OS Lab File

Submitted By: Tanmay Vig

Roll No.: 19BCS061

Class: 2nd year 4th semester

Table of Content:

S No.	Program	Page No.
1	Implement the priority queue schedulling algorithm using linked list.	8
2	Write a program to implement the First Come First Serve scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.	13
3	Write a program to implement the shortest job first non-preemptive scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.	18
4	Write a program to implement the Shortest Remaining Time First (Shortest job first preemptive) scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.	25
5	Write a program to implement the round robin scheduling algorithm having variables time quantum and	31

	find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.	
6	Write a program to implement the Non-preemptive priority scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.	40
7	Write a program to implement the preemptive priority scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.	47
8	Write a program to implement the Highest Response Ratio Next (Non-preemptive) algorithm and find the average turnaround time, waiting time, completion time and response time for overall process.	56
9	(a) Write a program to implement the First fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the	64

	details about memory allocated to process with fragmentation detail. (b) Write a program to implement the Next fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.	
10	Write a program to implement the Best fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.	71
11	Write a program to implement the worst fit memory management algorithm. The program should take input total no. of the memory block, their sizes, process name, and process size. The output of the program should give the details about memory allocated to process with fragmentation detail.	75
12	Write a program to implement the First In First Out(FIFO) page	79

	replacement algorithm. Program should takes input reference string and total no. of pages that can accommodate in memory. Output contains detail about each page fault details and calculate average page fault.	
13	(a) Write a program to implement the FCFS elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement. (b) Write a program to implement the SSTF elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.	83
14	(a) Write a program to implement the SCAN elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.	87

	(b) Write a program to implement the LOOK elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.	
15	(a) Write a program to implement the C-SCAN elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement. (b) Write a program to implement the C-LOOK elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.	98

Problem 1: Implement the priority queue scheduling algorithm using linked list.

Answer:

Source code:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct Node
{
    char process[3];
    int priority;
    struct Node* next;
};

struct Node* Head= NULL;

// print all elements of queue
void print(){
    printf("printing queue...\t");
    struct Node *temp = Head;
    if(temp->next==NULL){
        printf("|%s|%d|",temp->process,temp->priority);
    }else{
        while(temp != NULL){
            printf("|%s|%d|",temp->process,temp->priority);
            if(temp->next != NULL){
                printf("-->");
            }
            temp=temp->next;
        }
```

```
    }
    printf("\n");
}

// delete all nodes before exiting
void ext(){
    struct Node *trav = Head;
    struct Node *temp = Head;
    while(trav!=NULL){
        temp=trav;
        trav=trav->next;
        free(temp);
    }
    printf("Traversed\n");
}

// add element to queue
void enqeue(int priority, char* process){
    printf("Adding...\n");
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->priority=priority;
    strcpy(temp->process,process);
    temp->next=NULL;

    if(Head==NULL){
        Head=temp;
    }else{
        if(priority < Head->priority){
            temp->next=Head;
            Head = temp;
        }else{
            // traverse till dont find the correct priority
            // then insert node at the best position
            struct Node* trav= Head;
```

```
            while(trav->next!=NULL){
                if(trav->next->priority > priority){
                    break;
                }
                trav=trav->next;
            }
            temp->next=trav->next;
            trav->next=temp;
        }
    }
    print();
}

// executing process
void execute(){
    struct Node *temp = Head;
    if(temp==NULL){
        printf("No processes left.\n");
    }else if(temp->next==NULL){
        free(temp);
        printf("All processes finished.\n");
    }else{
        Head=temp->next;
        printf("%s process having priority %d executed.\n",temp->process,temp->priority);
        free(temp);
        print();
    }
}

void main()
{
    int choice = 0, br=0;
    while (choice!=3 && br!=1)
    {
```

```
        choice=0;
        printf("To ADD PROCESS press 1 \nTo EXECUTE PROCESS press 2 \nTo EXIT press 3\n");
        scanf("%d",&choice);
        switch (choice)
        {
        case 1:;
            // add node according to priority
            char *n;
            n=(char*)malloc(sizeof(char)*3);
            int i=0;
            printf("Enter process name\n");
            scanf("%s",n);
            printf("Enter PRIORITY\n");
            scanf("%d",&i);
            enqeue(i,n);
            break;
        case 2:
            // execute process
            execute();
            break;
        case 3:
            // Exit the process and clear space
            ext();
            printf("EXITING...\n");
            break;
        default:
            //print error and breaking loop
            printf("ILLEGAL INPUT\n");
            br=1;
            break;
        }
    }
}
```

Screen shots of code output:

```
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
1
Enter process name
p1
Enter PRIORITY
4
Adding...
printing queue...       |p1|4|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
1
Enter process name
p2
Enter PRIORITY
1
Adding...
printing queue...       |p2|1|-->|p1|4|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
1
Enter process name
p3
Enter PRIORITY
2
Adding...
printing queue...       |p2|1|-->|p3|2|-->|p1|4|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
1
Enter process name
p4
Enter PRIORITY
6
Adding...
printing queue...       |p2|1|-->|p3|2|-->|p1|4|-->|p4|6|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
2
p2 process having priority 1 executed.
printing queue...       |p3|2|-->|p1|4|-->|p4|6|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
2
```

```
p3 process having priority 2 executed.
printing queue...       |p1|4|-->|p4|6|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
1
Enter process name
p10
Enter PRIORITY
3
Adding...
printing queue...       |p10|3|-->|p1|4|-->|p4|6|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
2
p10 process having priority 3 executed.
printing queue...       |p1|4|-->|p4|6|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
2
p1 process having priority 4 executed.
printing queue...       |p4|6|
To ADD PROCESS press 1
To EXECUTE PROCESS press 2
To EXIT press 3
3
Traversed
EXITING...

--------------------------------
Process exited after 162.4 seconds with return value 3
Press any key to continue . . .
```

Problem 2: Write a program to implement the First Come First Serve scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

Solution:

Source code:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

typedef struct
{
   char process_name[3];
   int arrival_time;
   int burst_time;
   int complete_time;
   int turn_around_time;
   int wait_time;
   int response_time;
} process;

void print_process_table(process arr[],int n){
   int i;
   puts(" _____ _____ _____ _____ _____ _
_____ _____");
```

```
   puts("| Process Name | Arrival Time  | Burst Time | Complete Time | Turn Around Time | Wait Time | Response Time |");
   for(i=0; i<n;i++){
      puts("|_____|_____|_____|_____|_____|_____|_____|");
      printf("|    %3s    |    %3d    |    %3d   |    %3d    |    %4d    |   %3d   |    %3d    |\n",
      arr[i].process_name,arr[i].arrival_time,arr[i].burst_time,arr[i].complete_time,arr[i].turn_around_time,arr[i].wait_time,arr[i].response_time);
   }
   puts("|_____|_____|_____|_____|_____|_____|_____|");

}

void get_average(process arr[], int n){
   double tat=0,wt=0,rt=0;
   int i;
   for(i=0;i<n;i++){
      tat += (double)arr[i].turn_around_time;
      wt += (double)arr[i].wait_time;
      rt += (double)arr[i].response_time;
   }
   printf("Total time to Complete = %3d       Average Time to Complete = %.3f\n",arr[n-1].complete_time,(double)arr[n-1].complete_time/(double)n);
   printf("Total Turn Around Time = %.3f     Average Turn Around Time = %.3f\n",tat,tat/(double)n);
   printf("Total Waiting Time = %.3f        Average Waiting Time = %.3f\n",wt,wt/(double)n);
   printf("Total Response Time = %.3f       Average Response Time = %.3f\n",rt,rt/(double)n);
}

void gnatt(process arr[],int n){
```

```
int i,j;
// upper row
printf(" ");
for(i=0; i<n;i++){
   for(j=0;j<arr[i].burst_time+1;j++) printf("__");
   printf(" ");
}
printf("\n|");
// middle row
for(i=0;i<n;i++){
   for(j=0;j<arr[i].burst_time-1;j++){
      printf(" ");
   }
   printf("%3s",arr[i].process_name);
   for(j=0;j<arr[i].burst_time;j++){
      printf(" ");
   }
   printf("|");
}
printf("\n|");
// lower row
for(i=0; i<n;i++){
   for(j=0;j<arr[i].burst_time+1;j++) printf("__");
   printf("|");
}
printf("\n");
printf("0");
for(i=0; i<n; i++) {
   for(j=0; j<arr[i].burst_time+1; j++) printf("  ");
```

```
        if(arr[i].turn_around_time > 9) printf("\b");
        printf("%d", arr[i].turn_around_time);

    }
    printf("\n");
}

void main()
{
    int n =0,i;
    printf("Enter the number of processes\t");
    scanf("%d",&n);
    process arr[n];
    printf("Enter PROCESS_NAME ARRIVAL_TIME and BURST_TIME\n");
    for(i=0; i<n;i++)
    {
        scanf("%s %d %d",arr[i].process_name,&arr[i].arrival_time,&arr[i].burst_time);
    }
    // calculating completion time
    arr[0].complete_time=arr[0].burst_time + arr[0].arrival_time;
    arr[0].turn_around_time = arr[0].complete_time -  arr[0].arrival_time;
    arr[0].wait_time = arr[0].turn_around_time - arr[0].burst_time;
    arr[0].response_time = arr[0].wait_time;
    for(i=1; i<n;i++)
    {
        arr[i].complete_time = arr[i-1].complete_time + arr[i].burst_time;
        arr[i].turn_around_time = arr[i].complete_time-arr[i].arrival_time;
        arr[i].wait_time = arr[i].response_time =arr[i].turn_around_time-arr[i].burst_time;
    }
```

```
    print_process_table(arr,n);
    get_average(arr, n);
    puts("---------------------- GNATT CHART --------------------");
    gnatt(arr,n);
}
```

Output:

```
Enter the number of processes   5
Enter PROCESS_NAME ARRIVAL_TIME and BURST_TIME
p01 0 5
p02 4 7
p03 6 2
p04 8 4
p05 10 10
 _____________ _____________ _____________ _____________ _________________ _____________ _____________
| Process Name | Arrival Time | Burst Time | Complete Time | Turn Around Time | Wait Time | Response Time |
|_____________|_____________|_____________|_____________|_________________|_____________|_____________|
|      p01    |       0     |       5     |       5     |         5       |      0      |       0     |
|_____________|_____________|_____________|_____________|_________________|_____________|_____________|
|      p02    |       4     |       7     |      12     |         8       |      1      |       1     |
|_____________|_____________|_____________|_____________|_________________|_____________|_____________|
|      p03    |       6     |       2     |      14     |         8       |      6      |       6     |
|_____________|_____________|_____________|_____________|_________________|_____________|_____________|
|      p04    |       8     |       4     |      18     |        10       |      6      |       6     |
|_____________|_____________|_____________|_____________|_________________|_____________|_____________|
|      p05    |      10     |      10     |      28     |        18       |      8      |       8     |
|_____________|_____________|_____________|_____________|_________________|_____________|_____________|
Total time to Complete =  28        Average Time to Complete = 5.600
Total Turn Around Time = 49.000     Average Turn Around Time = 9.800
Total Waiting Time = 21.000         Average Waiting Time = 4.200
Total Response Time = 21.000        Average Response Time = 4.200
---------------------- GNATT CHART -------------------
 _____ _____________ _____ _____ _____________________
|  p01  |     p02     | p03 | p04 |         p05         |
|_____|_____________|_____|_____|_____________________|
0       5             8     8       10                  18

--------------------------------
Process exited after 40.79 seconds with return value 10
Press any key to continue . . .
```

Problem 3: Write a program to implement the shortest job first non-preemptive scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

Solution:

Source Code:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>


typedef struct

{

   char process_name[3];

   int arrival_time;

   int burst_time;

   int complete_time;

   int turn_around_time;

   int wait_time;

   int response_time;

} process;
```

```
void print_process_table(process arr[],int n){

   int i;

   puts(" _________________ _________________ _____________ _________________ __
_____________________ _____________ _____________________");

   puts("| Process Name | Arrival Time  | Burst Time | Complete Time | Turn Aroun
d Time | Wait Time | Response Time |");

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

      puts("|_________________|_________________|_____________|_________________
_|_____________________|_____________|_____________________|");

      printf("|     %3s     |     %3d     |     %3d     |     %3d     |      %4d      |    %3d    |
   %3d      |\n",

      arr[i].process_name,arr[i].arrival_time,arr[i].burst_time,arr[i].complete_time,ar
r[i].turn_around_time,arr[i].wait_time,arr[i].response_time);

   }

   puts("|_________________|_________________|_____________|_________________|
_____________________|_____________|_____________________|");



}



void get_average(process arr[], int n){

   double tat=0,wt=0,rt=0;

   int i;

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

      tat += (double)arr[i].turn_around_time;
```

```
        wt += (double)arr[i].wait_time;

        rt += (double)arr[i].response_time;

    }

    printf("Total time to Complete = %3d        Average Time to Complete = %.3f\n",arr
[n-1].complete_time,(double)arr[n-1].complete_time/(double)n);

    printf("Total Turn Around Time = %.3f     Average Turn Around Time = %.3f\n",tat,
tat/(double)n);

    printf("Total Waiting Time = %.3f         Average Waiting Time = %.3f\n",wt,wt/(dou
ble)n);

    printf("Total Response Time = %.3f        Average Response Time = %.3f\n",rt,rt/(do
uble)n);

