST JOB FIRST(SJF) – NON-PREEMPTIVE

```
#include <stdio.h>
int main() {
   int A[100][4];
   int i, j, n, total = 0, index, temp;
   float avg_wt, avg_tat;
   printf("Enter number of process: ");
   scanf("%d", &n);
   printf("Enter Burst Time:\n");
   for (i = 0; i < n; i++) {
      printf("P%d: ", i + 1);
      scanf("%d", &A[i][1]);
      A[i][0] = i + 1;
   }
   for (i = 0; i < n; i++) {
      index = i;
      for (j = i + 1; j < n; j++) {
         if (A[j][1] < A[index][1]) {
            index = j;
         }
      }
      temp = A[i][1];
      A[i][1] = A[index][1];
      A[index][1] = temp;
      temp = A[i][0];
```

```
        A[i][0] = A[index][0];
        A[index][0] = temp;
    }
    A[0][2] = 0;
    for (i = 1; i < n; i++) {
        A[i][2] = 0;
        for (j = 0; j < i; j++) {
            A[i][2] += A[j][1];
        }
        total += A[i][2];
    }
    avg_wt = (float)total / n;
    total = 0;
    printf("P\t BT\t WT\t TAT\n");
    for (i = 0; i < n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        total += A[i][3];
        printf("P%d\t %d\t %d\t %d\n", A[i][0], A[i][1], A[i][2], A[i][3]);
    }
    avg_tat = (float)total / n;
    printf("Average Waiting Time= %f", avg_wt);
    printf("\nAverage Turnaround Time= %f", avg_tat);
    return 0;
}
```

OUTPUT:

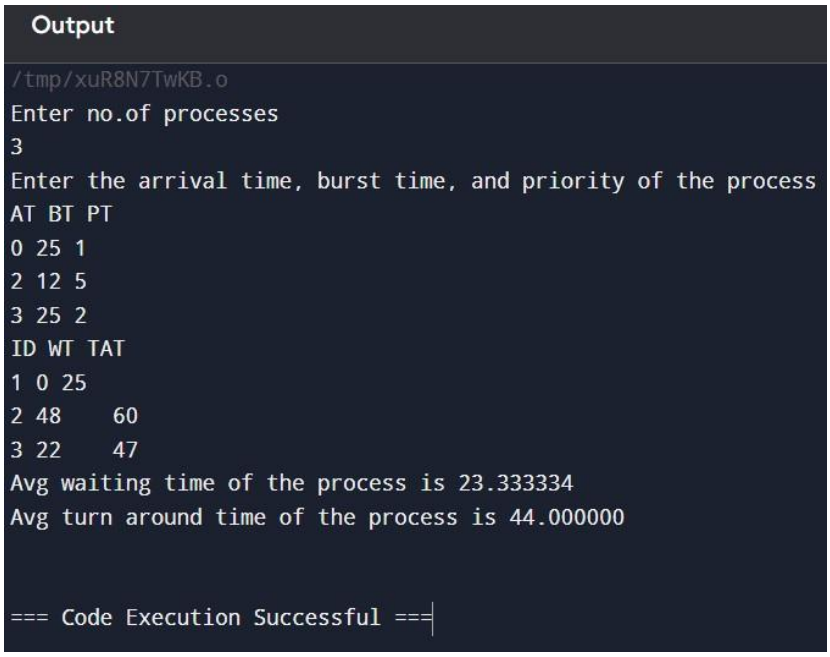
```
Output
/tmp/Dxpkvxypj1.o
Enter number of process: 3
Enter Burst Time:
P1: 24
P2: 5
P3: 7
P     BT   WT   TAT
P2    5    0    5
P3    7    5    12
P1    24   12   36
Average Waiting Time= 5.666667
Average Turnaround Time= 17.666666

=== Code Execution Successful ===
```

4.PRIORITY SCHEDULING - PREEMPTIVE

```
#include<stdio.h>
struct process {
   int WT, AT, BT, TAT, PT;
};
struct process a[10];
int main() {
   int n, temp[10], t, count = 0, short_p;
   float total_WT = 0, total_TAT = 0, Avg_WT, Avg_TAT;
   printf("Enter no.of processes\n");
   scanf("%d", &n);
   printf("Enter the arrival time, burst time, and priority of the process\n");
   printf("AT BT PT\n");
   for (int i = 0; i < n; i++) {
      scanf("%d%d%d", &a[i].AT, &a[i].BT, &a[i].PT);
      temp[i] = a[i].BT;
   }
   a[9].PT = 10000;
   for (t = 0; count != n; t++) {
      short_p = 9;
      for (int i = 0; i < n; i++) {
         if (a[short_p].PT > a[i].PT && a[i].AT <= t && a[i].BT > 0) {
            short_p = i;
         }
      }
      a[short_p].BT = a[short_p].BT - 1;
      if (a[short_p].BT == 0) {
         count++;
         a[short_p].WT = t + 1 - a[short_p].AT - temp[short_p];
         a[short_p].TAT = t + 1 - a[short_p].AT;
         total_WT = total_WT + a[short_p].WT;
         total_TAT = total_TAT + a[short_p].TAT;
      }
   }
   Avg_WT = total_WT / n;
   Avg_TAT = total_TAT / n;
   printf("ID WT TAT\n");
   for (int i = 0; i < n; i++) {
      printf("%d %d\t%d\n", i + 1, a[i].WT, a[i].TAT);
   }
   printf("Avg waiting time of the process is %f\n", Avg_WT);
   printf("Avg turn around time of the process is %f\n", Avg_TAT);
   return 0;
}
```

