

OS LAB -6

• Name :HITU RAJ

• Roll no. :2005025

• Branch :CSE

Q1.FIRST COME FIRST SERVE(FCFS)

#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[])

{

    int service\_time[n];

    service\_time[0] = at[0];

    wt[0] = 0; // calculating waiting time

    for (int i = 1; i < n; i++)

    { // Add burst time of previous processes

        service\_time[i] = service\_time[i - 1] + bt[i - 1];

        // Find waiting time for current process = sum - at[i]

        wt[i] = service\_time[i] - at[i];

        // If waiting time for a process is in negative that means it is already in the ready queue before CPU becomes idle so its waiting time is 0

        if (wt[i] < 0)

            wt[i] = 0;

    }

} // Function to calculate turn around time

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])

{ // Calculating turnaround time by adding bt[i] + wt[i]

    for (int i = 0; i < n; i++)

        tat[i] = bt[i] + wt[i];

}

// Function to calculate response time

void findResponseTime(int processes[], int n, int rt[], int wt[], int at[])

{ // Calculating response time by adding at[i] + wt[i]

    for (int i = 0; i < n; i++)

        rt[i] = wt[i] + at[i];

} // Function to calculate average waiting and turn-around times.

void findavgTime(int processes[], int n, int bt[], int at[])

{

    int wt[n], tat[n], rt[n];                      // Function to find waiting time of all processes

    findWaitingTime(processes, n, bt, wt, at);     // Function to find turn around time for all processes

    findTurnAroundTime(processes, n, bt, wt, tat); // Function to find response time for all processes

    findResponseTime(processes, n, rt, wt, at);    // Display processes along with all details

    printf("Process\t BT \tAT \tWT \tTAT \tCT \tRT\n");

    int total\_wt = 0, total\_tat = 0, total\_rt = 0;

    for (int i = 0; i < n; i++)

    {

        total\_wt = total\_wt + wt[i];

        total\_tat = total\_tat + tat[i];

        total\_rt = total\_rt + rt[i];

        int compl\_time = tat[i] + at[i];

        printf("%d\t %d\t %d\t %d\t %d\t %d\t %d\n", i + 1, bt[i], at[i], wt[i], tat[i], compl\_time, rt[i]);

    }

    printf("Average waiting time = %f\n", (float)total\_wt / n);

    printf("Average turn around time = %f\n", (float)total\_tat / n);

    printf("Average response time = %f\n", (float)total\_rt / n);

}

int main()

{

    int n = 0;

    printf("Enter the no. of process : ");

    scanf("%d", &n); // process id's , Burst time , arrival time of all processes

    int processes[n], burst\_time[n], arrival\_time[n];

    printf("Enter the process id's, burst time and arrival time : \n");

    for (int i = 0; i < n; i++)

    {

        printf("%d entry : ", i + 1);

        scanf("%d%d%d", &processes[i], &burst\_time[i], &arrival\_time[i]);