}



void gantt(process arr[],int n){

    int i,j;

    // upper row

    printf(" ");

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

        for(j=0;j<arr[i].burst_time+1;j++) printf("__");

        printf(" ");

    }

    printf("\n|");

    // middle row
```

```
for(i=0;i<n;i++){
    for(j=0;j<arr[i].burst_time-1;j++){
        printf(" ");
    }
    printf("%3s",arr[i].process_name);
    for(j=0;j<arr[i].burst_time;j++){
        printf(" ");
    }
    printf("|");
}
printf("\n|");
// lower row
for(i=0; i<n;i++){
    for(j=0;j<arr[i].burst_time+1;j++) printf("__");
    printf("|");
}
printf("\n");
printf("0");
for(i=0; i<n; i++) {
    for(j=0; j<arr[i].burst_time+1; j++) printf("  ");
    if(arr[i].turn_around_time > 9) printf("\b");
```

```
        printf("%d", arr[i].complete_time);


    }
    printf("\n");
}

void swap(process arr[],int ind1, int ind2){
    process temp = arr[ind1];
    arr[ind1] = arr[ind2];
    arr[ind2] = temp;


}

void main()
{
    int n =0,i,ct=0, mt=0,j, temp;
    printf("Enter the number of processes\t");
    scanf("%d",&n);
    process arr[n];
    printf("Enter PROCESS_NAME ARRIVAL_TIME and BURST_TIME\n");
    for(i=0; i<n;i++)
```

```
{
    scanf("%s %d %d",arr[i].process_name,&arr[i].arrival_time,&arr[i].burst_time);
}

 // calculating completion time
for(j=0;j<n;j++){
    mt=arr[j].burst_time;
    for(i=j+1;i<n;i++){
        if(arr[i].arrival_time<=ct && arr[i].burst_time<mt){
            swap(arr,j,i);
        }
        if(ct<arr[i].arrival_time){
            break;
        }
    }
    if(j==0){
        temp=0;
    }else{
        temp = arr[j-1].complete_time;
    }
    arr[j].complete_time=arr[j].burst_time+temp;
```

```
        arr[j].turn_around_time = arr[j].complete_time -  arr[j].arrival_time;
        arr[j].wait_time = arr[j].turn_around_time - arr[j].burst_time;
        arr[j].response_time = arr[j].wait_time;
        ct=arr[j].complete_time;
    }
    print_process_table(arr,n);
    get_average(arr, n);
    puts("---------------------- GANTT CHART -------------------");
    gantt(arr,n);
}
```

Output:

```
Enter the number of processes   4
Enter PROCESS_NAME ARRIVAL_TIME and BURST_TIME
p01 0 6
p02 2 9
p03 3 3
p04 4 2
 _____________ _______________ _____________ _______________ _________________ _____________ _________________
| Process Name | Arrival Time  | Burst Time  | Complete Time | Turn Around Time | Wait Time   | Response Time |
|_____________|_______________|_____________|_______________|_________________|_____________|_________________|
|      p01    |        0      |       6     |        6      |         6       |      0      |        0      |
|_____________|_______________|_____________|_______________|_________________|_____________|_________________|
|      p04    |        4      |       2     |        8      |         4       |      2      |        2      |
|_____________|_______________|_____________|_______________|_________________|_____________|_________________|
|      p03    |        3      |       3     |       11      |         8       |      5      |        5      |
|_____________|_______________|_____________|_______________|_________________|_____________|_________________|
|      p02    |        2      |       9     |       20      |        18       |      9      |        9      |
|_____________|_______________|_____________|_______________|_________________|_____________|_________________|
Total time to Complete =  20        Average Time to Complete = 5.000
Total Turn Around Time = 36.000     Average Turn Around Time = 9.000
Total Waiting Time = 16.000         Average Waiting Time = 4.000
Total Response Time = 16.000        Average Response Time = 4.000
---------------------- GANTT CHART -------------------
 _____________ _____ _______ ____________________
|     p01     | p04 |  p03  |        p02         |
|_____________|_____|_______|____________________|
0             6     8       11                   20
```

Program 4: Write a program to implement the Shortest Remaining Time First (Shortest job first preemptive) scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

Solution:

Source Code:

```
#include<stdio.h>
#include<string.h>
void sort(int arr[][6],char str[][10], int at, int bt, char p[], int n, int m);
void print(int n, char str[][10], int arr[][6] );
void ganttChart(int time[],char gantt[][10], int m, int l);

int main(){
   char process[10], gantt[100][10];
   int time[100];
   int arrival_time,burst_time,n;
   printf("Enter no of process :");
   scanf("%d",&n);

   int arr[n][6];
   int temp[n+1];
   char str[n][10];
```

```
printf("Enter 'process arrival_time burst_time' :\n");
scanf("%s",str[0]);
scanf("%d",&arr[0][0]);
scanf("%d",&arr[0][1]);
for (int i=1; i<n; i++){
   scanf("%s",process);
   scanf("%d",&arrival_time);
   scanf("%d",&burst_time);
   int j=0;
   while (j<i && arr[j][0]<=arrival_time){
      j++;
   }
   sort(arr,str,arrival_time,burst_time,process,i,j);
}

for (int i=0; i<n; i++){
   arr[i][5]=-1;
   temp[i]=arr[i][1];
}

temp[n]=1000;
time[0]=arr[0][0];
int l=1, m=0, count=0, prev=0, last;
for (int t=0; count<n; t++){
   int min=n;
   for (int i=0; i<n; i++){
      if (arr[i][0]<=t && temp[i]<temp[min] && temp[i]>0){
         min=i;
      }
```

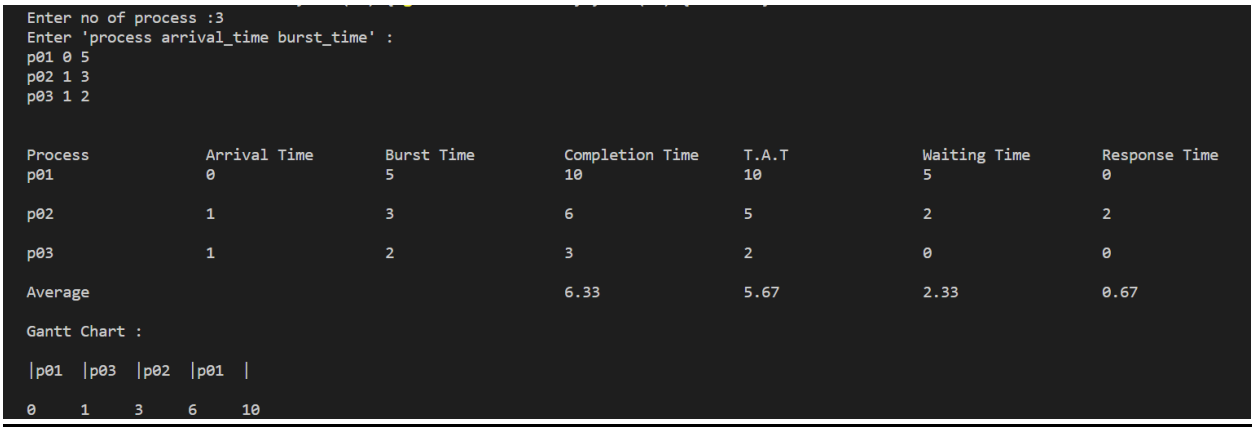
```
    }

    if (arr[min][5]==-1){
        arr[min][5]=t-arr[min][0];
    }

    if (prev!=min){
        strcpy(gantt[m],str[prev]);
        time[l]=t;
        l++;
        m++;
        prev=min;
    }

    temp[min]--;
    if (temp[min]==0){
        count++;
        arr[min][2]=t+1;
        last=min;
    }
}

strcpy(gantt[m],str[last]);
time[l]=arr[last][2];
l++;
m++;

for (int i=0; i<n; i++){
    arr[i][3]=arr[i][2]-arr[i][0];
```

```
        arr[i][4]=arr[i][3]-arr[i][1];
    }
    print(n,str,arr);
    ganttChart(time,gantt,m,l);
    return 0;
}

void sort(int arr[][6],char str[][10], int at, int bt, 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];
        strcpy(str[i+1],str[i]);
    }
    arr[m][0]=at;
    arr[m][1]=bt;
    strcpy(str[m],p);
}

void print(int n, char str[][10], int arr[][6] ){
    float avg;
    float sum;
    char title[7][20]={"Process","Arrival Time","Burst Time","Completion Time","T.A.T",
                       "Waiting Time","Response Time"};

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

```
    for (int i=0; i<n; i++){
        printf("%-20s",str[i]);
        for (int j=0; j<6; j++){
            printf("%-20d",arr[i][j]);
        }
        printf("\n\n");
    }
    printf("%-60s","Average");
    for (int j=2; j<6; 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:



```
Enter no of process :3
Enter 'process arrival_time burst_time' :
p01 0 5
p02 1 3
p03 1 2


Process             Arrival Time        Burst Time          Completion Time     T.A.T               Waiting Time        Response Time
p01                 0                   5                   10                  10                  5                   0

p02                 1                   3                   6                   5                   2                   2

p03                 1                   2                   3                   2                   0                   0

Average                                                     6.33                5.67                2.33                0.67

Gantt Chart :

|p01  |p03  |p02  |p01  |

0     1     3     6     10
```

Problem 5: Write a program to implement the round robin scheduling algorithm having variables time quantum and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

Solution:

Source Code:

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>


typedef struct

{

   char process_name[3];

   int arrival_time;

   int burst_time;

   int complete_time;

   int turn_around_time;

   int wait_time;
```

```
    int response_time;
    int time_left;
} process;

int have_task(process arr[], int n){
    // printf("HERE");
    int have = 0,i;
    for(i=0; i<n;i++){
        if(arr[i].time_left > 0){
            have=1;
            break;
        }
    }
    return have;
}

void print_process_table(process arr[],int n){
    int i;
    puts(" _____ _____ _____ _____ _____ _____ _____ _____");
```

```
    puts("| Process Name | Arrival Time  | Burst Time | Complete Time | T
urn Around Time | Wait Time | Response Time |");

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

        puts("|_____|_____|_____|_____
_____|_____|_____|_____|");

        printf("|     %3s     |     %3d     |     %3d     |     %3d     |     %4d
  |    %3d    |     %3d     |\n",

        arr[i].process_name,arr[i].arrival_time,arr[i].burst_time,arr[i].comp
lete_time,arr[i].turn_around_time,arr[i].wait_time,arr[i].response_time);

    }

    puts("|_____|_____|_____|_____
_____|_____|_____|_____|");



}


void get_average(process arr[], int n){

    double tat=0,wt=0,rt=0;

    int i;

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

        tat += (double)arr[i].turn_around_time;

        wt += (double)arr[i].wait_time;

        rt += (double)arr[i].response_time;
```

```
    }

    printf("Total time to Complete = %3d        Average Time to Complete
 = %.3f\n",arr[n-1].complete_time,(double)arr[n-
1].complete_time/(double)n);

    printf("Total Turn Around Time = %.3f      Average Turn Around Tim
e = %.3f\n",tat,tat/(double)n);

    printf("Total Waiting Time = %.3f          Average Waiting Time = %.3
f\n",wt,wt/(double)n);

    printf("Total Response Time = %.3f         Average Response Time = %
.3f\n",rt,rt/(double)n);

}



void gnatt(process arr[],int n,int time_quantum){

    int i,j,time=0,total_time=0;

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

        arr[i].time_left = arr[i].burst_time;

    }

    printf("\n|");

    i=0;

    while(have_task(arr,n)==1){

        if(arr[i].time_left>0){

            printf("%3s  ",arr[i].process_name);
```

```
            printf("|");
        }
        if(time_quantum<arr[i].time_left){
            time = time_quantum ? time_quantum<arr[i].time_left : arr[i].tim
e_left;
        }else{
            time = arr[i].time_left;
        }
        arr[i].time_left-=time;
        i++;
        i%=n;
    }
    printf("\n");
    for(i=0; i<n;i++){
        arr[i].time_left = arr[i].burst_time;
    }
    i=0;
    while(have_task(arr,n)==1){
        if(arr[i].time_left>0){
            printf("%2d",total_time);
```