OUTPUT:



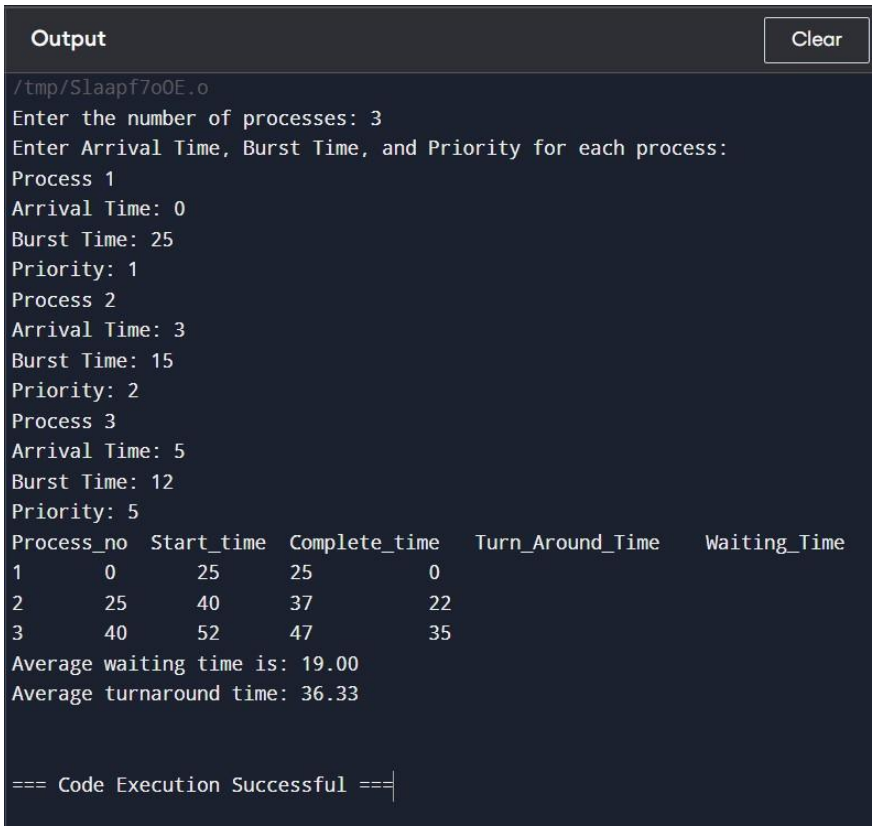
5.PRIORITY SCHEDULING – NON-PREEMPTIVE

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_PROCESS 50
struct process {
   int at, bt, pr, pno;
};
struct  process  proc[MAX_PROCESS];
int comp(const void* a, const void* b) {
   struct process* p1 = (struct process*)a;
   struct process* p2 = (struct process*)b;
   if (p1->at == p2->at) {
      return p1->pr - p2->pr;
   } else {
      return p1->at - p2->at;
   }
}
void get_wt_time(int wt[], int n) {
   int service[MAX_PROCESS];
   service[0] = proc[0].at;
   wt[0] = 0;
   for (int i = 1; i < n; i++) {
      service[i] = proc[i - 1].bt + service[i - 1];
      wt[i] = service[i] - proc[i].at;
      if (wt[i] < 0) wt[i] = 0;
```

```
    }
}
void get_tat_time(int tat[], int wt[], int n) {
    for (int i = 0; i < n; i++) {
        tat[i] = proc[i].bt + wt[i];
    }
}
void findgc(int n) {
    int wt[MAX_PROCESS], tat[MAX_PROCESS];
    double wavg = 0, tavg = 0;
    get_wt_time(wt, n);
    get_tat_time(tat, wt, n);
    int stime[MAX_PROCESS], ctime[MAX_PROCESS];
    stime[0] = proc[0].at;
    ctime[0] = stime[0] + tat[0];
    for (int i = 1; i < n; i++) {
        stime[i] = ctime[i - 1];
        ctime[i] = stime[i] + tat[i] - wt[i];
    }
    printf("Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaiting_Time\n");
    for (int i = 0; i < n; i++) {
        wavg += wt[i];
        tavg += tat[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n", proc[i].pno, stime[i], ctime[i], tat[i], wt[i]);
    }
    printf("Average waiting time is: %.2f\n", wavg / n);
    printf("Average turnaround time: %.2f\n", tavg / n);
}
int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    if (n > MAX_PROCESS) {
        printf("Number of processes exceeds maximum limit.\n");
        return 1;
    }
    printf("Enter Arrival Time, Burst Time, and Priority for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Arrival Time: ");
        scanf("%d", &proc[i].at);
        printf("Burst Time: ");
        scanf("%d", &proc[i].bt);
        printf("Priority: ");
        scanf("%d", &proc[i].pr);
```