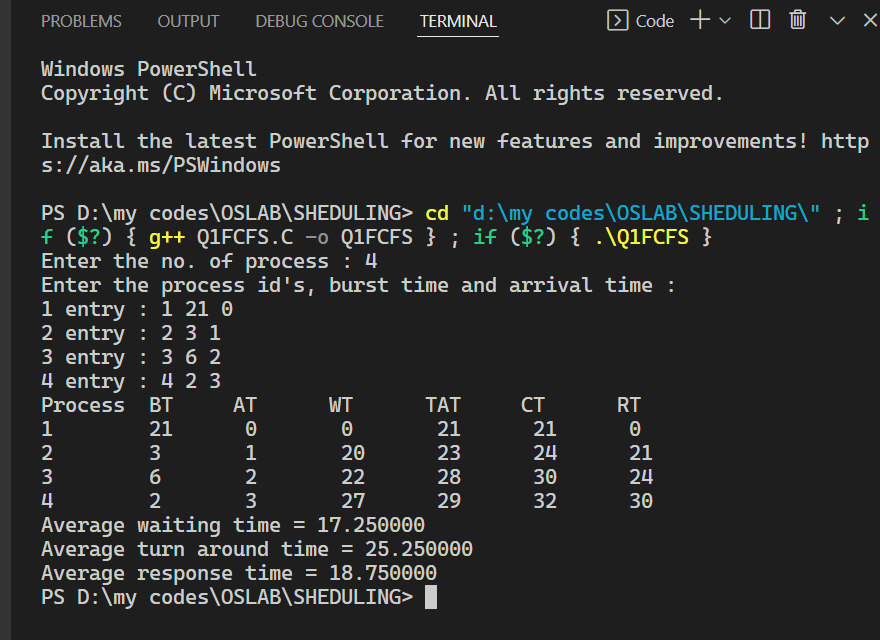
    }

    findavgTime(processes, n, burst\_time, arrival\_time);

    return 0;

}

**OUTPUT -1**

****

**Q2. SHORTEST JOB FIRST (SJF)**

// Question 2 : Shortest Job First(SJF) Scheduling

#include <stdio.h>

int main()

{

    int i, n, p[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, min, k = 0, btime = 0;

    int bt[10], temp, j, at[10], wt[10], tt[10], ta = 0, sum = 0;

    float wavg = 0, tavg = 0, tsum = 0, wsum = 0;

    printf("SJF (NP)\n");

    printf("\nEnter the No. of processes : ");

    scanf("%d", &n);

    for (i = 0; i < n; i++)

    {

        printf("Enter the burst time & arrival time of %d process : ", i + 1);

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

    }

    // Sorting According to Arrival Time

    // for (i = 0; i < n; i++)

    // {

    // for (j = 0; j < n; j++)

    // { // if (at[i] < at[j])

    // {

    // temp = p[j];

    // p[j] = p[i];

    // p[i] = temp;

    // temp = at[j];

    // at[j] = at[i];

    // at[i] = temp;

    // temp = bt[j];

    // bt[j] = bt[i];

    // bt[i] = temp;

    // }

    // }

    // }

    /\* Arranging the table according to Burst time, Execution time and Arrival Time Arrival time <= Execution time \*/

    for (j = 0; j < n; j++)

    {

        btime = btime + bt[j];

        min = bt[1];

        for (i = k; i < n; i++)

        {

            if (btime >= at[i] && bt[i] < min)

            {

                temp = p[k];

                p[k] = p[i];

                p[i] = temp;

                temp = at[k];

                at[k] = at[i];

                at[i] = temp;

                temp = bt[k];

                bt[k] = bt[i];

                bt[i] = temp;

            }

        }

        k++;

    }

    wt[0] = 0;

    for (i = 1; i < n; i++)

    {

        sum = sum + bt[i - 1];

        wt[i] = sum - at[i];

        wsum = wsum + wt[i];

    }

    wavg = (wsum / n);

    for (i = 0; i < n; i++)

    {

        ta = ta + bt[i];

        tt[i] = ta - at[i];

        tsum = tsum + tt[i];

    }

    tavg = (tsum / n);

    printf("\nProcess\t\tBurst\t\tArrival\t\tWaiting\t\tTurn-around");

    for (i = 0; i < n; i++)

    {

        printf("\n p%d\t\t %d\t\t %d\t\t %d\t\t %d", p[i], bt[i], at[i], wt[i], tt[i]);