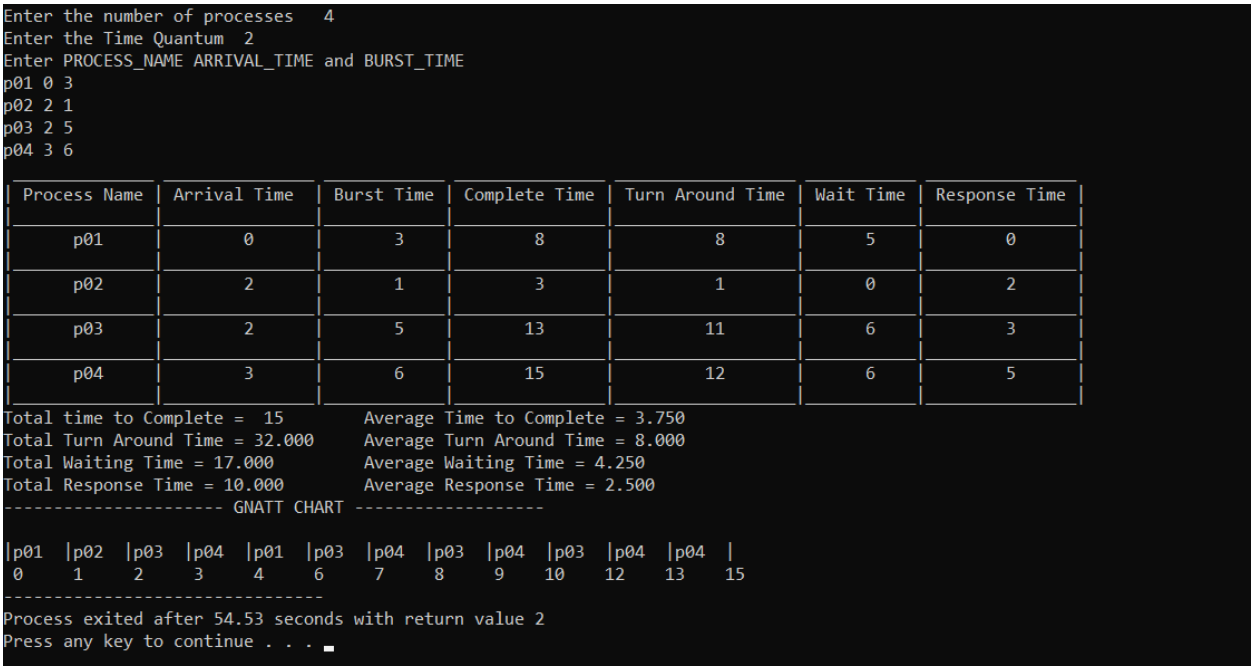
```
            printf("    ",arr[i].process_name);


        }
        if(time_quantum<arr[i].time_left){
            time = time_quantum ? time_quantum<arr[i].time_left : arr[i].tim
e_left;
        }else{
            time = arr[i].time_left;
        }
        arr[i].time_left-=time;
        total_time+=time;
        i++;
        i%=n;
    }
    printf("%d",total_time);


}


void main()
{
```

```
    int n =0,i, total_time=0, time_quantum;
    printf("Enter the number of processes\t");
    scanf("%d",&n);
    printf("Enter the Time Quantum\t");
    scanf("%d",&time_quantum);
    process arr[n];
    printf("Enter PROCESS_NAME ARRIVAL_TIME and BURST_TI
ME\n");
    for(i=0; i<n;i++)
    {
        scanf("%s %d %d",arr[i].process_name,&arr[i].arrival_time,&arr[i]
.burst_time);
        arr[i].time_left=arr[i].burst_time;
        // response time
        arr[i].response_time = total_time;
        if(arr[i].burst_time<time_quantum){
            total_time += arr[i].burst_time;
        }else{
            total_time += time_quantum;
        }
    }
```

```
    // complete time
    total_time=0;
    i=0;
    while(have_task(arr,n)==1){
        // printf("HERE IN WHILE");
        if(arr[i].time_left > 0){
            if(time_quantum<arr[i].time_left){
                total_time+=time_quantum;
                arr[i].time_left -= time_quantum;
            }else{
                total_time+=arr[i].time_left;
                arr[i].time_left -= arr[i].time_left;
            }
            if(arr[i].time_left==0){
                arr[i].complete_time = total_time;
                arr[i].turn_around_time = arr[i].complete_time-
arr[i].arrival_time;
                arr[i].wait_time = arr[i].turn_around_time-arr[i].burst_time;
            }
        }
```

```
        i++;
        i%=n;
    }
    print_process_table(arr,n);
    get_average(arr, n);
    printf("---------------------- GNATT CHART -------------------\n");
    gnatt(arr,n,time_quantum);
}
```

Output:



```
Enter the number of processes   4
Enter the Time Quantum  2
Enter PROCESS_NAME ARRIVAL_TIME and BURST_TIME
p01 0 3
p02 2 1
p03 2 5
p04 3 6
```

Process Name	Arrival Time	Burst Time	Complete Time	Turn Around Time	Wait Time	Response Time
p01	0	3	8	8	5	0
p02	2	1	3	1	0	2
p03	2	5	13	11	6	3
p04	3	6	15	12	6	5

```
Total time to Complete =  15        Average Time to Complete = 3.750
Total Turn Around Time = 32.000     Average Turn Around Time = 8.000
Total Waiting Time = 17.000         Average Waiting Time = 4.250
Total Response Time = 10.000        Average Response Time = 2.500
---------------------- GNATT CHART -------------------

|p01  |p02  |p03  |p04  |p01  |p03  |p04  |p03  |p04  |p03  |p04  |p04  |
 0     1     2     3     4     6     7     8     9     10    12    13    15
--------------------------------
Process exited after 54.53 seconds with return value 2
Press any key to continue . . .
```

Problem 6: Write a program to implement the Non-preemptive priority scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

Solution:

Source Code:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<limits.h>

typedef struct
{
   char process_name[3];
   int arrival_time;
   int burst_time;
   int priority;
   int complete_time;
   int turn_around_time;
   int wait_time;
   int response_time;
   int done;
```

```
} process;


void print_process_table(process arr[],int n){
    int i;
    puts(" _____ _____ _____ _____ _
_____ _____ _____");
    puts("| Process Name | Arrival Time  | Burst Time | Complete Time | Turn Aroun
d Time | Wait Time | Response Time |");
    for(i=0; i<n;i++){
        puts("|_____|_____|_____|_____|
_____|_____|_____|");
        printf("|  %3s  |  %3d  |  %3d  |  %3d  |  %4d  |  %3d
|  %3d  |\n",
        arr[i].process_name,arr[i].arrival_time,arr[i].burst_time,arr[i].complete_time,
arr[i].turn_around_time,arr[i].wait_time,arr[i].response_time);
    }
    puts("|_____|_____|_____|_____|__
_____|_____|_____|");


}


void get_average(process arr[], int n){
    double tat=0,wt=0,rt=0;
    int i;
    for(i=0;i<n;i++){
        tat += (double)arr[i].turn_around_time;
        wt += (double)arr[i].wait_time;
```

```
        rt += (double)arr[i].response_time;
    }
    printf("Total time to Complete = %3d       Average Time to Complete = %.3f\n",
arr[n-1].complete_time,(double)arr[n-1].complete_time/(double)n);
    printf("Total Turn Around Time = %.3f     Average Turn Around Time = %.3f\n
",tat,tat/(double)n);
    printf("Total Waiting Time = %.3f         Average Waiting Time = %.3f\n",wt,wt/
(double)n);
    printf("Total Response Time = %.3f        Average Response Time = %.3f\n",rt,rt
/(double)n);
}

void gnatt(process arr[],int n){
    int i,j;
    // upper row
    printf(" ");
    for(i=0; i<n;i++){
        for(j=0;j<arr[i].burst_time+1;j++) printf("__");
        printf(" ");
    }
    printf("\n|");
    // middle row
    for(i=0;i<n;i++){
        for(j=0;j<arr[i].burst_time-1;j++){
            printf(" ");
        }
        printf("%3s",arr[i].process_name);
```

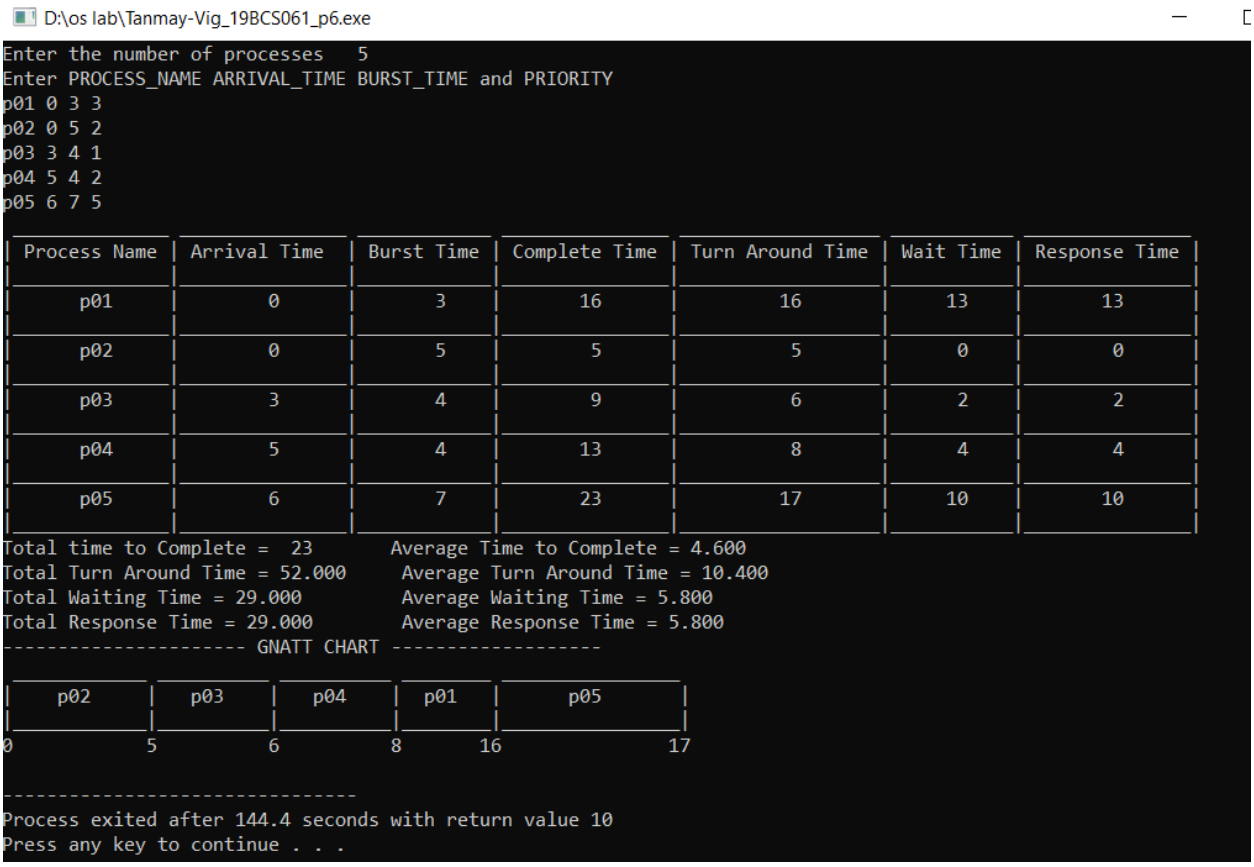
```
        for(j=0;j<arr[i].burst_time;j++){
            printf(" ");
        }
        printf("|");
    }
    printf("\n|");
    // lower row
    for(i=0; i<n;i++){
        for(j=0;j<arr[i].burst_time+1;j++) printf("__");
        printf("|");
    }
    printf("\n");
    printf("0");
    for(i=0; i<n; i++) {
        for(j=0; j<arr[i].burst_time+1; j++) printf("  ");
        if(arr[i].turn_around_time > 9) printf("\b");
        printf("%d", arr[i].turn_around_time);

    }
    printf("\n");
}

int completed(process arr[], int n){
    int i=0,flag=1;
    for(i=0;i<n;i++){
```

```
        if(arr[i].done==0){
            flag=0;
            break;
        }
    }
    return flag;
}

int best_process(process arr[], int n, int time){
    int ind=-1,i=0,priority=INT_MAX;
    for(i=0;i<n;i++){
        if(arr[i].arrival_time > time){
            break;
        }else{
            if(arr[i].done==0 && arr[i].priority<priority){
                priority=arr[i].priority;
                ind=i;
            }
        }
    }
    return ind;
}

void main()
{
```

```
    int n =0,i, total_time=0,temp=0;
    printf("Enter the number of processes\t");
    scanf("%d",&n);
    process arr[n], gnt[n];
    printf("Enter PROCESS_NAME ARRIVAL_TIME BURST_TIME and PRIOR
ITY\n");
    for(i=0; i<n;i++)
    {
        scanf("%s %d %d %d",arr[i].process_name,&arr[i].arrival_time,&arr[i].burst
_time,&arr[i].priority);
        arr[i].done=0;
    }


    i=0;
    while (completed(arr,n)!=1)
    {
        temp=best_process(arr,n, total_time); //return index of that process to execute.
        if(temp==-1){
            total_time++;
        }else{
            arr[temp].complete_time = total_time+arr[temp].burst_time;
            arr[temp].turn_around_time = arr[temp].complete_time-
arr[temp].arrival_time;
            arr[temp].response_time = total_time-arr[temp].arrival_time;
            arr[temp].wait_time = arr[temp].turn_around_time-arr[temp].burst_time;
            total_time += arr[temp].burst_time;
```

