GARV NANWANI
ROLL NO – 19BCS049
4TH SEMESTER

OPERATING SYSTEM LAB FILE

SUBMITTED TO →
DR SHAHZAD ALAM

DEPARTMENT OF COMPUTER SCIENCE ENGINNERING

JAMIA MILLIA ISLAMIA

TABLE OF CONTENTS

PROGRAM 1

CODE →

5

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    int data;
    int priority;
    struct node* next;
} Node;
Node* newNode(int d, int p)
{
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = d;
    temp->priority = p;
    temp->next = NULL;
    return temp;
}

void traversal(Node** head)
{
    Node* ptr = *head;
    while (ptr != NULL)
    {
        printf("| p%d | %d |  --> ", ptr->data, ptr->priority);
        ptr = ptr->next;
    }
    printf("\n");
}
void pop(Node** head)
{
    if(*head==NULL)
    printf("No process pending\n");
```

```
    else{
    Node* temp = *head;
    (*head) = (*head)->next;
    free(temp);
    }
}
void push(Node** head, int d, int p)
{
    Node* start = (*head);
    Node* temp = newNode(d, p);
    if ((*head)->priority > p) {
        temp->next = *head;
        (*head) = temp;
    }
    else {
        while (start->next != NULL &&
            start->next->priority < p) {
            start = start->next;
        }
        temp->next = start->next;
        start->next = temp;
    }
}
int main()
{
    int x;
    int p,d=0;
    printf("Enter the first process data\n");
    scanf("%d",&d);
    printf("Enter the first process priority\n");
    scanf("%d",&p);
```

```
Node* pq = newNode(d,p);
int num = 1;
while(num>0)
{
    printf("Enter 1 to enter a process\n");
    printf("Enter 2 to execute a process\n");
    printf("Enter 3 to disply all the process\n");
    printf("Enter 4 to exit\n");
    printf("Enter your choice\n");
    scanf("%d",&x);
    switch(x)
    {
        case 1:
        printf("Enter the process data\n");
        scanf("%d",&d);
        printf("Enter the process priority\n");
        scanf("%d",&p);
        push(&pq, d, p);
        break;

        case 2: pop(&pq);
        break;

        case 3: traversal(&pq);
        break;

        case 4: num =0;
        break;

        default: printf("Choice other than 1, 2 and 3\n");
        break;
```

```
        }


    }
    return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program1 (main) 🍔
➜ ./program1
Enter the first process data
5
Enter the first process priority
6
Enter 1 to enter a process
Enter 2 to execute a process
Enter 3 to disply all the process
Enter 4 to exit
Enter your choice
1
Enter the process data
4
Enter the process priority
5
Enter 1 to enter a process
Enter 2 to execute a process
Enter 3 to disply all the process
Enter 4 to exit
Enter your choice
3
| p4 | 5 |  --> | p5 | 6 |  -->
Enter 1 to enter a process
Enter 2 to execute a process
Enter 3 to disply all the process
Enter 4 to exit
Enter your choice
4
```

PROGRAM 2

CODE →

```
#include<stdio.h>

int main(){

	int st[10]={0}, bt[10]={0}, at[10]={0}, tat[10]={0}, wt[10]={0},
ct[10]={0}, rt[10]={0};
	int n,sum=0;
	float totalTAT=0,totalWT=0;
	printf("Enter number of processes ");
	scanf("%d",&n);
	printf("Enter arrival time and burst time for each process\n\n");
	for(int i=0;i<n;i++) {
		printf("Arrival time of process[%d] ",i+1);
		scanf("%d",&at[i]);
		printf("Burst time of process[%d] ",i+1);
		scanf("%d",&bt[i]);
		printf("\n");
	}

	for(int j=0; j<n; j++) {
		sum+=bt[j];
		ct[j]+=sum;
	}

	for(int k=0; k<n; k++) {
		tat[k]=ct[k]-at[k];
		totalTAT+=tat[k];
```

```
        }

        for(int k=0; k<n; k++) {
                wt[k]=tat[k]-bt[k];
                totalWT+=wt[k];
        }
    // rt[0] = 0;
    // int cur = bt[0];
    // for(int k=1; k<n; k++) {
    //     rt[k] = cur - at[k];
    //     cur += bt[k];
    // }

  for(int i=0; i<n; i++)
    {
        if(i==0)
            st[i]=at[i];
        else
            st[i]=ct[i-1];

        rt[i] = st[i] - at[i];
    }

        printf("Solution: \n\n");
        printf("P#\t AT\t BT\t CT\t TAT\t WT\t RT\t\n\n");

        for(int i=0;i<n;i++) {
                printf("P%d\t %d\t %d\t %d\t %d\t %d\t %d\
n",i+1,at[i],bt[i],ct[i],tat[i],wt[i],rt[i]);
        }
```

```
    printf("\n\nAverage Turnaround Time = %f\n",totalTAT/n);
    printf("Average WT = %f\n\n",totalWT/n);
    return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program2 (main) 🍔
➜ ./program2
Enter number of processes 4
Enter arrival time and burst time for each process

Arrival time of process[1] 0
Burst time of process[1] 8

Arrival time of process[2] 5
Burst time of process[2] 7

Arrival time of process[3] 4
Burst time of process[3] 9

Arrival time of process[4] 7
Burst time of process[4] 5

Solution:

P#      AT      BT      CT      TAT     WT      RT

P1      0       8       8       8       0       0
P2      5       7       15      10      3       3
P3      4       9       24      20      11      11
P4      7       5       29      22      17      17