    }

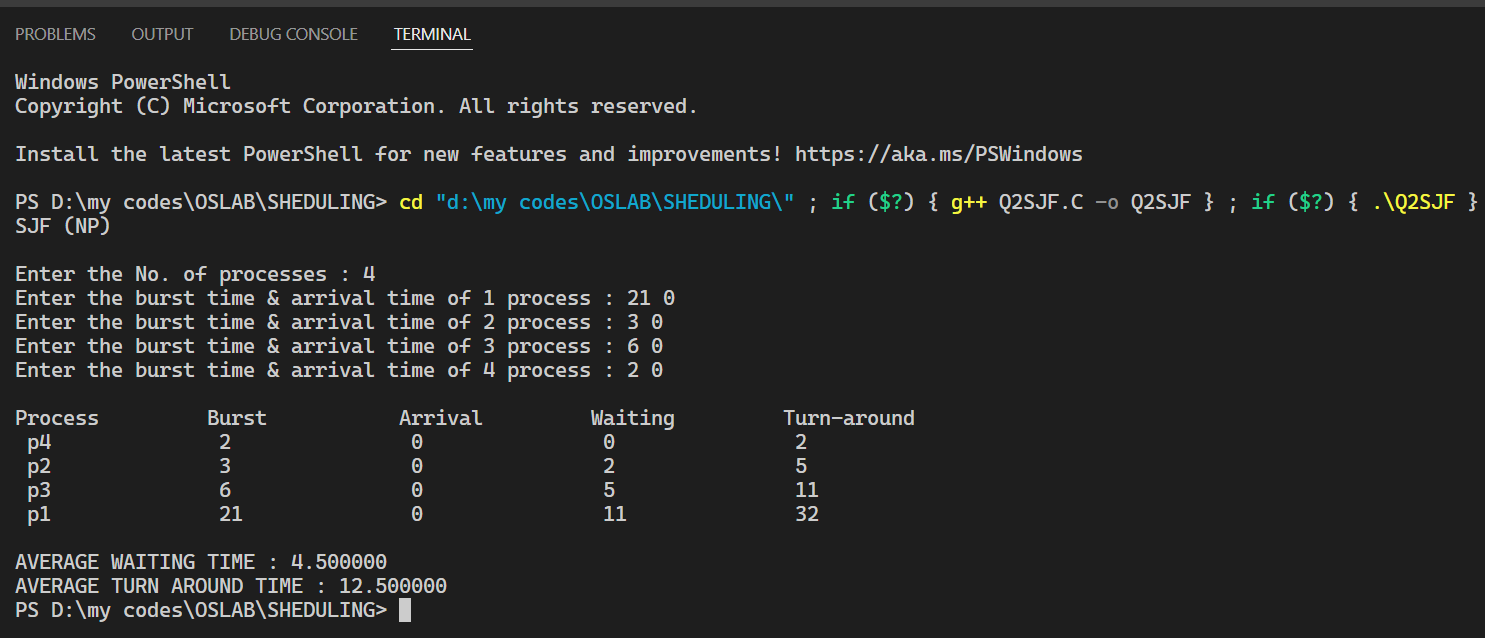
    printf("\n\nAVERAGE WAITING TIME : %f", wavg);

    printf("\nAVERAGE TURN AROUND TIME : %f \n", tavg);

    return 0;

}

**OUTPUT-2**

****

**Q3-ROUND ROBIN**

// Question 3 : Round -Robin Scheduling

#include <stdio.h>

struct process

{

    char name;

    int at, bt, wt, tt, rt;

    int completed;

    float ntt;

} p[10];

int n;

int q[10]; // queue

int front = -1, rear = -1;

void enqueue(int i)

{

    if (rear == 10)

        printf("overflow");

    rear++;

    q[rear] = i;

    if (front == -1)

        front = 0;

}

int dequeue()

{

    if (front == -1)

        printf("underflow");

    int temp = q[front];

    if (front == rear)

        front = rear = -1;

    else

        front++;

    return temp;

}

int isInQueue(int i)

{

    int k;

    for (k = front; k <= rear; k++)

    {

        if (q[k] == i)

            return 1;

    }

    return 0;

}

void sortByArrival()

{

    struct process temp;

    int i, j;

    for (i = 0; i < n - 1; i++)

    {

        for (j = i + 1; j < n; j++)

        {

            if (p[i].at > p[j].at)

            {

                temp = p[i];

                p[i] = p[j];

                p[j] = temp;

            }

        }

    }

}

int main()

{

    int i, j, time = 0, sum\_bt = 0, tq;

    int c;

    float avgwt = 0, avgtt = 0;

    printf("Enter no of processes : ");

    scanf("%d", &n);

    for (i = 0, c = 1; i < n; i++, c++)

    {

        p[i].name = c;

        printf("\nEnter the arrival time and burst time of process %d: ", i + 1);

        scanf("%d%d", &p[i].at, &p[i].bt);

        p[i].rt = p[i].bt;

        p[i].completed = 0;

        sum\_bt += p[i].bt;