```
            arr[temp].done=1;
            gnt[i++]=arr[temp];
        }
    }
    print_process_table(arr,n);
    get_average(arr, n);
    puts("---------------------- GNATT CHART -------------------");
    gnatt(gnt,n);
}
```

Output:



Problem 7: Write a program to implement the preemptive priority scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

Solution:

Source code:

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


struct node{
    char process_name;
    int burst_time;
    int arrival_time;
    int response_time;
    int priority;
    int wait_time;
    int complete_time;
}arr[1000],brr[1000],crr[1000];
```

```
void insert(int n){
    int i;
    for(i=0;i<n;i++){
        cin>>arr[i].process_name;
        cin>>arr[i].priority;
        cin>>arr[i].arrival_time;
        cin>>arr[i].burst_time;
        arr[i].wait_time=-arr[i].arrival_time+1;
    }
}

bool arrival_time_sort(node arr,node brr){
    return arr.arrival_time < brr.arrival_time;
}
bool prioritySort(node arr,node brr){
    return arr.priority < brr.priority;
}
int k=0,f=0,r=0;
void disp(int nop,int qt){
    int n=nop,q;
    sort(arr,arr+n,arrival_time_sort);
```

```
int ttime=0,i;
int j,tArray[n];
int alltime=0;
bool moveLast=false;
for(i=0;i<n;i++){
    alltime+=arr[i].burst_time;
}
alltime+=arr[0].arrival_time;
for(i=0;ttime<=alltime;){
    j=i;
    while(arr[j].arrival_time<=ttime&&j!=n){
        brr[r]=arr[j];
        j++;
        r++;
    }
    if(r==f){
        crr[k].process_name='i';
        crr[k].burst_time=arr[j].arrival_time-ttime;
        crr[k].arrival_time=ttime;
        ttime+=crr[k].burst_time;
        k++;
```

```
    continue;
}
i=j;
if(moveLast==true){
    sort(brr+f,brr+r,prioritySort);
}

j=f;
if(brr[j].burst_time>qt){
    crr[k]=brr[j];
    crr[k].burst_time=qt;
    k++;
    brr[j].burst_time=brr[j].burst_time-qt;
    ttime+=qt;
    moveLast=true;
    for(q=0;q<n;q++){
        if(brr[j].process_name!=arr[q].process_name){
            arr[q].wait_time+=qt;
        }
    }
}
```

```
    else{
        crr[k]=brr[j];
        k++;
        f++;
        ttime+=brr[j].burst_time;
        moveLast=false;
        for(q=0;q<n;q++){
            if(brr[j].process_name!=arr[q].process_name){
                arr[q].wait_time+=brr[j].burst_time;
            }
        }
    }
    if(f==r&&i>=n)
    break;
}
tArray[i]=ttime;
ttime+=arr[i].burst_time;
for(i=0;i<k-1;i++){
    if(crr[i].process_name==crr[i+1].process_name){
        crr[i].burst_time+=crr[i+1].burst_time;
        for(j=i+1;j<k-1;j++)
```

```
            crr[j]=crr[j+1];
        k--;
        i--;
    }
}

int rtime=0;
for(j=0;j<n;j++){
    rtime=0;
    for(i=0;i<k;i++){
        if(crr[i].process_name==arr[j].process_name){
            arr[j].response_time=rtime;
            break;
        }
        rtime+=crr[i].burst_time;
    }
}

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'<< crr[i].process_name << "    ";
   rtime+=crr[i].burst_time;
   for(j=0;j<n;j++){
      if(arr[j].process_name==crr[i].process_name)
         arr[j].complete_time=rtime;
   }
}
cout<<"|\n";
rtime=0;
for (i=0; i<k+1; i++){
   cout << rtime << "\t";
   tArray[i]=rtime;
   rtime+=crr[i].burst_time;
}

cout<<"\n";
```

```
    cout<<"\n";
    cout<<" Process Name| Priority| Arrival Time| Burst Time| Complete Time| Turn Around Time| Wait Time| Response Time|\n";
    for (i=0; i<nop&&arr[i].process_name!='i'; i++){
        if(arr[i].process_name=='\0')
            break;
        cout <<"          P"<< arr[i].process_name;
        cout <<"        "<<arr[i].priority;
        cout <<"           " <<arr[i].arrival_time;
        cout <<"         "<< arr[i].burst_time;
        cout <<"           "<< arr[i].complete_time;
        cout <<"               "<<arr[i].wait_time+arr[i].complete_time-rtime+arr[i].burst_time;
        averageTAT+=arr[i].wait_time+arr[i].complete_time-rtime+arr[i].burst_time;
        cout <<"       "<< arr[i].wait_time+arr[i].complete_time-rtime;
        averageWaitingTime+=arr[i].wait_time+arr[i].complete_time-rtime;
        cout <<"           "<<arr[i].response_time-arr[i].arrival_time;
        averageResponseTime+=arr[i].response_time-arr[i].arrival_time;
        cout <<"\n";
    }
    cout<<"Average Waiting Time: "<<(float)averageWaitingTime/(float)n<<endl;
    cout<<"Average Turn Around Time: "<<(float)averageTAT/(float)n<<endl;
```

```
}


int main(){
    int nop,choice,i,qt;
    cout<<"Enter number of processes\n";
    cin>>nop;
    cout<<"Enter Process Name, Priority, Arrival Time, Burst Time\n";
    insert(nop);
    disp(nop,1);
    return 0;
}
```

Output:

```
D:\os lab\Tanmay-Vig_19BCS061_p7.exe
Enter number of processes
4
Enter Process Name, Priority, Arrival Time, Burst Time
1 4 0 8
2 1 2 6
3 3 5 10
4 2 6 11

Gantt Chart
|  P1   |  P2   |  P4   |  P3   |  P1   |
0       2       8       19      29      35

 Process Name| Priority| Arrival Time| Burst Time| Complete Time| Turn Around Time| Wait Time| Response Time|
          P1        4           0          8            35               35        27           0
          P2        1           2          6             8              6         0           0
          P3        3           5          10             29               24        14           14
          P4        2           6          11             19               13        2           2
Average Waiting Time: 10.75
Average Turn Around Time: 19.5

--------------------------------
Process exited after 31.2 seconds with return value 0
Press any key to continue . . .
```

Problem 8: Write a program to implement the Highest Response Ratio Next (Non-preemptive) algorithm and find the average turnaround time, waiting time, completion time and response time for overall process.

Answer:

Source Code

```
#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<limits.h>

#include<float.h>


typedef struct

{

   char process_name[3];

   int arrival_time;

   int burst_time;

   float hrr;

   int complete_time;

   int turn_around_time;

   int wait_time;
```

```
    int response_time;

    int done;

} process;



void print_process_table(process arr[],int n){

        int i;

    puts(" _____ _____ _____ _____
_____ _____ _____");

    puts("| Process Name | Arrival Time  | Burst Time | Complete Time | Turn Around Time | Wait
Time | Response Time |");

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


puts("|_____|_____|_____|_____|_____
___|_____|_____|");

       printf("|    %3s    |    %3d    |    %3d    |    %3d    |    %4d    |    %3d    |    %3d
|\n",


arr[i].process_name,arr[i].arrival_time,arr[i].burst_time,arr[i].complete_time,arr[i].turn_around_t
ime,arr[i].wait_time,arr[i].response_time);

    }


puts("|_____|_____|_____|_____|_____
___|_____|_____|");



}



void get_average(process arr[], int n){

    double tat=0,wt=0,rt=0;
```

```
    int i;
    for(i=0;i<n;i++){
        tat += (double)arr[i].turn_around_time;
        wt += (double)arr[i].wait_time;
        rt += (double)arr[i].response_time;
    }
    printf("Total time to Complete = %3d       Average Time to Complete = %.3f\n",arr[n-
1].complete_time,(double)arr[n-1].complete_time/(double)n);
    printf("Total Turn Around Time = %.3f     Average Turn Around Time =
%.3f\n",tat,tat/(double)n);
    printf("Total Waiting Time = %.3f         Average Waiting Time = %.3f\n",wt,wt/(double)n);
    printf("Total Response Time = %.3f        Average Response Time = %.3f\n",rt,rt/(double)n);
}

void gnatt(process arr[],int n){
            int i,j;
    // upper row
    printf(" ");
    for(i=0; i<n;i++){
        for(j=0;j<arr[i].burst_time+1;j++) printf("__");
        printf(" ");
    }
    printf("\n|");
    // middle row
```

```
for(i=0;i<n;i++){
    for(j=0;j<arr[i].burst_time-1;j++){
        printf(" ");
    }
    printf("%3s",arr[i].process_name);
    for(j=0;j<arr[i].burst_time;j++){
        printf(" ");
    }
    printf("|");
}
printf("\n|");
// lower row
for(i=0; i<n;i++){
    for(j=0;j<arr[i].burst_time+1;j++) printf("__");
    printf("|");
}
printf("\n");
printf("0");
for(i=0; i<n; i++) {
    for(j=0; j<arr[i].burst_time+1; j++) printf("  ");
    if(arr[i].turn_around_time > 9) printf("\b");
    printf("%d", arr[i].turn_around_time);
```

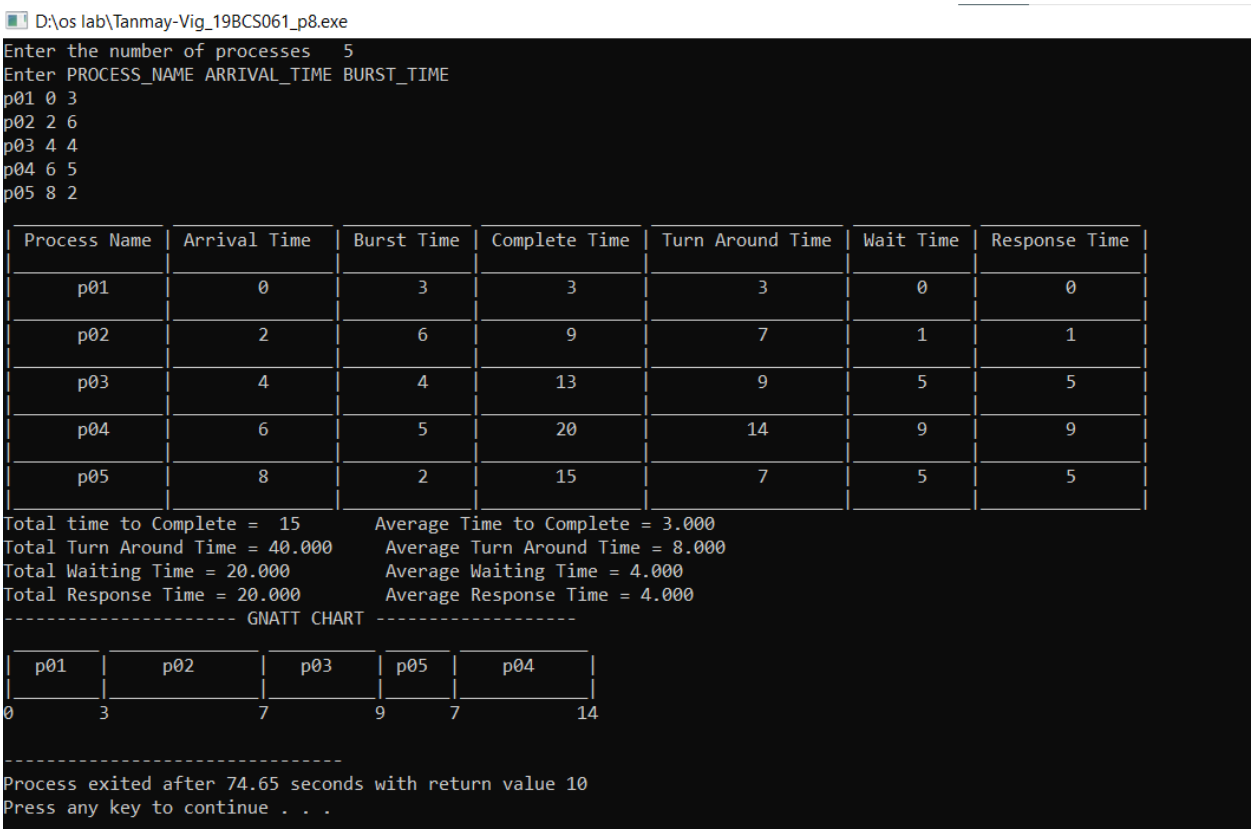
```
    }
    printf("\n");
}

int completed(process arr[], int n){
    int i=0,flag=1;
    for(i=0;i<n;i++){
        if(arr[i].done==0){
            flag=0;
            break;
        }
    }
    return flag;
}

void update_hrr(process arr[], int n, int time){
    int i=0;
    for(i=0;i<n;i++){
        if(arr[i].done==0){
            arr[i].hrr=(float)(time-arr[i].arrival_time+arr[i].burst_time)/(float)arr[i].burst_time;
        }
    }
}
```

```
int best_process(process arr[], int n, int time){
    int ind=-1,i=0;
    float priority=FLT_MIN;
    for(i=0;i<n;i++){
        if(arr[i].arrival_time > time){
            break;
        }else{
            if(arr[i].done==0 && arr[i].hrr>priority){
                priority=arr[i].hrr;
                ind=i;
            }
        }
    }
    return ind;
}

void main()
{
    int n =0,i, total_time=0,temp=0;
    printf("Enter the number of processes\t");
    scanf("%d",&n);
    process arr[n], gnt[n];
```

```
printf("Enter PROCESS_NAME ARRIVAL_TIME BURST_TIME\n");
for(i=0; i<n;i++)
{
    scanf("%s %d %d",arr[i].process_name,&arr[i].arrival_time,&arr[i].burst_time);
    arr[i].done=0;
    arr[i].hrr=1;
}

i=0;
while (completed(arr,n)!=1)
{
    update_hrr(arr,n,total_time);
    temp=best_process(arr,n, total_time); //return index of that process to execute.
    if(temp==-1){
        total_time++;
    }else{
        arr[temp].complete_time = total_time+arr[temp].burst_time;
        arr[temp].turn_around_time = arr[temp].complete_time-arr[temp].arrival_time;
        arr[temp].response_time = total_time-arr[temp].arrival_time;
        arr[temp].wait_time = arr[temp].turn_around_time-arr[temp].burst_time;
        total_time += arr[temp].burst_time;
        arr[temp].done=1;
        gnt[i++]=arr[temp];
```

```
        }
    }
    print_process_table(arr,n);
    get_average(arr, n);
    puts("---------------------- GNATT CHART -------------------");
    gnatt(gnt,n);
}
```

Output:



Problem 9:

(a) Write a program to implement the First fit memory management algorithm. Program should take input total no. of memory block, their sizes, process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.