Average Turnaround Time = 15.000000
Average WT = 7.750000
```

PROGRAM 3

CODE →

```
#include<stdio.h>
int main()
{
    int n;
printf(" -------Shortest Job First Scheduling ( NP )-------\n");
printf("\nEnter the No. of processes :");
scanf("%d",&n);
int bt[n],temp,i,j,at[n],wt[n],ct[n],ta[n],pid[n],f[n];
int st=0,tot=0;
float avgwt=0,avgta=0;
for(i=0;i<n;i++)
{
printf("Enter the arrival time of %d process :",i+1);
scanf(" %d",&at[i]);
printf("Enter the burst time of %d process :",i+1);
scanf(" %d",&bt[i]);
pid[i]=i+1;
f[i]=0;
}
while(1)
        {
            int c=n, min = 999999;

            if (tot == n)
                break;

            for (i=0; i<n; i++)
```

```
        {

            if ((at[i] <= st) && (f[i] == 0) && (bt[i]<min))
            {
                min=bt[i];
                c=i;
            }
        }
        if (c==n)
            st++;
        else
        {
            ct[c]=st+bt[c];
            st+=bt[c];
            ta[c]=ct[c]-at[c];
            wt[c]=ta[c]-bt[c];
            f[c]=1;
            pid[tot] = c + 1;
            tot++;
        }
    }
  for(i=0;i<n;i++)
  {
    avgwt+= wt[i];
    avgta+= ta[i];
  }
printf("************************");
printf("\nProcess      Burst      Arrival      Completion      Waiting
Turn-around" );
for(i=0;i<n;i++)
{
```

```
printf("\np%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t
%d",pid[i],bt[i],at[i],ct[i],wt[i],ta[i]);
}

printf("\n\nAVERAGE WAITING TIME : %f",(avgwt/n));
printf("\nAVERAGE TURN AROUND TIME : %f",(avgta/n));
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program3 (main) 🍔
➜ ./program3
 -------Shortest Job First Scheduling ( NP )-------

Enter the No. of processes :4
Enter the arrival time of 1 process :0
Enter the burst time of 1 process :8
Enter the arrival time of 2 process :5
Enter the burst time of 2 process :6
Enter the arrival time of 3 process :7
Enter the burst time of 3 process :5
Enter the arrival time of 4 process :8
Enter the burst time of 4 process :9
**********************
Process       Burst        Arrival       Completion        Waiting         Turn-around
p1              8              0              8               0               8
p3              6              5              19              8               14
p2              5              7              13              1               6
p4              9              8              28              11              20

AVERAGE WAITING TIME : 5.000000
AVERAGE TURN AROUND TIME : 12.000000%
```

PROGRAM 4

CODE →

```
#include <iostream>
#include <algorithm>
#include <iomanip>
#include <string.h>
using namespace std;

struct process {
    int pid;
    int arrival_time;
    int burst_time;
    int start_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    int response_time;
};

int main() {

    int n;
    struct process p[100];
    float avg_turnaround_time;
    float avg_waiting_time;
    float avg_response_time;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;
    int total_response_time = 0;
```

```
int total_idle_time = 0;
int burst_remaining[100];
int is_completed[100];
memset(is_completed,0,sizeof(is_completed));

cout << setprecision(2) << fixed;

cout<<"Enter the number of processes: ";
cin>>n;

for(int i = 0; i < n; i++) {
    cout<<"Enter arrival time of process "<<i+1<<": ";
    cin>>p[i].arrival_time;
    cout<<"Enter burst time of process "<<i+1<<": ";
    cin>>p[i].burst_time;
    p[i].pid = i+1;
    burst_remaining[i] = p[i].burst_time;
    cout<<endl;
}

int current_time = 0;
int completed = 0;
int prev = 0;

while(completed != n) {
    int idx = -1;
    int mn = 10000000;
    for(int i = 0; i < n; i++) {
        if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
            if(burst_remaining[i] < mn) {
                mn = burst_remaining[i];
```

```
                idx = i;
            }
            if(burst_remaining[i] == mn) {
                if(p[i].arrival_time < p[idx].arrival_time) {
                    mn = burst_remaining[i];
                    idx = i;
                }
            }
        }
    }

    if(idx != -1) {
        if(burst_remaining[idx] == p[idx].burst_time) {
            p[idx].start_time = current_time;
            total_idle_time += p[idx].start_time - prev;
        }
        burst_remaining[idx] -= 1;
        current_time++;
        prev = current_time;

        if(burst_remaining[idx] == 0) {
            p[idx].completion_time = current_time;
            p[idx].turnaround_time = p[idx].completion_time -
p[idx].arrival_time;
            p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
            p[idx].response_time = p[idx].start_time - p[idx].arrival_time;

            total_turnaround_time += p[idx].turnaround_time;
            total_waiting_time += p[idx].waiting_time;
            total_response_time += p[idx].response_time;
```

```
            is_completed[idx] = 1;
            completed++;
        }
    }
    else {
        current_time++;
    }
}

int min_arrival_time = 10000000;
int max_completion_time = -1;
for(int i = 0; i < n; i++) {
    min_arrival_time = min(min_arrival_time,p[i].arrival_time);
    max_completion_time =
max(max_completion_time,p[i].completion_time);
}

avg_turnaround_time = (float) total_turnaround_time / n;
avg_waiting_time = (float) total_waiting_time / n;

cout<<endl<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\
t"<<"\n"<<endl;

for(int i = 0; i < n; i++) {
    cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\
t"<<p[i].start_time<<"\t"<<p[i].completion_time<<"\
t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\
t"<<p[i].response_time<<"\t"<<"\n"<<endl;
}
```

```
    cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
    cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;

}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program4 (main) 🍔
➜ ./program4
Enter the number of processes: 4
Enter arrival time of process 1: 3
Enter burst time of process 1: 4

Enter arrival time of process 2: 5
Enter burst time of process 2: 6

Enter arrival time of process 3: 7
Enter burst time of process 3: 8

Enter arrival time of process 4: 9
Enter burst time of process 4: 4


#P      AT      BT      ST      CT      TAT     WT      RT

1       3       4       3       7       4       0       0

2       5       6       7       13      8       2       2

3       7       8       17      25      18      10      10

4       9       4       13      17      8       4       4

Average Turnaround Time = 9.50
Average Waiting Time = 4.00
```

PROGRAM 5

CODE →

```
#include<stdio.h>

struct times
{
	int p,art,but,wtt,tat,rnt,ct;
};


void sortart(struct times a[],int pro)
{
	int i,j;
	struct times temp;
	for(i=0;i<pro;i++)
	{
		for(j=i+1;j<pro;j++)
		{
			if(a[i].art > a[j].art)
			{
				temp = a[i];
				a[i] = a[j];
				a[j] = temp;
			}
		}
	}
	return;
}
```