    }

    printf("\nEnter the time quantum : ");

    scanf("%d", &tq);

    sortByArrival(); // sorting on the basis of arrival time

    enqueue(0);      // enqueue the first

    printf("Process execution order : ");

    for (time = p[0].at; time < sum\_bt;) // run until the total burst timereached

    {

        i = dequeue();

        if (p[i].rt <= tq)

        { /\* for processes having remaining time with less than or equal to time quantum \*/

            time += p[i].rt;

            p[i].rt = 0;

            p[i].completed = 1;

            printf(" %d ", p[i].name);

            p[i].wt = time - p[i].at - p[i].bt;

            p[i].tt = time - p[i].at;

            p[i].ntt = ((float)p[i].tt / p[i].bt);

            for (j = 0; j < n; j++) /\*enqueue the processes which have come while scheduling \*/

            {

                if (p[j].at <= time && p[j].completed != 1 && isInQueue(j) != 1)

                {

                    enqueue(j);

                }

            }

        }

        else // more than time quantum

        {

            time += tq;

            p[i].rt -= tq;

            printf(" %d ", p[i].name);

            for (j = 0; j < n; j++) /\*first enqueue the processes which have come while scheduling \*/

            {

                if (p[j].at <= time && p[j].completed != 1 && i != j && isInQueue(j) != 1)

                {

                    enqueue(j);

                }

            }

            enqueue(i); // then enqueue the uncompleted process

        }

    }

    printf("\nName\tArrival Time\tBurst Time\tWaiting Time\tTurnAround Time\n");

    for (i = 0; i < n; i++)

    {

        avgwt += p[i].wt;

        avgtt += p[i].tt;

        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t", p[i].name, p[i].at, p[i].bt, p[i].wt, p[i].tt);

    }

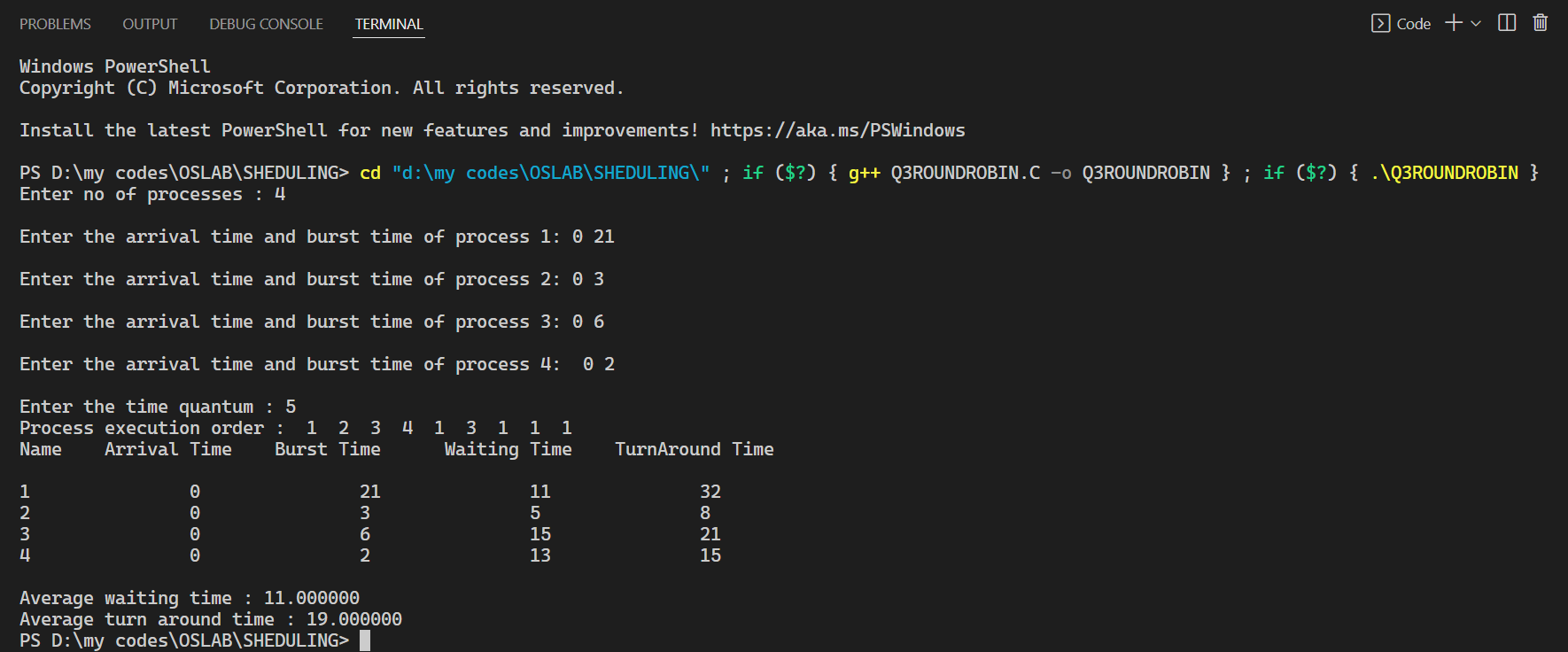
    printf("\n\nAverage waiting time : %f\n", avgwt / n);