(b) Write a program to implement the Next fit memory management algorithm. Program should take input total no. of memory block, their sizes, process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.

Answer:

(a) Source code:

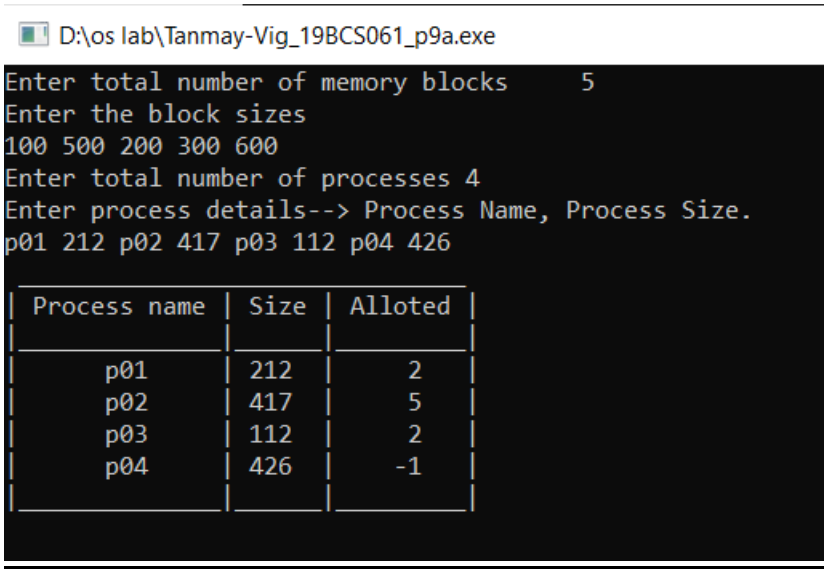
```
#include<stdio.h>

#include<limits.h>

#include<stdbool.h>
```

```
typedef struct{
    char process_name[3];
    int size,allocated;
}process;

void print_table(process pr[],int m){
    puts(" _____");
    puts("| Process name | Size | Alloted |");
    puts("|_____|_____|_____|");
    for(int i=0;i<m;i++){
    printf("|      %s     | %3d  |    %2d   |\n",
                    pr[i].process_name,pr[i].size,pr[i].allocated);
    }
    puts("|_____|_____|_____|");
}

void main()
{
    int n,m,i,j;
    printf("Enter total number of memory blocks\t");
```

```
    scanf("%d",&n);
    int mem_block[n];
    printf("Enter the block sizes\n");
    for(i=0;i<n;i++){
        scanf("%d",&mem_block[i]);
    }
    printf("Enter total number of processes\t");
    scanf("%d",&m);
    process pr[m];
    printf("Enter process details--> Process Name, Process
Size.\n");
    for(i=0;i<m;i++){
        scanf("%s %d",pr[i].process_name,&pr[i].size);
        pr[i].allocated=-1;
    }
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            if(mem_block[j]>=pr[i].size){
                mem_block[j]-=pr[i].size;
                pr[i].allocated=j+1;
```

```
            break;
        }
    }
  }
  print_table(pr,m);
}
```

Output:



(b) Source Code:

```
#include<stdio.h>


typedef struct{
  char process_name[3];
  int size,allocated;
```

```
}process;

void algorithm(int mem_block[],process pr[],int m, int n){
    int i,j,k=0;
    for(i=0;i<m;i++){
        j=k;
        while(1){
            if(mem_block[j]>=pr[i].size){
                mem_block[j]-=pr[i].size;
                pr[i].allocated=j+1;
                k=j;
                break;
            }
            j=(j+1)%n;
            if(j==k) break;
        }
    }
}

void print_table(process pr[],int m){
```

```
    puts(" _____");
    puts("| Process name | Size | Alloted |");
    puts("|_____|_____|_____|");
    for(int i=0;i<m;i++){
    printf("|      %s     | %3d  |    %2d   |\n",
                    pr[i].process_name,pr[i].size,pr[i].allocated);
    }
    puts("|_____|_____|_____|");
}

void main(){
    int n,m,i,j;
    printf("Enter total number of memory blocks\t");
    scanf("%d",&n);
    int mem_block[n];
    printf("Enter the block sizes\n");
    for(i=0;i<n;i++){
        scanf("%d",&mem_block[i]);
    }
    printf("Enter total number of processes\t");
```

```
    scanf("%d",&m);

    process pr[m];

    printf("Enter process details--> Process Name, Process
Size.\n");

    for(i=0;i<m;i++){

        scanf("%s %d",pr[i].process_name,&pr[i].size);

        pr[i].allocated=-1;

    }

    algorithm(mem_block,pr,m,n);

    print_table(pr,m);

}
```

Output:

```
($?) { .\Tanmay-Vig_19BCS061_p9b }
Enter total number of memory blocks     3
Enter the block sizes
5 10 20
Enter total number of processes 3
Enter process details--> Process Name, Process Size.
p01 10 p02 20 p03 30

 _____________________________
| Process name | Size | Alloted |
|______________|______|_________|
|      p01     |  10  |     2   |
|      p02     |  20  |     3   |
|      p03     |  30  |    -1   |
|______________|______|_________|
```

Problem 10: Write a program to implement the Best fit memory management algorithm. Program should take input total no. of memory block, their sizes, process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.

Answer:

Source code:

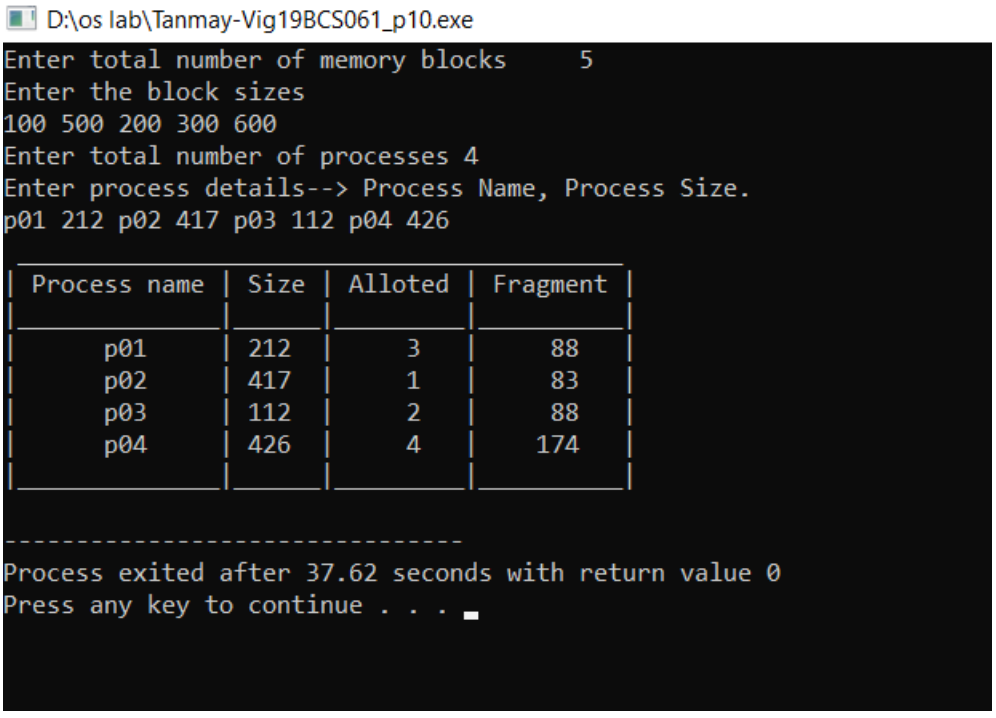
```
#include<stdio.h>

#include<stdlib.h>


typedef struct{

    char process_name[3];

    int size,allocated;

}process;


typedef struct{

    int size,fragment_size;

}mem;


void algorithm(mem mem_block[],int n, process pr[], int m){

    int i,j,best_block=-1;
```

```
    for(i=0;i<m;i++){ // iterate in process array
        best_block=-1;
        for(j=0;j<n;j++){
            if(mem_block[j].fragment_size==0 && mem_block[j].size>=pr[i].size){
                if(best_block==-1){
                    best_block=j;
                }
                else if(mem_block[best_block].size>=mem_block[j].size){
                    best_block=j;
                }
            }
        }
        pr[i].allocated=best_block;
        mem_block[best_block].fragment_size=mem_block[best_block].size-
pr[i].size;
    }
}


void print_table(process pr[],int m, mem mem_block[]){
         int i,frag;
    puts(" _____________________________________________");
    puts("| Process name | Size | Alloted | Fragment |");
    puts("|_____________|_____|_________|__________|");
```

```
    for(i=0;i<m;i++){
        if(pr[i].allocated==-1)
            frag =-1;
        else
            frag=mem_block[pr[i].allocated].fragment_size;
        printf("|      %s     | %3d  |    %2d   |    %3d   |\n",
                    pr[i].process_name,pr[i].size,pr[i].allocated,frag);
    }
    puts("|_____________|_____|_________|___________|");
}

void main(){
    int n,m,i,j;
    printf("Enter total number of memory blocks\t");
    scanf("%d",&n);
    mem mem_block[n];
    printf("Enter the block sizes\n");
    for(i=0;i<n;i++){
        scanf("%d",&mem_block[i].size);
        mem_block[i].fragment_size=0;
    }
    printf("Enter total number of processes\t");
```

```
    scanf("%d",&m);
    process pr[m];
    printf("Enter process details--> Process Name, Process Size.\n");
    for(i=0;i<m;i++){
        scanf("%s %d",pr[i].process_name,&pr[i].size);
        pr[i].allocated=-1;
    }
    algorithm(mem_block,n,pr,m);
    print_table(pr,m,mem_block);
}
```

Output:



Program 11: Write a program to implement the worst fit memory management algorithm. The program should take input total no. of the memory block, their sizes, process name, and process size. The output of the program should give the details about memory allocated to process with fragmentation detail.

Answer:

Source Code:

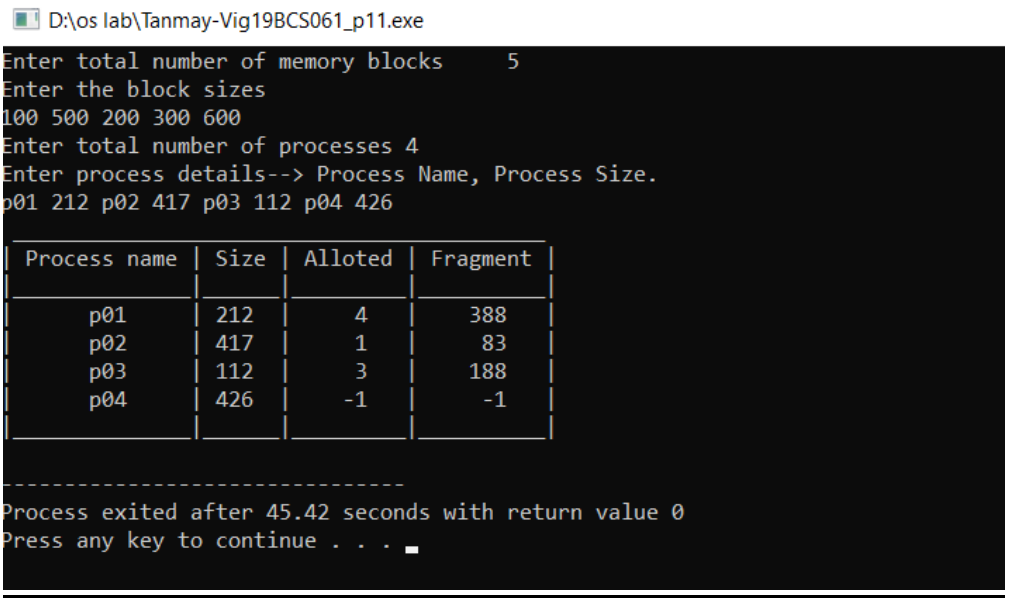
```
#include<stdio.h>

#include<stdlib.h>


typedef struct{

    char process_name[3];

    int size,allocated;

}process;


typedef struct{

    int size,fragment_size,allocated;

}mem;


void algorithm(mem mem_block[],int n, process pr[], int m){

   int i,j,ind=-1;
```

```
  for(i=0;i<m;i++){
      ind=-1;
      for(j=0;j<n;j++){
         if(mem_block[j].allocated==0){
            if(ind==-1){ //check if ind is alloted
               ind=j;
            }
            if(mem_block[j].size>=mem_block[ind].size){ // finding biggest mem
block
               ind=j;
            }
         }
      }
      if(mem_block[ind].size>=pr[i].size){
         mem_block[ind].fragment_size=mem_block[ind].size-pr[i].size;
         pr[i].allocated=ind;
         mem_block[ind].allocated=1;
      }
   }
}

void print_table(process pr[],int m, mem mem_block[]){
        int i,frag;
```

```
    puts(" _____________________________________________");
    puts("| Process name | Size | Alloted | Fragment |");
    puts("|_____________|_____|_________|__________|");
    for(i=0;i<m;i++){
        if(pr[i].allocated==-1)
            frag =-1;
        else
            frag=mem_block[pr[i].allocated].fragment_size;
        printf("|      %s     | %3d  |    %2d   |    %3d   |\n",
                    pr[i].process_name,pr[i].size,pr[i].allocated,frag);
    }
    puts("|_____________|_____|_________|__________|");
}

void main(){
    int n,m,i,j;
    printf("Enter total number of memory blocks\t");
    scanf("%d",&n);
    mem mem_block[n];
    printf("Enter the block sizes\n");
    for(i=0;i<n;i++){
        scanf("%d",&mem_block[i].size);
```

```
        mem_block[i].fragment_size=0;
        mem_block[i].allocated=0;
    }
    printf("Enter total number of processes\t");
    scanf("%d",&m);
    process pr[m];
    printf("Enter process details--> Process Name, Process Size.\n");
    for(i=0;i<m;i++){
        scanf("%s %d",pr[i].process_name,&pr[i].size);
        pr[i].allocated=-1;
    }
    algorithm(mem_block,n,pr,m);
    print_table(pr,m,mem_block);
}
```

Output:



Program 12: Write a program to implement the First In First Out (FIFO) page replacement algorithm. Program should take input reference string and total no. of pages that can accommodate in memory. Output contains detail about each page fault details and calculate average page fault.