```
int main()
{
	int i,j,pro,time,remain,flag=0,ts;
	struct times a[100];
	float avgwt=0,avgtt=0;
	printf("Round Robin Scheduling Algorithm\n");
	printf("Enter Number Of Processes : ");
	scanf("%d",&pro);
	remain=pro;
	for(i=0;i<pro;i++)
	{
		printf("Enter arrival time and Burst time for Process P%d : ",i);
		scanf("%d%d",&a[i].art,&a[i].but);
		a[i].p = i;
		a[i].rnt = a[i].but;
	}
	sortart(a,pro);
	printf("Enter Time Quantum Number : ");
	scanf("%d",&ts);
	printf("\n****************************************\n");
	printf("Gantt Chart\n");
	printf("0");
	for(time=0,i=0;remain!=0;)
	{
		if(a[i].rnt<=ts && a[i].rnt>0)
		{
			time = time + a[i].rnt;
			printf(" -> [P%d] <- %d",a[i].p,time);
			a[i].rnt=0;
			flag=1;
		}
```

```
        else if(a[i].rnt > 0)
        {
            a[i].rnt = a[i].rnt - ts;
            time = time + ts;
            printf(" -> [P%d] <- %d",a[i].p,time);
        }
        if(a[i].rnt==0 && flag==1)
        {
            remain--;
            a[i].tat = time-a[i].art;
            a[i].wtt = time-a[i].art-a[i].but;
            a[i].ct = a[i].tat+a[i].but;
            avgwt = avgwt + time-a[i].art-a[i].but;
            avgtt = avgtt + time-a[i].art;
            flag=0;
        }
        if(i==pro-1)
            i=0;
        else if(a[i+1].art <= time)
            i++;
        else
            i=0;
    }
    printf("\n\n");
    printf("****************************************\n");
    printf("Pro\tArTi\tBuTi\tCt\tTaTi\tWtTi\n");
    printf("****************************************\n");
    for(i=0;i<pro;i++)
    {
        printf("P%d\t%d\t%d\t%d\t%d\t%d\
n",a[i].p,a[i].art,a[i].but,a[i].ct,a[i].tat,a[i].wtt);
```

```
    }
    printf("****************************************\n");
    avgwt = avgwt/pro;
    avgtt = avgtt/pro;
    printf("Average Waiting Time : %.2f\n",avgwt);
    printf("Average Turnaround Time : %.2f\n",avgtt);
    return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program5 (main) 🍔
→ ./program5
Round Robin Scheduling Algorithm
Enter Number Of Processes : 3
Enter arrival time and Burst time for Process P0 : 4
5
Enter arrival time and Burst time for Process P1 : 6
7
Enter arrival time and Burst time for Process P2 : 5
7
Enter Time Quantum Number : 5

****************************************
Gantt Chart
0 -> [P0] <- 5 -> [P2] <- 10 -> [P1] <- 15 -> [P2] <- 17 -> [P1] <- 19

****************************************
Pro      ArTi     BuTi     Ct       TaTi     WtTi
****************************************
P0       4        5        6        1        -4
P2       5        7        19       12       5
P1       6        7        20       13       6
****************************************
Average Waiting Time : 2.33
Average Turnaround Time : 8.67
```

PROGRAM 6

CODE →

```
#include<stdio.h>
#include<string.h>
void sort(int arr[][7],char str[][10], int at, int bt, int pr, char p[], int n, int
m);
void print(int n, char str[][10], int arr[][7] );
void ganttChart(int time[],char gantt[][10], int m, int l);

int main(){
	char process[10], gantt[100][10];
	int time[100];
	int at,bt,n,pr;
	printf("Enter no of process :");
	scanf("%d",&n);
	int arr[n+1][7];
	int temp[n];
	char str[n][10];

	printf("Enter 'process priority arrival_time burst_time' :\n");
	scanf("%s",str[0]);
	scanf("%d",&arr[0][0]);
	scanf("%d",&arr[0][1]);
	scanf("%d",&arr[0][2]);
	for (int i=1; i<n; i++){
		scanf("%s",process);
		scanf("%d",&pr);
		scanf("%d",&at);
		scanf("%d",&bt);
```

```
        int j=0;
        while (j<i && arr[j][1]<=at){
                j++;
        }
        sort(arr,str,at,bt,pr,process,i,j);
}

for (int i=0; i<n; i++){
        arr[i][6]=-1;
        temp[i]=arr[i][2];
}

time[0]=arr[0][1];
int l=1, m=0, cnt=0, t=0;
arr[n][0]=10000;

while (cnt<n){
        int min=n;
        bool flag=false;
        for (int i=0; i<n; i++){
                if (arr[i][1]<=t && temp[i]>0 && arr[i][0]<arr[min][0]){
                        min=i;
                        flag=true;
                }
        }

        if (flag){
                arr[min][3]=t+arr[min][2];
                arr[min][4]=arr[min][3]-arr[min][1];
                arr[min][5]=arr[min][4]-arr[min][2];
                arr[min][6]=t-arr[min][1];
```

```
                temp[min]=0;
                t+=arr[min][2];
                time[l]=t;
                l++;
                strcpy(gantt[m],str[min]);
                m++;
                cnt++;
            }
            else{
                int num=0;
                for (int i=0; i<n; i++){
                    if (temp[i]>0){
                        num=i;
                        break;
                    }
                }
                t=arr[num][1];
                time[l]=t;
                l++;
                strcpy(gantt[m],"lag");
                m++;
            }
        }

        print(n,str,arr);
        ganttChart(time,gantt,m,l);
        return 0;
}

void sort(int arr[][7],char str[][10], int at, int bt, int pr, char p[], int n, int
m){
```

```
	for (int i=n-1; i>=m; i--){
		arr[i+1][0]=arr[i][0];
		arr[i+1][1]=arr[i][1];
		arr[i+1][2]=arr[i][2];
		strcpy(str[i+1],str[i]);
	}
	arr[m][0]=pr;
	arr[m][1]=at;
	arr[m][2]=bt;
	strcpy(str[m],p);
}

void print(int n, char str[][10], int arr[][7] ){
	float avg;
	float sum;
	char title[8][20]={"Process","Priority","Arrival Time","Burst
Time","Completion Time","T.A.T",
						"Waiting Time","Response Time"};

	printf("\n\n");
	for (int i=0; i<8; i++){
		printf("%-20s",title[i]);
	}
	printf("\n");
	for (int i=0; i<n; i++){
		printf("%-20s",str[i]);
		for (int j=0; j<7; j++){
			printf("%-20d",arr[i][j]);
		}
		printf("\n\n");
	}
```