    printf("Average turn around time : %f\n", avgtt / n);

    return 0;

}

**OUTPUT -3**

****

**Q4A PRIORITY SCHEDULING (PREMPTIVE)**

//   Question 4 (A): Priority Scheduling Pre -emptive

#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 the number of the process : ");

    scanf("%d", &n);

    printf("Enter the burst time, priority and arrival time of the process\n");

    printf("BT PT AT\n");

    for (int i = 0; i < n; i++)

        scanf("%d%d%d", &a[i].BT, &a[i].PT, &a[i].AT); // copying the burst time in

        // a temp array fot futher use

        temp[i] = a[i].BT;

    }

    // we initialize the priority

    // of a process with maximum

    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 any process is completed

        if (a[short\_p].BT == 0)

        { // one process is completed

            // so count increases by 1

            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 calculation

            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; // printing of the answer

    printf("\nProcess.ID\tWaiting Time\tTurn Around Time\n");

    for (int i = 0; i < n; i++)

    {

        printf("%d\t\t%d\t\t%d\n", i + 1, a[i].WT, a[i].TAT);

    }

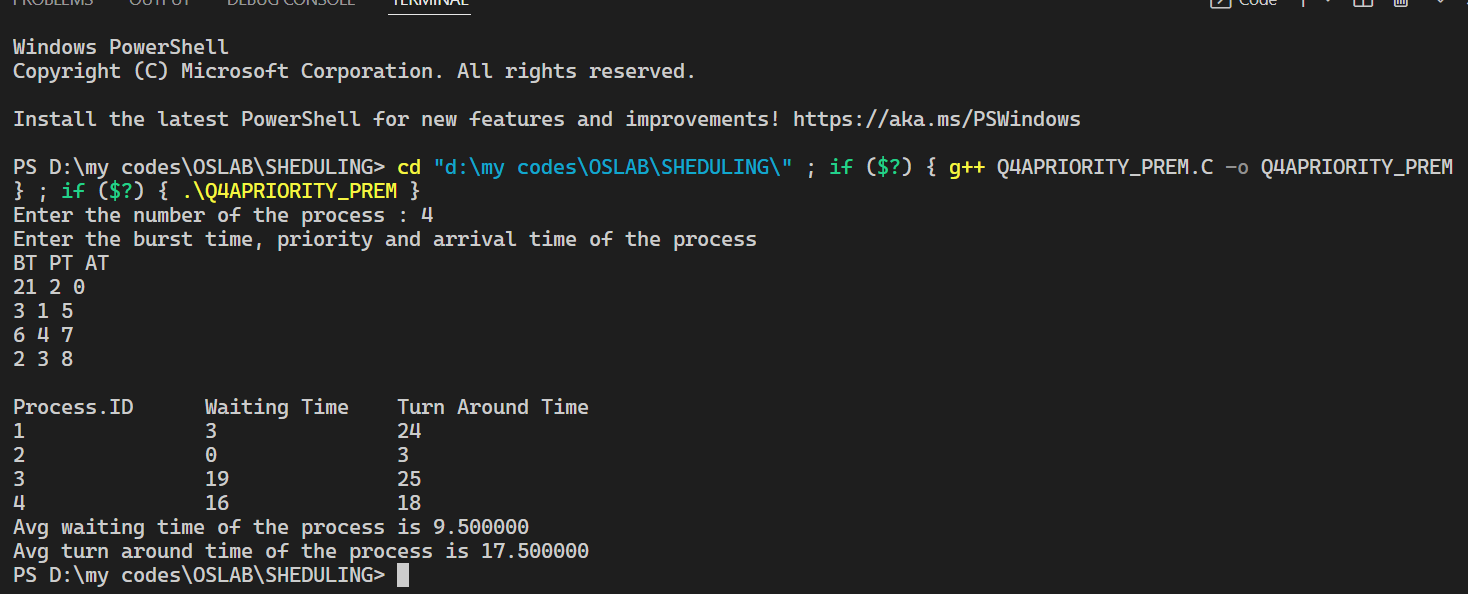
    printf("Avg waiting time of the process is %f\n", Avg\_WT);