```
    proc[i].pno = i + 1;
  }
  qsort(proc, n, sizeof(struct process), comp);
  findgc(n);
  return 0;
}
```

OUTPUT:

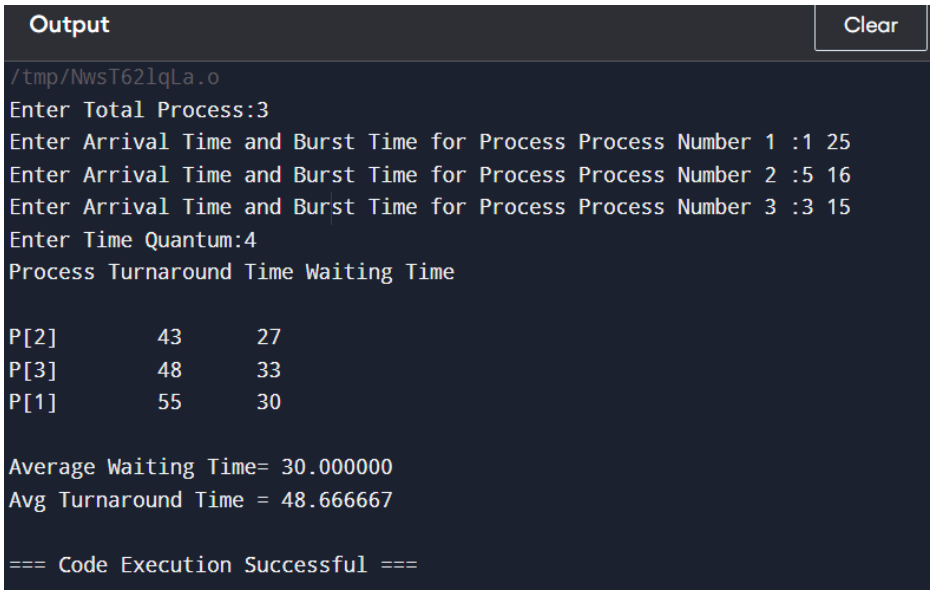


6.ROUND ROBIN

```
#include<stdio.h>
int main()
{
 int cnt,j,n,t,remain,flag=0,tq;
 int wt=0,tat=0,at[10],bt[10],rt[10];
 printf("Enter Total Process:");
 scanf("%d",&n);
 remain=n;
 for(cnt=0;cnt<n;cnt++)
 {
  printf("Enter Arrival Time and Burst Time for Process Process Number %d :",cnt+1);
  scanf("%d",&at[cnt]);
```

```
  scanf("%d",&bt[cnt]);
  rt[cnt]=bt[cnt];
 }
 printf("Enter Time Quantum:");
 scanf("%d",&tq);
 printf("Process\tTurnaround Time Waiting Time\n\n");
 for(t=0,cnt=0;remain!=0;)
 {
  if(rt[cnt]<=tq && rt[cnt]>0)
  {
   t+=rt[cnt];
   rt[cnt]=0;
   flag=1;
  }
  else if(rt[cnt]>0)
  {
   rt[cnt]-=tq;
   t+=tq;
  }
  if(rt[cnt]==0 && flag==1)
  {
   remain--;
   printf("P[%d]\t\t%d\t\t%d\n",cnt+1,t-at[cnt],t-at[cnt]-bt[cnt]);
   wt+=t-at[cnt]-bt[cnt];
   tat+=t-at[cnt];
   flag=0;
  }
  if(cnt==n-1)
   cnt=0;
  else if(at[cnt+1]<=t)
   cnt++;
  else
   cnt=0;
 }
 printf("\nAverage Waiting Time= %f\n",wt*1.0/n);
 printf("Avg Turnaround Time = %f",tat*1.0/n);
 return 0;
}
```

OUTPUT:


Output
Clear
/tmp/NwsT62lqLa.o
Enter Total Process:3
Enter Arrival Time and Burst Time for Process Process Number 1 :1 25
Enter Arrival Time and Burst Time for Process Process Number 2 :5 16
Enter Arrival Time and Burst Time for Process Process Number 3 :3 15
Enter Time Quantum:4
Process Turnaround Time Waiting Time
P[2] 43 27
P[3] 48 33
P[1] 55 30
Average Waiting Time= 30.000000
Avg Turnaround Time = 48.666667
=== Code Execution Successful ===