```
	printf("%-80s","Average");
	for (int j=3; j<7; j++){
		sum=0;
		for (int i=0; i<n; i++){
			sum+=arr[i][j];
		}
		avg=sum/n;
		printf("%-20.2f",avg);
	}
	printf("\n\n");
}

void ganttChart(int time[],char gantt[][10], int m, int l){
	printf("Gantt Chart :\n\n");
	printf("|");
	for (int i=0; i<m; i++){
		printf("%-5s|",gantt[i]);
	}
	printf("\n\n");
	for (int i=0; i<l; i++){
		printf("%-6d",time[i]);
	}
}
```

OUTPUT →

```
~/Projects/OS_Lab_Sem4/program6 (main)
→ ./program6
Enter no of process :3
Enter 'process priority arrival_time burst_time' :
p1 2 0 5
p2 4 5 6
p3 3 6 7


Process          Priority         Arrival Time     Burst Time       Completion Time    T.A.T            Waiting Time
   Response Time
p1               2                0                5                5                  5                0
   0

p2               4                5                6                11                 6                0
   0

p3               3                6                7                18                 12               5
   5

Average                                                            11.33              7.67             1.67
   1.67

Gantt Chart :

|p1   |p2   |p3   |

0    5    11   18   %
```

PROGRAM 7

CODE →

```
#include<iostream>
#include<algorithm>
using namespace std;

struct node{
    char pname;
    int btime;
    int atime;
    int priority;
    int restime=0;
    int ctime=0;
    int wtime=0;
}a[1000],b[1000],c[1000];

void insert(int n){
    int i;
    for(i=0;i<n;i++){
        cin>>a[i].pname;
        cin>>a[i].priority;
        cin>>a[i].atime;
        cin>>a[i].btime;
        a[i].wtime=-a[i].atime+1;
    }
}

bool btimeSort(node a,node b){
    return a.btime < b.btime;
```

```
}

bool atimeSort(node a,node b){
    return a.atime < b.atime;
}
bool prioritySort(node a,node b){
    return a.priority < b.priority;
}
int k=0,f=0,r=0;
void disp(int nop,int qt){
    int n=nop,q;
    sort(a,a+n,atimeSort);
    int ttime=0,i;
    int j,tArray[n];
    int alltime=0;
    bool moveLast=false;
    for(i=0;i<n;i++){
        alltime+=a[i].btime;
    }
    alltime+=a[0].atime;
    for(i=0;ttime<=alltime;){
        j=i;
        while(a[j].atime<=ttime&&j!=n){
            b[r]=a[j];
            j++;
            r++;
        }
        if(r==f){
            c[k].pname='i';
            c[k].btime=a[j].atime-ttime;
            c[k].atime=ttime;
```

```
        ttime+=c[k].btime;
        k++;
        continue;
    }
    i=j;
    if(moveLast==true){
        sort(b+f,b+r,prioritySort);
    }

    j=f;
    if(b[j].btime>qt){
        c[k]=b[j];
        c[k].btime=qt;
        k++;
        b[j].btime=b[j].btime-qt;
        ttime+=qt;
        moveLast=true;
        for(q=0;q<n;q++){
            if(b[j].pname!=a[q].pname){
                a[q].wtime+=qt;
            }
        }
    }
    else{
        c[k]=b[j];
        k++;
        f++;
        ttime+=b[j].btime;
        moveLast=false;
        for(q=0;q<n;q++){
            if(b[j].pname!=a[q].pname){
```

```
                a[q].wtime+=b[j].btime;
            }
        }
    }
    if(f==r&&i>=n)
    break;
}
tArray[i]=ttime;
ttime+=a[i].btime;
for(i=0;i<k-1;i++){
    if(c[i].pname==c[i+1].pname){
        c[i].btime+=c[i+1].btime;
        for(j=i+1;j<k-1;j++)
            c[j]=c[j+1];
        k--;
        i--;
    }
}

int rtime=0;
for(j=0;j<n;j++){
    rtime=0;
    for(i=0;i<k;i++){
        if(c[i].pname==a[j].pname){
            a[j].restime=rtime;
            break;
        }
        rtime+=c[i].btime;
    }
}
```

```
float averageWaitingTime=0;
float averageResponseTime=0;
float averageTAT=0;

cout<<"\nGantt Chart\n";
rtime=0;
for (i=0; i<k; i++){
    if(i!=k)
        cout<<"|  "<<'P'<< c[i].pname << "   ";
    rtime+=c[i].btime;
    for(j=0;j<n;j++){
        if(a[j].pname==c[i].pname)
            a[j].ctime=rtime;
    }
}
cout<<"\n";
rtime=0;
for (i=0; i<k+1; i++){
    cout << rtime << "\t";
    tArray[i]=rtime;
    rtime+=c[i].btime;
}

cout<<"\n";
cout<<"\n";
cout<<"P.Name Priority AT\tBT\tCT\tTAT\tWT\tRT\n";
for (i=0; i<nop&&a[i].pname!='i'; i++){
    if(a[i].pname=='\0')
        break;
    cout <<'P'<< a[i].pname << "\t";
    cout << a[i].priority << "\t";
```

```
        cout << a[i].atime << "\t";
        cout << a[i].btime << "\t";
        cout << a[i].ctime << "\t";
        cout << a[i].wtime+a[i].ctime-rtime+a[i].btime << "\t";
        averageTAT+=a[i].wtime+a[i].ctime-rtime+a[i].btime;
        cout << a[i].wtime+a[i].ctime-rtime << "\t";
        averageWaitingTime+=a[i].wtime+a[i].ctime-rtime;
        cout << a[i].restime-a[i].atime << "\t";
        averageResponseTime+=a[i].restime-a[i].atime;
        cout <<"\n";
    }
    cout<<"Average Waiting time:
"<<(float)averageWaitingTime/(float)n<<endl;
    cout<<"Average TA time: "<<(float)averageTAT/(float)n<<endl;