    printf("Avg turn around time of the process is %f\n", Avg\_TAT);

    return 0;

}

**OUTPUT 4(A)**

****

**Q4B PRIORITY SCHEDULING (NON-PREMPTIVE)**

// (4B)Non Pre -emptive

#include<stdio.h>

 struct process

{

    int id, WT, AT, BT, TAT, PR;

};

struct process a[10]; // function for swapping

void swap(int \*b, int \*c)

{

    int tem;

    tem = \*c;

    \*c = \*b;

    \*b = tem;

}

int main()

{

    int n, check\_ar = 0;

    int Cmp\_time = 0;

    float Total\_WT = 0, Total\_TAT = 0, Avg\_WT, Avg\_TAT;

    printf("Enter the number of process: ");

    scanf("%d", &n);

    printf("Enter the burst time, priority and arrival time of the process\n");

    printf("BT PT AT\n");

    for (int i = 0; i < n; i++)

    {

        scanf("%d%d%d", &a[i].BT, &a[i].PR, &a[i].AT);

        a[i].id = i + 1; // here we are checking that arrival time // of the process are same or different

        if (i == 0)

            check\_ar = a[i].AT;

        if (check\_ar != a[i].AT)

            check\_ar = 1;

    } // if process are arrived at the different time // then sort the process on the basis of AT

    if (check\_ar != 0)

    {

        for (int i = 0; i < n; i++)

        {

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

            {

                if (a[j].AT > a[j + 1].AT)

                {

                    swap(&a[j].id, &a[j + 1].id);

                    swap(&a[j].AT, &a[j + 1].AT);

                    swap(&a[j].BT, &a[j + 1].BT);

                    swap(&a[j].PR, &a[j + 1].PR);

                }

            }

        }

    } // logic of Priority scheduling ( non preemptive) algo // if all the process are arrived at different time

    if (check\_ar != 0)

    {

        a[0].WT = a[0].AT;

        a[0].TAT = a[0].BT - a[0].AT; // cmp\_time for completion time

        Cmp\_time = a[0].TAT;

        Total\_WT = Total\_WT + a[0].WT;

        Total\_TAT = Total\_TAT + a[0].TAT;

        for (int i = 1; i < n; i++)

        {

            int min = a[i].PR;

            for (int j = i + 1; j < n; j++)

            {

                if (min > a[j].PR && a[j].AT <= Cmp\_time)

                {

                    min = a[j].PR;

                    swap(&a[i].id, &a[j].id);

                    swap(&a[i].AT, &a[j].AT);

                    swap(&a[i].BT, &a[j].BT);

                    swap(&a[i].PR, &a[j].PR);

                }

            }

            a[i].WT = Cmp\_time - a[i].AT;

            Total\_WT = Total\_WT + a[i].WT; // completion time of the process

            Cmp\_time = Cmp\_time + a[i].BT; // Turn Around Time of the process // compl -Arival

            a[i].TAT = Cmp\_time - a[i].AT;

            Total\_TAT = Total\_TAT + a[i].TAT;

        }

    }

    // if all the process are arrived at same time

    else

    {

        for (int i = 0; i < n; i++)

        {

            int min = a[i].PR;

            for (int j = i + 1; j < n; j++)

            {

                if (min > a[j].PR && a[j].AT <= Cmp\_time)