Answer:

Source Code:

```
#include<iostream>

#include <vector>

#include <queue>

#include <string>

using namespace std;


float algorithm(int max_page,queue<int> programs){

   int fault=0,oldest=0;

   vector<int> page;

   for(int i=0;i<max_page;i++){

      page.push_back(programs.front());

      cout<<"Number of Faults= 1.\nAdding "<<programs.front()<<" to Page Table."<<endl;

      cout<<"----------------------------------------"<<endl;

      programs.pop();

      fault+=1;

   }

   while(!programs.empty()){

      bool mila=false;
```

```
        for (int i=0; i<=(int)page.size(); i++){
            if(programs.front()==page[i]){
                mila=true;
                break;
            }
        }
        if(mila){
            cout<<"Number of Faults= 0."<<endl;
            cout<<"----------------------------------------"<<endl;
        }else{
            fault+=1;
            cout<<"Number of Faults= 1.\nReplacing "<<page[oldest]<<" with "<<programs.front()<<"."<<endl;
            cout<<"----------------------------------------"<<endl;
            page[oldest]=programs.front();
            oldest= ++oldest % (int)page.size();
        }
        programs.pop();
    }
    return (float)fault;
}
int main(){
    queue<int> programs;
    int max_page,n,ind=0,t;
    string s,temp;
    cout<< "Enter Maximum Page Holding Capacity"<<endl;
    cin>>max_page;
    cout<< "Enter Page Refrencing String(to end reading enter $):\n";
```

```
    while(true){
        cin>>t;
        cin>>temp;
        programs.push(t);
        if(temp[0]=='$'){
            break;
        }
    }
    float total_faults=algorithm(max_page, programs);
    cout<<"Average Page Fault= "<<total_faults/programs.size()<<endl;
    return 0;
}
```

Output:

```
D:\os lab\Tanmay-Vig19BCS061_p12.exe
Enter Maximum Page Holding Capacity
4
Enter Page Refrencing String(to end reading enter $):
 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1$
Number of Faults= 1.
Adding 0 to Page Table.
---------------------------------------
Number of Faults= 1.
Adding 2 to Page Table.
---------------------------------------
Number of Faults= 1.
Adding 1 to Page Table.
---------------------------------------
Number of Faults= 1.
Adding 6 to Page Table.
---------------------------------------
Number of Faults= 1.
Replacing 0 with 4.
---------------------------------------
Number of Faults= 1.
Replacing 2 with 0.
---------------------------------------
Number of Faults= 0.
---------------------------------------
Number of Faults= 0.
---------------------------------------
Number of Faults= 1.
Replacing 1 with 3.
---------------------------------------
Number of Faults= 1.
Replacing 6 with 1.
---------------------------------------
Number of Faults= 1.
Replacing 4 with 2.
---------------------------------------
Number of Faults= 0.
---------------------------------------
Average Page Fault= 0.75

--------------------------------
Process exited after 16.81 seconds with return value 0
Press any key to continue . . .
```

Program 13:

(a) Write a program to implement the FCFS elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.

(b) Write a program to implement the SSTF elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.

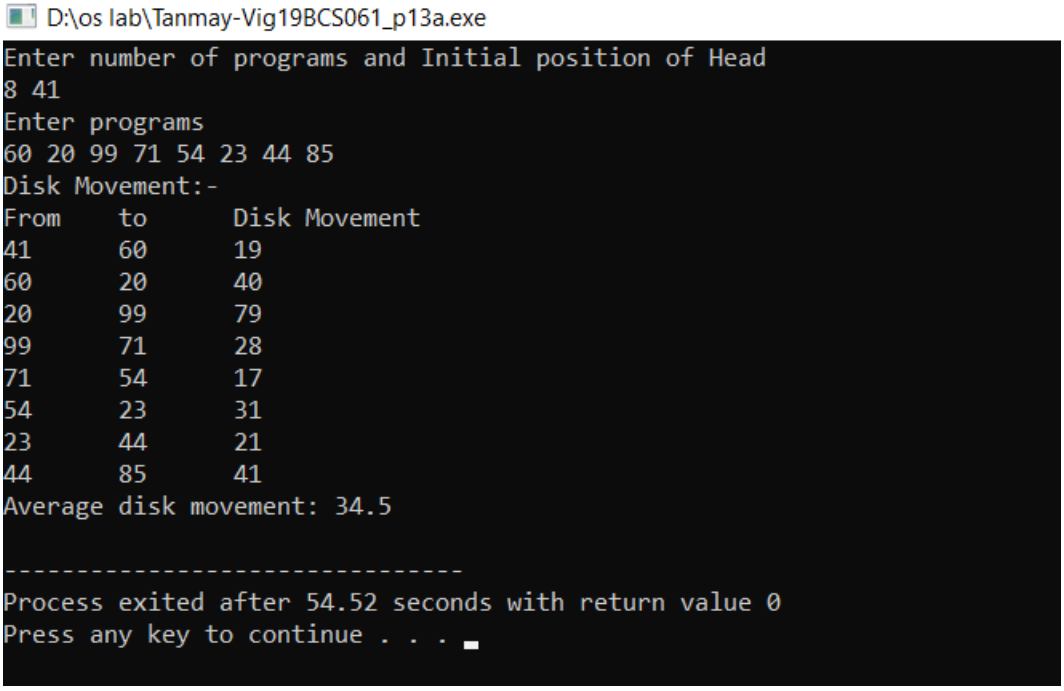
Answer:

a) Source code:

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;
int algo(vector<int> programs,int pos){
   cout<<"Disk Movement:-"<<endl;
   cout<<"From\tto\tDisk Movement"<<endl;
   int sum=0,diff;
   for (int i=0; i<(int)programs.size();i++){
      diff=abs(pos-programs[i]);
      sum+=diff;
      cout<<pos<<"\t"<<programs[i]<<"\t"<<diff<<endl;
      pos=programs[i];
   }
```

```
    return sum;
}
int main()
{
    int n,pos;
    cout << "Enter number of programs and Initial position of Head"<<endl;
    cin>>n>>pos;
    vector<int> programs(n);
    cout<<"Enter programs"<<endl;
    for (int i=0; i<n;i++){
        cin>>programs[i];
    }
    int total_movements=algo(programs,pos);
    total_movements=(float)total_movements;
    cout<<"Average disk movement: "<<total_movements/(float)n<<endl;
    return 0;
}
```

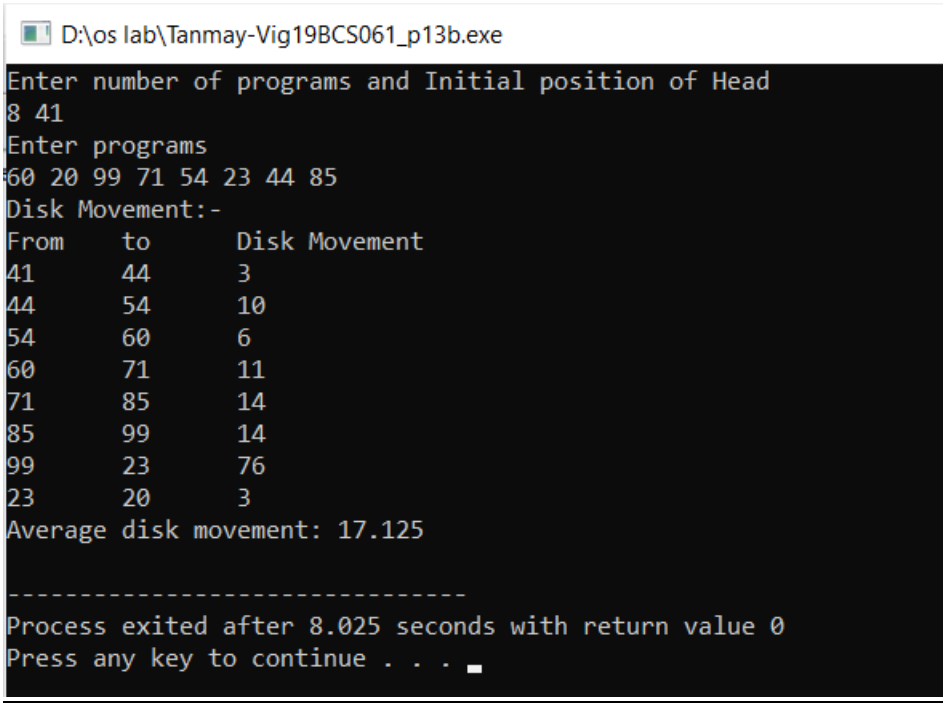
Output:



b) Source Code:

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;
int algo(vector<int> programs, int pos){
   int total_movement=0, diff,next;

   cout<<"Disk Movement:-"<<endl;
   cout<<"From\tto\tDisk Movement"<<endl;
   while(!programs.empty()){
      next=0;
      for (int i=0; i!=(int)programs.size(); i++){
         if(abs(programs[i]-pos)<abs(programs[next]-pos)){
            next=i;
         }
      }
      diff=abs(programs[next]-pos);
      total_movement+=diff;
      cout<<pos<<"\t"<<programs[next]<<"\t"<<diff<<endl;
      pos=programs[next];
      programs.erase(programs.begin()+next);
   }
   return total_movement;
}
int main()
{
   int n,pos;
   cout << "Enter number of programs and Initial position of Head"<<endl;
   cin>>n>>pos;
   vector<int> programs(n);
   cout<<"Enter programs"<<endl;
   for (int i=0; i<n;i++){
      cin>>programs[i];
   }
   int total_movements=algo(programs,pos);
   total_movements=(float)total_movements;
   cout<<"Average disk movement: "<<total_movements/(float)n<<endl;
   return 0;
}
```

Output:


D:\os lab\Tanmay-Vig19BCS061_p13b.exe
Enter number of programs and Initial position of Head
8 41
Enter programs
60 20 99 71 54 23 44 85
Disk Movement:-
From to Disk Movement
41 44 3
44 54 10
54 60 6
60 71 11
71 85 14
85 99 14
99 23 76
23 20 3
Average disk movement: 17.125

Process exited after 8.025 seconds with return value 0
Press any key to continue . . .

Program 14:

(a) Write a program to implement the SCAN elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.

(b) Write a program to implement the LOOK elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.