}

int main(){
    int nop,choice,i,qt;
    cout<<"Enter number of processes\n";
    cin>>nop;
    cout<<"Enter process, priority, AT, BT\n";
    insert(nop);
    disp(nop,1);
    return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program7 (main) 🍔
➜ ./program7
Enter number of processes
3
Enter process, priority, AT, BT
1 3 5 7
2 4 6 7
3 5 6 8

Gantt Chart
|  Pi   |  P1   |  P2   |  P3
0       5       12      19      27

P.Name Priority AT      BT      CT      TAT     WT      RT
P1      3       5       7       12      2       -5      0
P2      4       6       7       19      8       1       6
P3      5       6       8       27      16      8       13
Average Waiting time: 1.33333
Average TA time: 8.66667
```

PROGRAM 8

CODE →

```
#include<iostream>
#include<algorithm>
using namespace std;

struct node{
    char pname[50];
    int btime;
    int atime;
    int wtime;
    float rr=0;
}a[50];

void insert(int n){
    int i;
    for(i=0;i<n;i++){
        cin>>a[i].pname;
        cin>>a[i].atime;
        cin>>a[i].btime;
        a[i].rr=0;
        a[i].wtime=-a[i].atime;
    }
}

bool btimeSort(node a,node b){
    return a.btime < b.btime;
}
```

```
bool atimeSort(node a,node b){
    return a.atime < b.atime;
}

bool rrtimeSort(node a,node b){
    return a.rr > b.rr;
}

void disp(int n){
    sort(a,a+n,btimeSort);
    sort(a,a+n,atimeSort);
    int ttime=0,i;
    int j,tArray[n];
    for(i=0;i<n;i++){
        j=i;
        while(a[j].atime<=ttime&&j!=n){
            j++;
        }

        for(int q = i;q<j;q++){
            a[q].wtime=ttime-a[q].atime;
            a[q].rr=(float)(a[q].wtime+a[q].btime)/(float)a[q].btime;
        }
        sort(a+i,a+j,rrtimeSort);
        tArray[i]=ttime;
        cout<<endl;
        ttime+=a[i].btime;
    }
    tArray[i] = ttime;

    float averageWaitingTime=0;
```

```
    float averageResponseTime=0;
    float averageTAT=0;
    cout<<"\n";
    cout<<"P.Name  AT\tBT\tCT\tTAT\tWT\tRT\n";
    for (i=0; i<n; i++){
        cout <<'P'<< a[i].pname << "\t";
        cout << a[i].atime << "\t";
        cout << a[i].btime << "\t";
        cout << tArray[i+1] << "\t";
        cout << tArray[i]-a[i].atime+a[i].btime << "\t";
        averageTAT+=tArray[i]-a[i].atime+a[i].btime;
        cout << a[i].wtime << "\t";
        averageWaitingTime+=tArray[i]-a[i].atime;
        cout << tArray[i]-a[i].atime << "\t";
        averageResponseTime+=tArray[i]-a[i].atime;
        cout <<"\n";
    }
    cout<<"\n";
    cout<<"\nGantt Chart\n";
    for (i=0; i<n; i++){
        cout <<"|  P"<< a[i].pname << "   ";
    }
    cout<<"\n";
    for (i=0; i<n+1; i++){
        cout << tArray[i] << "\t";
    }
    cout<<"\n";
    cout<<"Average Response time:
"<<(float)averageResponseTime/(float)n<<endl;
    cout<<"Average Waiting time:
"<<(float)averageWaitingTime/(float)n<<endl;
```

```
    cout<<"Average TA time: "<<(float)averageTAT/(float)n<<endl;
}

int main(){
    int nop,choice,i;
    cout<<"Enter number of processes\n";
    cin>>nop;
    cout<<"Enter process, AT, BT\n";
    insert(nop);
    disp(nop);
    return 0;
}
```

OUTPUT →

```
~/Projects/OS_Lab_Sem4/program8 (main)
→ ./program8
Enter number of processes
3
Enter process, AT, BT
1 4 6
2 5 7
3 4 2

P.Name   AT      BT      CT      TAT     WT      RT
P3       4       2       2       -2      -4      -4
P1       4       6       8       4       -4      -2
P2       5       7       15      10      3       3

Gantt Chart
|  P3   |  P1   |  P2
0       2       8       15
Average Response time: -1
Average Waiting time: -1
Average TA time: 4
```

PROGRAM 9

(a)

CODE →

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

void First_Fit(int block_size[], int total_blocks, int process_size[], int
total_process) {
  int allocation[total_process];
  memset(allocation, -1, sizeof(allocation));
  for (int i = 0; i < total_process; i++) {
    for (int j = 0; j < total_blocks; j++) {
      if (block_size[j] >= process_size[i]) {
        allocation[i] = j;
        block_size[j] -= process_size[i];
        break;
      }
    }
  }
  cout << "\nProcess No.\tProcess Size\tBlock no.\n";
  for (int i = 0; i < total_process; i++) {
    cout << " " << i+1 << "\t\t" << process_size[i] << "\t\t";
    if (allocation[i] != -1)
      cout << allocation[i] + 1;
    else
      cout << "Not Allocated";
      cout << endl;
  }
```

```
}
int main() {
   int total_blocks,total_process;
    cout << "Enter Number of Memory Blocks\n";
    cin >> total_blocks;
    cout << "Enter Number of Process\n";
    cin >> total_process;

   int block_size[total_blocks], process_size[total_process];
   cout << "Enter values of Memory Blocks\n";
   for (int i = 0; i < total_blocks; i++) {
       cin >> block_size[i];
   }
   cout << "Enter values of Process\n";
   for (int i = 0; i < total_process; i++) {
       cin >> process_size[i];
   }
   First_Fit(block_size, total_blocks, process_size, total_process);
   return 0 ;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program9 (main) 🍔
→ ./program9a
Enter Number of Memory Blocks
5
Enter Number of Process
5
Enter values of Memory Blocks
100 534 432 456 245
Enter values of Process
344 567 324 212 100

Process No.     Process Size    Block no.
 1              344             2
 2              567             Not Allocated
 3              324             3
 4              212             4
 5              100             1
```