                {

                    min = a[j].PR;

                    swap(&a[i].id, &a[j].id);

                    swap(&a[i].AT, &a[j].AT);

                    swap(&a[i].BT, &a[j].BT);

                    swap(&a[i].PR, &a[j].PR);

                }

            }

            a[i].WT = Cmp\_time - a[i].AT;  // completion time of the process

            Cmp\_time = Cmp\_time + a[i].BT; // Turn Around Time of the process

                                           // compl -Arrival

            a[i].TAT = Cmp\_time - a[i].AT;

            Total\_WT = Total\_WT + a[i].WT;

            Total\_TAT = Total\_TAT + a[i].TAT;

        }

    }

    Avg\_WT = Total\_WT / n;

    Avg\_TAT = Total\_TAT / n;

    // Printing of the results

    printf("\nProcess.ID\tWaiting Time\tTurn Around Time\n");

    for (int i = 0; i < n; i++)

    {

        printf("%d\t\t%d\t\t%d\n", a[i].id, a[i].WT, a[i].TAT);

    }

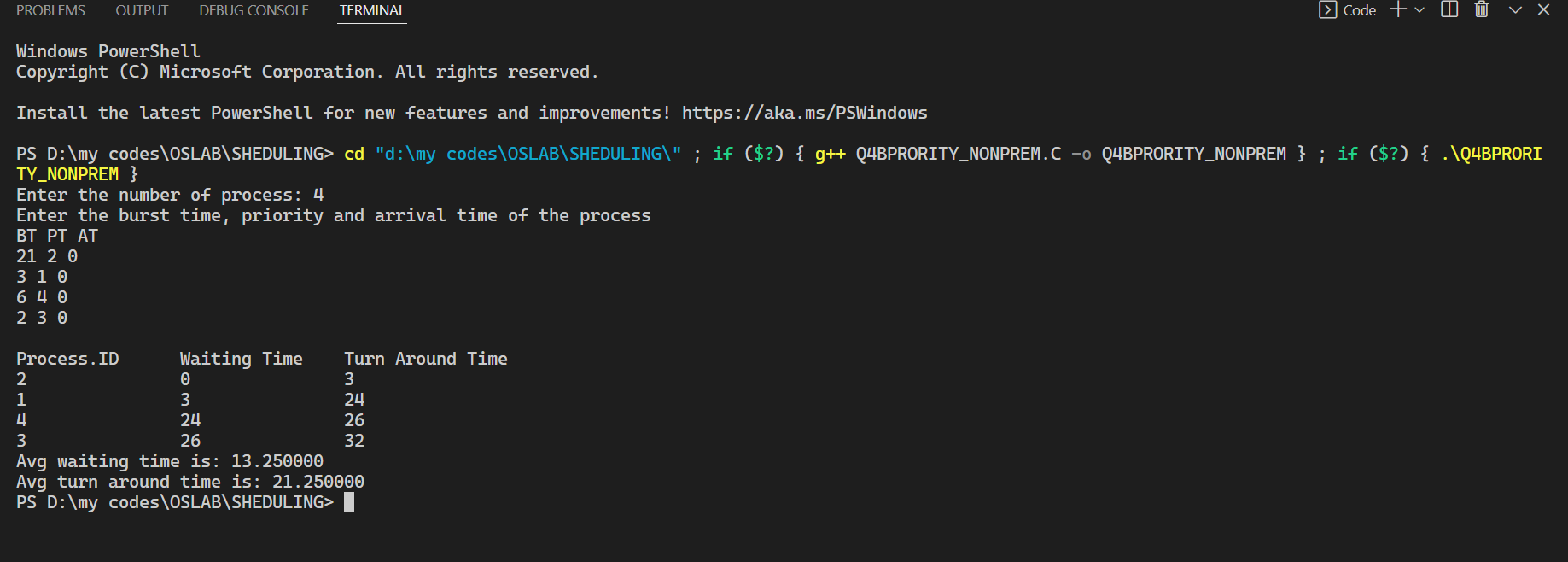
    printf("Avg waiting time is: %f\n", Avg\_WT);

    printf("Avg turn around time is: %f\n", Avg\_TAT);

    return 0;

}

**OUTPUT 4(B)**

****