Answer:

(a) **Source code:**

```
#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>



using namespace std;

int search_(vector<int> programs,int lo, int hi, int x){// find point where arr[mid]<=head &&
arr[mid+1]>head

if(lo<hi){

   int mid=(hi-lo)/2;

   if(programs[mid]==x){
```

```
        return mid;
    }else if(programs[mid]<x){
        if((mid+1)==(int)programs.size() || programs[mid+1]>x) return mid;
        else search_(programs,lo,mid-1,x);
    }else{
        if((mid)==0 || programs[mid-1]<x) return mid;
        else search_(programs,mid+1,hi,x);
    }
}
return -1;
}

int piv(vector<int> &arr, int lo, int hi){
    int i=lo, p=arr[hi];
    for (int j=lo; j<=hi;j++){
        if(arr[j]<p){
            swap(arr[j],arr[i]);
            i+=1;
        }
    }
    swap(arr[hi],arr[i]);
    return i;
}

void sort_(vector<int> &arr,int lo, int hi){
```

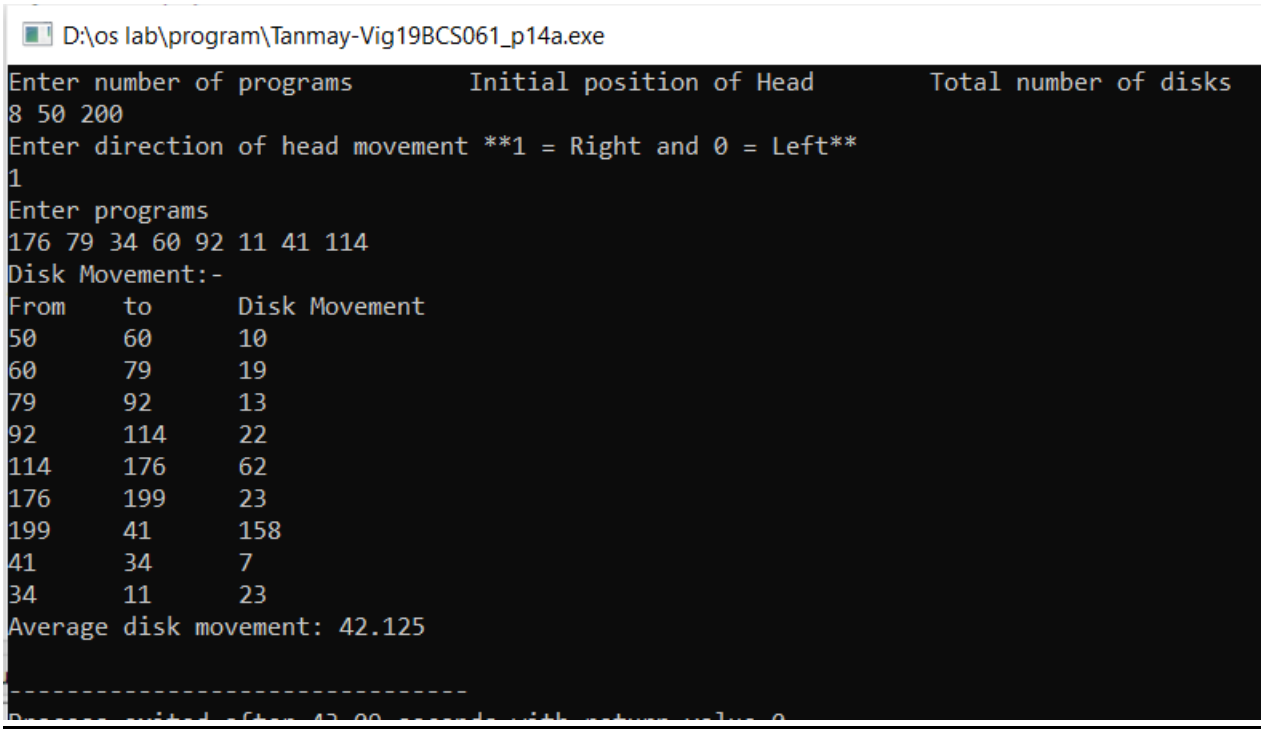
```
    if(lo<hi){
        int p=piv(arr,lo,hi);
        sort_(arr,lo,p-1);
        sort_(arr,p+1,hi);
    }
}

int left_move(vector<int> programs,int ind,int pos){
    int sum=0;
    for(int i=ind;i>=0;i--){
        sum+=abs(programs[i]-pos);
        cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
        pos=programs[i];
    }
    return sum;
}

int right_move(vector<int> programs, int ind, int pos){
    int sum=0;
    for(int i=ind+1;i<programs.size();i++){
        sum+=abs(programs[i]-pos);
        cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
        pos=programs[i];
    }
    return sum;
```

```
}

int algo(vector<int> programs,int pos,int dir,int disk){
   int sum=0,diff, n=(int)programs.size();

   cout<<"Disk Movement:-"<<endl;
   cout<<"From\tto\tDisk Movement"<<endl;

   sort_(programs,0,n-1); //sorting
   int ind=search_(programs,0,n-1,pos); //searching nearest index(0 based)
   if(programs[ind]>pos) ind-=1;

   if(dir==0){
      // for left side
      sum+=left_move(programs,ind, pos);
      //to zero
      sum+=programs[0];
      cout<<programs[0]<<"\t"<<0<<"\t"<<programs[0]<<endl;

      //for right
      sum+=right_move(programs,ind,0);
   }else{
      // for left side
      sum+=right_move(programs,ind, pos);
      //to end
```

```
        sum+=(disk-1-programs[n-1]);
        cout<<programs[n-1]<<"\t"<<disk-1<<"\t"<<disk-1-programs[n-1]<<endl;

        //for right
        sum+=left_move(programs,ind,disk-1);
    }
    return sum;
}
int main()
{
    int n,pos,dir,disk;
    cout << "Enter number of programs\tInitial position of Head\tTotal number of disks"<<endl;
    cin>>n>>pos>>disk;
    cout << "Enter direction of head movement **1 = Right and 0 = Left**"<<endl;
    cin>>dir;
    vector<int> programs(n);
    cout<<"Enter programs"<<endl;
    for (int i=0; i<n;i++){
        cin>>programs[i];
    }
    int total_movements=algo(programs,pos,dir,disk);
    total_movements=(float)total_movements;
    cout<<"Average disk movement: "<<total_movements/(float)n<<endl;
    return 0;
}
```

Output:

```
Enter number of programs        Initial position of Head        Total number of disks
8 50 200
Enter direction of head movement **1 = Right and 0 = Left**
0
Enter programs
176 79 34 60 92 11 41 114
Disk Movement:-
From    to      Disk Movement
50      41      9
41      34      7
34      11      23
11      0       11
0       60      60
60      79      19
79      92      13
92      114     22
114     176     62
Average disk movement: 28.25
```



(b) Source code:

```
#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

using namespace std;
```

```
int search_(vector<int> programs,int lo, int hi, int x){// find point where arr[mid]<=head &&
arr[mid+1]>head

if(lo<hi){

   int mid=(hi-lo)/2;

   if(programs[mid]==x){

      return mid;

   }else if(programs[mid]<x){

      if((mid+1)==(int)programs.size() || programs[mid+1]>x) return mid;

      else search_(programs,lo,mid-1,x);

   }else{

      if((mid)==0 || programs[mid-1]<x) return mid;

      else search_(programs,mid+1,hi,x);

   }

}

return -1;

}



int piv(vector<int> &arr, int lo, int hi){

   int i=lo, p=arr[hi];

   for (int j=lo; j<=hi;j++){

      if(arr[j]<p){

         swap(arr[j],arr[i]);

         i+=1;

      }

   }

   swap(arr[hi],arr[i]);
```

```
    return i;
}

void sort_(vector<int> &arr,int lo, int hi){
    if(lo<hi){
        int p=piv(arr,lo,hi);
        sort_(arr,lo,p-1);
        sort_(arr,p+1,hi);
    }
}

int left_move(vector<int> programs,int ind,int pos){
    int sum=0;
    for(int i=ind;i>=0;i--){
        sum+=abs(programs[i]-pos);
        cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
        pos=programs[i];
    }
    return sum;
}
int right_move(vector<int> programs, int ind, int pos){
    int sum=0;
    for(int i=ind+1;i<programs.size();i++){
        sum+=abs(programs[i]-pos);
        cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
```

```
        pos=programs[i];
    }
    return sum;
}

int algo(vector<int> programs,int pos, int dir){
    int sum=0,diff, n=(int)programs.size();

    cout<<"Disk Movement:-"<<endl;
    cout<<"From\tto\tDisk Movement"<<endl;

    sort_(programs,0,n-1); //sorting
    int ind=search_(programs,0,n-1,pos); //searching nearest index(0 based)
    if(programs[ind]>pos) ind-=1;

    if(dir==0){
        // for left side
        sum += left_move(programs,ind,pos);
        //for right
        sum+= right_move(programs,ind,programs[0]);
    }else{
        //for right
        sum+= right_move(programs,ind,pos);
        // for left side
        sum += left_move(programs,ind,programs[n-1]);
```

```
    }
    return sum;
}
int main()
{
    int n,pos,dir;
    cout << "Enter number of programs and Initial position of Head"<<endl;
    cin>>n>>pos;
    cout << "Enter direction of head movement **1 = Right and 0 = Left**"<<endl;
    cin>>dir;
    vector<int> programs(n);
    cout<<"Enter programs"<<endl;
    for (int i=0; i<n;i++){
        cin>>programs[i];
    }
    int total_movements=algo(programs,pos,dir);
    total_movements=(float)total_movements;
    cout<<"Average disk movement: "<<total_movements/(float)n<<endl;
    return 0;
}
```

Output:

D:\os lab\program\Tanmay-Vig19BCS061_p14b.exe

```
Enter number of programs and Initial position of Head
8 50
Enter direction of head movement **1 = Right and 0 = Left**
0
Enter programs
176 79 34 60 92 11 41 114
Disk Movement:-
From    to      Disk Movement
50      41      9
41      34      7
34      11      23
11      60      49
60      79      19
79      92      13
92      114     22
114     176     62
Average disk movement: 25.5

--------------------------------
```

D:\os lab\program\Tanmay-Vig19BCS061_p14b.exe

```
Enter number of programs and Initial position of Head
8 50
Enter direction of head movement **1 = Right and 0 = Left**
1
Enter programs
176 79 34 60 92 11 41 114
Disk Movement:-
From    to      Disk Movement
50      60      10
60      79      19
79      92      13
92      114     22
114     176     62
176     41      135
41      34      7
34      11      23
Average disk movement: 36.375
```

Program 15:

(a) Write a program to implement the C-SCAN elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.

(b) Write a program to implement the C-LOOK elevator disk scheduling algorithm. The program should give detail about each disk movement from starting head position (input from the user) and calculate average head movement.

Answer:

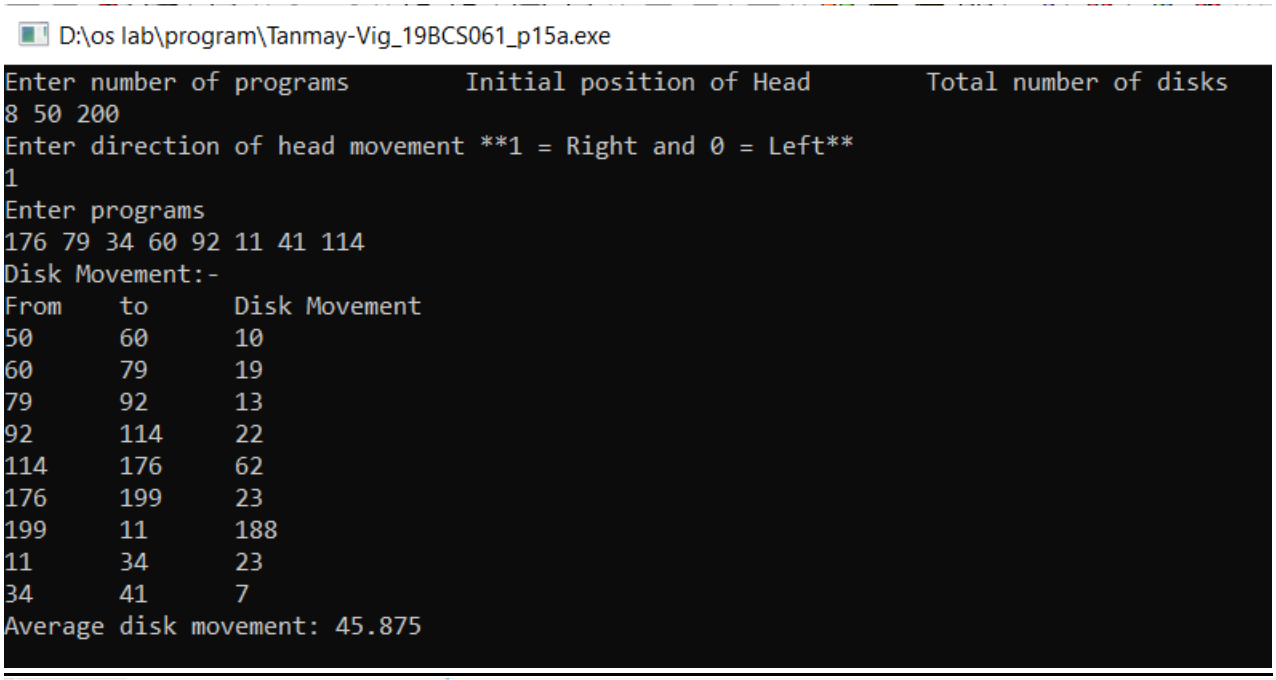
(a) Source Code:

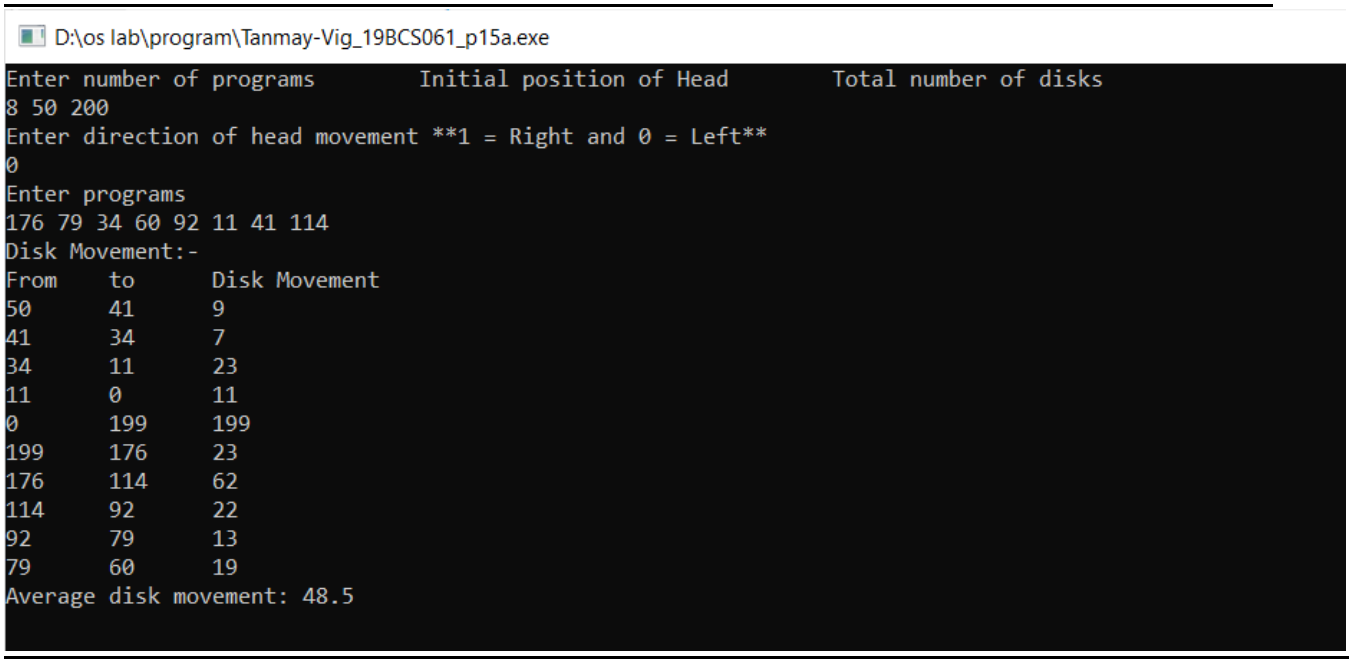
```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>

using namespace std;
int search_(vector<int> programs,int lo, int hi, int x){// find point where arr[mid]<=head &&
arr[mid+1]>head
if(lo<hi){
   int mid=(hi-lo)/2;
   if(programs[mid]==x){
      return mid;
   }else if(programs[mid]<x){
      if((mid+1)==(int)programs.size() || programs[mid+1]>x) return mid;
      else search_(programs,lo,mid-1,x);
   }else{
      if((mid)==0 || programs[mid-1]<x) return mid;
      else search_(programs,mid+1,hi,x);
   }
}
return -1;
}
```

```
int piv(vector<int> &arr, int lo, int hi){
   int i=lo, p=arr[hi];
   for (int j=lo; j<=hi;j++){
      if(arr[j]<p){
         swap(arr[j],arr[i]);
         i+=1;
      }
   }
   swap(arr[hi],arr[i]);
   return i;
}

void sort_(vector<int> &arr,int lo, int hi){
   if(lo<hi){
      int p=piv(arr,lo,hi);
      sort_(arr,lo,p-1);
      sort_(arr,p+1,hi);
   }
}


int left_move(vector<int> programs, int from, int to, int pos){
   int sum=0;
   for(int i=from; i>=to; i--){
      sum+=abs(programs[i]-pos);
      cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
      pos=programs[i];
   }
   return sum;
}

int right_move(vector<int> programs, int from , int to, int pos){
   int sum=0;
   for(int i=from+1;i<=to;i++){
      sum+=abs(programs[i]-pos);
      cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
      pos=programs[i];
   }
   return sum;
}

int algo(vector<int> programs,int pos,int dir,int disk){
   programs.push_back(disk-1);
   programs.push_back(0);
   int sum=0,diff, n=(int)programs.size();

   sort_(programs,0,n-1); //sorting
   int ind=search_(programs,0,n-1,pos); //searching nearest index(0 based)
```

```
    if(programs[ind]>pos) ind-=1;

    cout<<"Disk Movement:-"<<endl;
    cout<<"From\tto\tDisk Movement"<<endl;

    if(dir==0){
        // for left side
        sum+=left_move(programs,ind, 0, pos);
        sum+=left_move(programs,n-1,ind+1,0);
    }else{
        // for left side
        sum+=right_move(programs,ind,n-1,pos);
        sum+=right_move(programs,0,ind,disk-1);
    }
    return sum;
}
int main()
{
    int n,pos,dir,disk;
    cout << "Enter number of programs\tInitial position of Head\tTotal number of disks"<<endl;
    cin>>n>>pos>>disk;
    cout << "Enter direction of head movement **1 = Right and 0 = Left**"<<endl;
    cin>>dir;
    vector<int> programs(n);
    cout<<"Enter programs"<<endl;
    for (int i=0; i<n;i++){
        cin>>programs[i];
    }
    int total_movements=algo(programs,pos,dir,disk);
    total_movements=(float)total_movements;
    cout<<"Average disk movement: "<<total_movements/(float)n<<endl;
    return 0;
}
```

Output:





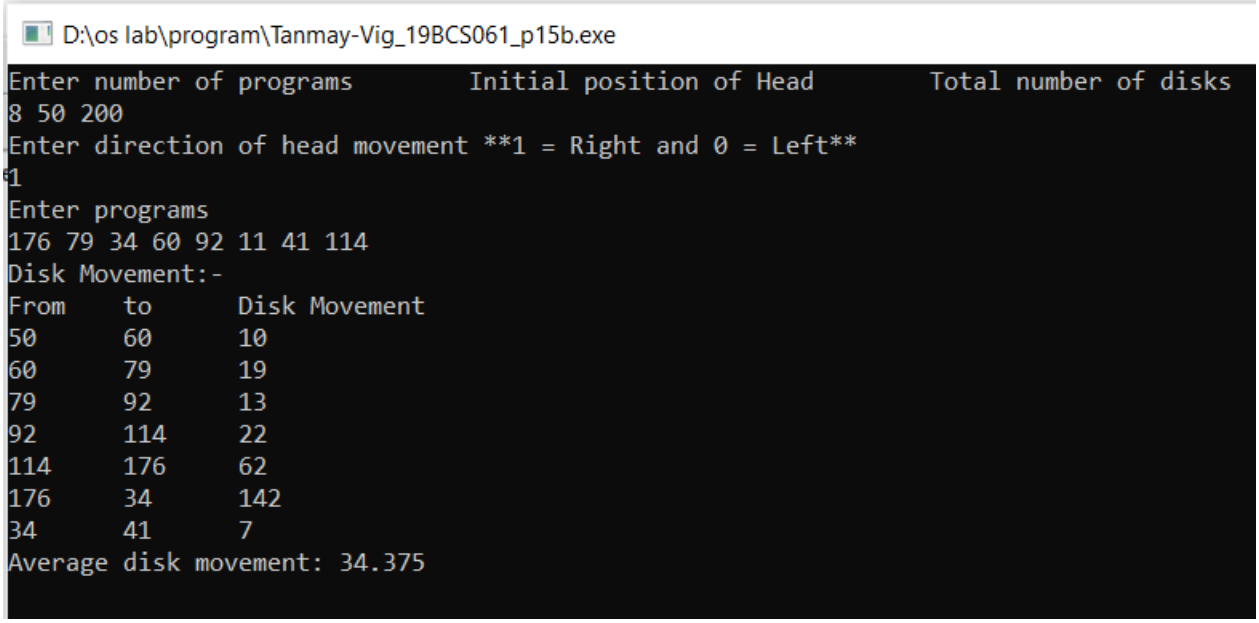
(b) Source code:

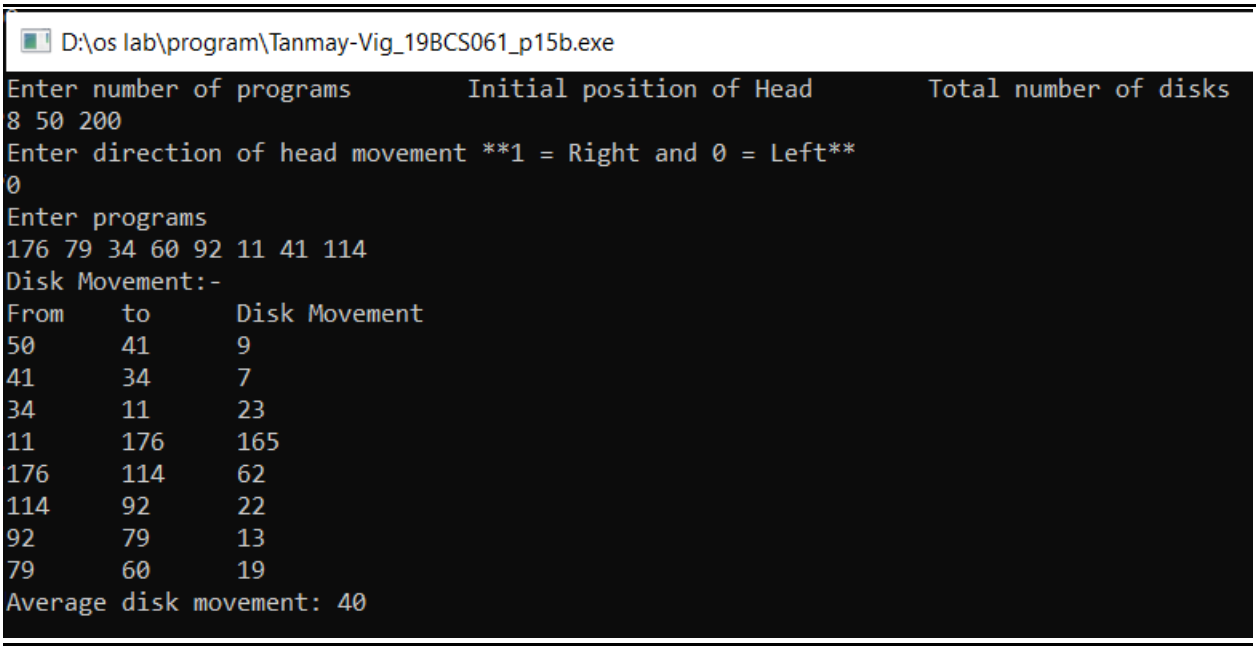
```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>

using namespace std;
int search_(vector<int> programs,int lo, int hi, int x){// find point where arr[mid]<=head &&
arr[mid+1]>head
if(lo<hi){
   int mid=(hi-lo)/2;
   if(programs[mid]==x){
      return mid;
   }else if(programs[mid]<x){
```

```
        if((mid+1)==(int)programs.size() || programs[mid+1]>x) return mid;
        else search_(programs,lo,mid-1,x);
    }else{
        if((mid)==0 || programs[mid-1]<x) return mid;
        else search_(programs,mid+1,hi,x);
    }
}
return -1;
}

int piv(vector<int> &arr, int lo, int hi){
    int i=lo, p=arr[hi];
    for (int j=lo; j<=hi;j++){
        if(arr[j]<p){
            swap(arr[j],arr[i]);
            i+=1;
        }
    }
    swap(arr[hi],arr[i]);
    return i;
}

void sort_(vector<int> &arr,int lo, int hi){
    if(lo<hi){
        int p=piv(arr,lo,hi);
        sort_(arr,lo,p-1);
        sort_(arr,p+1,hi);
    }
}


int left_move(vector<int> programs, int from, int to, int pos){
    int sum=0;
    for(int i=from; i>=to; i--){
        sum+=abs(programs[i]-pos);
        cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
        pos=programs[i];
    }
    return sum;
}

int right_move(vector<int> programs, int from , int to, int pos){
    int sum=0;
    for(int i=from+1;i<=to;i++){
        sum+=abs(programs[i]-pos);
        cout<<pos<<"\t"<<programs[i]<<"\t"<<abs(programs[i]-pos)<<endl;
        pos=programs[i];
    }
    return sum;
```

```
}

int algo(vector<int> programs,int pos,int dir,int disk){
   int sum=0,diff, n=(int)programs.size();

   sort_(programs,0,n-1); //sorting
   int ind=search_(programs,0,n-1,pos); //searching nearest index(0 based)
   if(programs[ind]>pos) ind-=1;
   cout<<"Disk Movement:-"<<endl;
   cout<<"From\tto\tDisk Movement"<<endl;

   if(dir==0){
      // for left side
      sum+=left_move(programs,ind, 0, pos);
      sum+=left_move(programs,n-1,ind+1,programs[0]);
   }else{
      // for left side
      sum+=right_move(programs,ind,n-1,pos);
      sum+=right_move(programs,0,ind,programs[n-1]);
   }
   return sum;
}
int main()
{
   int n,pos,dir,disk;
   cout << "Enter number of programs\tInitial position of Head\tTotal number of disks"<<endl;
   cin>>n>>pos>>disk;
   cout << "Enter direction of head movement **1 = Right and 0 = Left**"<<endl;
   cin>>dir;
   vector<int> programs(n);
   cout<<"Enter programs"<<endl;
   for (int i=0; i<n;i++){
      cin>>programs[i];
   }
   int total_movements=algo(programs,pos,dir,disk);
   total_movements=(float)total_movements;
   cout<<"Average disk movement: "<<total_movements/(float)n<<endl;
   return 0;
}
```

Output:


D:\os lab\program\Tanmay-Vig_19BCS061_p15b.exe
Enter number of programs Initial position of Head Total number of disks
8 50 200
Enter direction of head movement **1 = Right and 0 = Left**
1
Enter programs
176 79 34 60 92 11 41 114
Disk Movement:-
From to Disk Movement
50 60 10
60 79 19
79 92 13
92 114 22
114 176 62
176 34 142
34 41 7
Average disk movement: 34.375


D:\os lab\program\Tanmay-Vig_19BCS061_p15b.exe
Enter number of programs Initial position of Head Total number of disks
8 50 200
Enter direction of head movement **1 = Right and 0 = Left**
0
Enter programs
176 79 34 60 92 11 41 114
Disk Movement:-
From to Disk Movement
50 41 9
41 34 7
34 11 23
11 176 165
176 114 62
114 92 22
92 79 13
79 60 19
Average disk movement: 40