(b)

CODE →

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

void Next_Fit(int block_size[], int total_blocks, int process_size[], int
total_process)
```

```
{
    int allocation[total_process], j = 0;
    memset(allocation, -1, sizeof(allocation));
    for (int i = 0; i < total_process; i++) {
        while (j < total_blocks) {
            if (block_size[j] >= process_size[i]) {
                allocation[i] = j;
                block_size[j] -= process_size[i];
                break;
            }
            j = (j + 1) % total_blocks;
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
   for (int i = 0; i < total_process; i++) {
      cout << " " << i+1 << "\t\t" << process_size[i] << "\t\t";
      if (allocation[i] != -1)
         cout << allocation[i] + 1;
      else
         cout << "Not Allocated";
         cout << endl;
    }
}
int main() {
   int total_blocks,total_process;
    cout << "Enter Number of Memory Blocks\n";
    cin >> total_blocks;
    cout << "Enter Number of Process\n";
    cin >> total_process;
```

```
    int block_size[total_blocks], process_size[total_process];
    cout << "Enter values of Memory Blocks\n";
    for (int i = 0; i < total_blocks; i++) {
        cin >> block_size[i];
    }
    cout << "Enter values of Process\n";
    for (int i = 0; i < total_process; i++) {
        cin >> process_size[i];
    }
    Next_Fit(block_size, total_blocks, process_size, total_process);
    return 0 ;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program9 (main) 🍔
➜ ./program9b
Enter Number of Memory Blocks
3
Enter Number of Process
3
Enter values of Memory Blocks
245 200 567
Enter values of Process
344 100 234

Process No.     Process Size    Block no.
 1              344             3
 2              100             3
 3              234             1
```

PROGRAM 10

CODE →

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

void Best_Fit(int block_size[], int total_blocks, int process_size[], int
total_process) {
   int allocation[total_process];
   memset(allocation, -1, sizeof(allocation));
   for (int i = 0; i < total_process; i++) {
       int bestIdx = -1;
      for (int j = 0; j < total_blocks; j++) {
         if (block_size[j] >= process_size[i]) {
           if (bestIdx == -1)
               bestIdx = j;
            else if (block_size[bestIdx] > block_size[j])
               bestIdx = j;
         }
      }
      if (bestIdx != -1) {
         allocation[i] = bestIdx;
         block_size[bestIdx] -= process_size[i];
      }
   }
   cout << "\nProcess No.\tProcess Size\tBlock no.\n";
   for (int i = 0; i < total_process; i++) {
      cout << " " << i+1 << "\t\t" << process_size[i] << "\t\t";
      if (allocation[i] != -1)
         cout << allocation[i] + 1;
```

```
      else
        cout << "Not Allocated";
        cout << endl;
   }
}
int main() {
   int total_blocks,total_process;
    cout << "Enter Number of Memory Blocks\n";
    cin >> total_blocks;
    cout << "Enter Number of Process\n";
    cin >> total_process;

   int block_size[total_blocks], process_size[total_process];
   cout << "Enter values of Memory Blocks\n";
   for (int i = 0; i < total_blocks; i++) {
       cin >> block_size[i];
   }
   cout << "Enter values of Process\n";
   for (int i = 0; i < total_process; i++) {
       cin >> process_size[i];
   }
   Best_Fit(block_size, total_blocks, process_size, total_process);
   return 0 ;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program10 (main) 🍔
➜ ./program10
Enter Number of Memory Blocks
4
Enter Number of Process
4
Enter values of Memory Blocks
100 456 543 432
Enter values of Process
134 323 254 346

Process No.     Process Size    Block no.
 1              134             4
 2              323             2
 3              254             4
 4              346             3
```

PROGRAM 11 →

CODE →

```
#include<bits/stdc++.h>
using namespace std;
void Worst_Fit(int block_size[], int total_blocks, int process_size[], int
total_process) {
int allocation[total_process];
memset(allocation, -1, sizeof(allocation));
for (int i = 0; i < total_process; i++) {
int worstIdx = -1;
for (int j = 0; j < total_blocks; j++) {
if (block_size[j] >= process_size[i]) {
if (worstIdx == -1)
worstIdx = j;
else if (block_size[worstIdx] < block_size[j])
worstIdx = j;
}
}
if (worstIdx != -1) {
allocation[i] = worstIdx;
block_size[worstIdx] -= process_size[i];
}
}
cout << "\nProcess No.\tProcess Size\tBlock no.\n";
for (int i = 0; i < total_process; i++) {
cout << " " << i+1 << "\t\t" << process_size[i] << "\t\t";
if (allocation[i] != -1)
cout << allocation[i] + 1;
```

```
else
cout << "Not Allocated";
cout << endl;
}
}
int main() {
int total_blocks,total_process;
cout << "Enter Number of Memory Blocks\n";
cin >> total_blocks;
cout << "Enter Number of Process\n";
cin >> total_process;
int block_size[total_blocks], process_size[total_process];

cout << "Enter values of Memory Blocks\n";
for (int i = 0; i < total_blocks; i++) {
cin >> block_size[i];
}
cout << "Enter values of Process\n";
for (int i = 0; i < total_process; i++) {
cin >> process_size[i];
}
Worst_Fit(block_size, total_blocks, process_size, total_process);
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program11 (main)🍔
➜ ./program11
Enter Number of Memory Blocks
4
Enter Number of Process
4
Enter values of Memory Blocks
100 456 543 432
Enter values of Process
134 323 254 346

Process No.     Process Size    Block no.
 1              134             3
 2              323             2
 3              254             4
 4              346             3
```

PROGRAM 12

CODE →

```
#include<stdio.h>
int main()
{
int i,j,n,frame_no,k,avail,count=0;
printf("\n ENTER THE NUMBER OF PAGES:\n");
scanf("%d",&n);
int page[n];

printf("\n ENTER THE PAGE NUMBER :\n");
for(i=1;i<=n;i++)
scanf("%d",&page[i]);

printf("\n ENTER THE NUMBER OF FRAMES :");
scanf("%d",&frame_no);

int frame[frame_no];
for(i=0;i<frame_no;i++)
   frame[i]= -1;

j=0;

printf("ref string\t page frames\n");

for(i=1;i<=n;i++)
```

```
{
    printf("%d\t\t",page[i]);
    avail=0;
    for(k=0;k<frame_no;k++)
    if(frame[k]==page[i])
        avail=1;
    if (avail==0)
    {
        frame[j]=page[i];
        j=(j+1)%frame_no;
        count++;
        for(k=0;k<frame_no;k++)
        printf("%d\t",frame[k]);
    }
    printf("\n");
}
printf("Page Fault Is %d",count);
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program12 (main) 🍔
➜ ./program12

 ENTER THE NUMBER OF PAGES:
10

 ENTER THE PAGE NUMBER :
1 0 7 4 2 5 8 6 9 1

 ENTER THE NUMBER OF FRAMES :3
ref string       page frames
1               1       -1      -1
0               1       0       -1
7               1       0       7
4               4       0       7
2               4       2       7
5               4       2       5
8               8       2       5
6               8       6       5
9               8       6       9
1               1       6       9
Page Fault Is 10%
```

PROGRAM 13

(a)

CODE →

```
#include<stdio.h>
#include<math.h>
using namespace std;
int main()
{
int n,head,i,j,k,seek=0,max,diff;
float avg;
printf("Enter the max range of disk\n");
scanf("%d",&max);
printf("Enter the size of queue request\n");
scanf("%d",&n);
int queue[n+1];
printf("Enter the queue of disk positions to be read\n");
for(i=1;i<=n;i++) {
    scanf("%d",&queue[i]);
}
printf("Enter the initial head position\n");
scanf("%d",&head);
queue[0]=head;
printf("Disk head moves from \t to \t with seek\n" );
for(j=0;j<=n-1;j++)
{
diff=abs(queue[j+1]-queue[j]);
seek+=diff;
```

```
printf("%d \t\t %d \t %d\n",queue[j],queue[j+1],diff);
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program13 (main) 💥
➜ ./program13
Enter the max range of disk
199
Enter the size of queue request
8
Enter the queue of disk positions to be read
14 34 56 77 86 34 97 150
Enter the initial head position
54
Disk head moves from     to      with seek
54               14      40
14               34      20
34               56      22
56               77      21
77               86      9
86               34      52
34               97      63
97               150     53
Total seek time is 280
Average seek time is 35.000000
```

(b)

CODE →

```
#include<stdio.h>
#include<math.h>
using namespace std;
int main()
{
int n,head,i,j,k,curr,seek=0,max,diff,complete;
float avg;
printf("Enter the max range of disk\n");
scanf("%d",&max);
printf("Enter the size of queue request\n");
scanf("%d",&n);
complete=n;
int queue[n];
printf("Enter the queue of disk positions to be read\n");
for(i=0;i<n;i++)
scanf("%d",&queue[i]);
printf("Enter the initial head position\n");
scanf("%d",&head);
curr=head;
printf("Disk head movmes from \t to \t with seek\n" );
while(complete-->0)
{
int index=-1;
int min = max+1;
for(int j =0;j<n;j++){
if(queue[j]!=-1)
```

```
{
int mn = abs(curr-queue[j]);
if(mn<min)
{
min = mn;
index = j;
}
}
}
diff=abs(curr-queue[index]);seek+=diff;
printf("%d \t\t %d \t %d\n",curr,queue[index],diff);
curr=queue[index];
queue[index]=-1;
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program13 (main) 💥
→ ./program13\(b\)
Enter the max range of disk
199
Enter the size of queue request
8
Enter the queue of disk positions to be read
14 34 56 77 86 34 97 150
Enter the initial head position
54
Disk head movmes from     to       with seek
54                56      2
56                77      21
77                86      9
86                97      11
97                150     53
150               34      116
34                34      0
34                14      20
Total seek time is 232
Average seek time is 29.000000
```

PROGRAM 14

(a)

CODE →

```
#include<stdio.h>
#include<math.h>
#include <bits/stdc++.h>
using namespace std;
int main()
{
int n,head,i,j,k,seek=0,max,diff,curr;
float avg;
string direction;
vector<int> left, right;
printf("Enter the max range of disk\n");
scanf("%d",&max);
printf("Enter the size of queue request\n");
scanf("%d",&n);
int queue[n];
printf("Enter the queue of disk positions to be read\n");for(i=0;i<n;i++)
scanf("%d",&queue[i]);
printf("Enter the direction\n");
cin>>direction;
printf("Enter the initial head position\n");
scanf("%d",&head);
if (direction == "left")
left.push_back(0);
else if (direction == "right")
right.push_back(max - 1);
```

```
for (i = 0; i < n; i++)
{
if (queue[i] <= head)
left.push_back(queue[i]);
if (queue[i] > head)
right.push_back(queue[i]);
}
std::sort(left.begin(), left.end());
std::sort(right.begin(), right.end());
printf("Disk head moves from \t to \t with seek\n" );
int run = 2;
while (run-->0) {
if (direction == "left") {
for ( i = left.size() - 1; i >= 0; i--) {
curr = left[i];
diff = abs(curr - head);
printf("%d \t\t %d \t %d\n",head,curr,diff);seek += diff;
head = curr;
}
direction = "right";
}
else if (direction == "right") {
for ( i = 0; i < right.size(); i++) {
curr = right[i];
diff = abs(curr - head);
printf("%d \t\t %d \t %d\n",head,curr,diff);
seek += diff;
head = curr;
}
direction = "left";
}
```

```
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program14 (main) 💥
➜ ./program14\(a\)
Enter the max range of disk
199
Enter the size of queue request
8
Enter the queue of disk positions to be read
14 34 56 77 86 34 97 150
Enter the direction
right
Enter the initial head position
54
Disk head moves from     to      with seek
54                 56      2
56                 77      21
77                 86      9
86                 97      11
97                 150     53
150                198     48
198                34      164
34                 34      0
34                 14      20
Total seek time is 328
Average seek time is 41.000000
```

(b)

CODE →

```
#include<stdio.h>
#include<math.h>
#include <bits/stdc++.h>
using namespace std;
int main()
{
int n,head,i,j,k,seek=0,max,diff,curr;
float avg;
string direction;
vector<int> left, right;
printf("Enter the max range of disk\n");
scanf("%d",&max);
printf("Enter the size of queue request\n");
scanf("%d",&n);
int queue[n];
printf("Enter the queue of disk positions to be read\n");
for(i=0;i<n;i++)
scanf("%d",&queue[i]);
printf("Enter the direction\n");
cin>>direction;
printf("Enter the initial head position\n");
scanf("%d",&head);
for (i = 0; i < n; i++)
{
if (queue[i] <= head)
left.push_back(queue[i]);
```

```
if (queue[i] > head)
right.push_back(queue[i]);
}
std::sort(left.begin(), left.end());
std::sort(right.begin(), right.end());
printf("Disk head moves from \t to \t with seek\n" );
int run = 2;
while (run-->0) {
if (direction == "left") {
for ( i = left.size() - 1; i >= 0; i--) {curr = left[i];
diff = abs(curr - head);
printf("%d \t\t %d \t %d\n",head,curr,diff);
seek += diff;
head = curr;
}
direction = "right";
}
else if (direction == "right") {
for ( i = 0; i < right.size(); i++) {
curr = right[i];
diff = abs(curr - head);
printf("%d \t\t %d \t %d\n",head,curr,diff);
seek += diff;
head = curr;
}
direction = "left";
}
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
```

```
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program14 (main) 💥
➜ ./program14\(b\)
Enter the max range of disk
199
Enter the size of queue request
8
Enter the queue of disk positions to be read
14 34 56 77 86 34 97 150
Enter the direction
right
Enter the initial head position
54
Disk head moves from     to      with seek
54                 56      2
56                 77      21
77                 86      9
86                 97      11
97                 150     53
150                34      116
34                 34      0
34                 14      20
Total seek time is 232
Average seek time is 29.000000
```

PROGRAM 15

(a)

CODE →

```
#include<stdio.h>
#include<math.h>
#include <bits/stdc++.h>
using namespace std;
int main()
{
int n,head,i,j,k,seek=0,max,diff,curr;
float avg;
string direction;
vector<int> left, right;
printf("Enter the max range of disk\n");
scanf("%d",&max);
printf("Enter the size of queue request\n");
scanf("%d",&n);
int queue[n];
printf("Enter the queue of disk positions to be read\n");
for(i=0;i<n;i++)
scanf("%d",&queue[i]);
printf("Enter the direction\n");
cin>>direction;
printf("Enter the initial head position\n");
scanf("%d",&head);
left.push_back(0);
right.push_back(max - 1);
for (i = 0; i < n; i++)
```

```
{
if (queue[i] <= head)
left.push_back(queue[i]);
if (queue[i] > head)
right.push_back(queue[i]);
}
std::sort(left.begin(), left.end());
std::sort(right.begin(), right.end());
printf("Disk head moves from \t to \t with seek\n" );
int run = 2;
while (run-->0) {if (direction == "left") {
for ( i = left.size() - 1; i >= 0; i--) {
curr = left[i];
diff = abs(curr - head);
printf("
%d \t\t %d \t %d\n",head,curr,diff);
seek += diff;
head = curr;
}
direction = "right";
std::reverse(right.begin(), right.end());
}
else if (direction == "right") {
for ( i = 0; i < right.size(); i++) {
curr = right[i];
diff = abs(curr - head);
printf("
%d \t\t %d \t %d\n",head,curr,diff);
seek += diff;
head = curr;
}
```

```
direction = "left";
std::reverse(left.begin(), left.end());
}
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program15 (main) 💥
➜ ./program15\(a\)
Enter the max range of disk
200
Enter the size of queue request
8
Enter the queue of disk positions to be read
14 34 56 77 86 34 97 150
Enter the direction
left
Enter the initial head position
57
Disk head moves from     to      with seek
57                 56      1
56                 34      22
34                 34      0
34                 14      20
14                 0       14
0                  199     199
199                150     49
150                97      53
97                 86      11
86                 77      9
Total seek time is 378
Average seek time is 47.250000
```

(b)

CODE →

```
#include<stdio.h>
#include<math.h>
#include <bits/stdc++.h>
using namespace std;
int main()
{
int n,head,i,j,k,seek=0,max,diff,curr;
float avg;
string direction;
vector<int> left, right;
printf("Enter the max range of disk\n");
scanf("%d",&max);
printf("Enter the size of queue request\n");
scanf("%d",&n);
int queue[n];
printf("Enter the queue of disk positions to be read\n");
for(i=0;i<n;i++)
scanf("%d",&queue[i]);
printf("Enter the direction\n");
cin>>direction;
printf("Enter the initial head position\n");
scanf("%d",&head);
for (i = 0; i < n; i++)
{
```

```
if (queue[i] <= head)
left.push_back(queue[i]);
if (queue[i] > head)
right.push_back(queue[i]);
}
std::sort(left.begin(), left.end());
std::sort(right.begin(), right.end());
printf("Disk head moves from \t to \t with seek\n" );
int run = 2;
while (run-->0) {
if (direction == "left") {
for ( i = left.size() - 1; i >= 0; i--) {curr = left[i];
diff = abs(curr - head);
printf("
%d \t\t %d \t %d\n",head,curr,diff);
seek += diff;
head = curr;
}
direction = "right";
std::reverse(right.begin(), right.end());
}
else if (direction == "right") {
for ( i = 0; i < right.size(); i++) {
curr = right[i];
diff = abs(curr - head);
printf("
%d \t\t %d \t %d\n",head,curr,diff);
seek += diff;
head = curr;
}
direction = "left";
```

```
std::reverse(left.begin(), left.end());
}
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}
```

OUTPUT →

```
🚀 ~/Projects/OS_Lab_Sem4/program15 (main) 💥
➜ ./program15\(b\)
Enter the max range of disk
200
Enter the size of queue request
8
Enter the queue of disk positions to be read
14 34 56 77 86 34 97 150
Enter the direction
left
Enter the initial head position
57
Disk head moves from     to      with seek
57                 56      1
56                 34      22
34                 34      0
34                 14      20
14                 150     136
150                97      53
97                 86      11
86                 77      9
Total seek time is 252
Average seek time is 31.500000
```