INDEX

Name: Ashish
1BM22CS056

Std: ____ Sec: ____

Roll No. ____ Subject ____ School/College ____

School/College Tel. No. ____ Parents Tel. No. ____

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1	15/5/24	Lab-2 ~~FCFS~~ FCFS & SJF	1	
2	22/5/24	Lab-3 a) Priority b) Round Robin	7	
3	5/6/24	Lab-4 Rate-Monotonic & Earliest deadline first		
4	12/6/24	Lab-5 semaphores Dining - Philosophers		
5	18-6-24	Lab-6 Bankers algorithm, Deadlock Detection		
6	3-7-24	Lab-7 Worst fit, best fit, First-fit		
7	10-7-24	Lab-8 FCFS, SCAN, C-SCAN		

Sl.1

Sl.1 to 7/24

Q. Write a c program to simulate the following non-preemptive CPU scheduling algorithm to find turnaround time & waiting time using ① FCFS ② SJF

Date 15/5/24
Page 1

Lab-2

```
#include <stdio.h>

int n, i, j, pos, temp, choice,
Burst_time[20], Waiting_time[20],
Turn_around_time[20], process[20],
total=0;

float avg_Turn_around_time = 0,
avg_waiting_time = 0;

int FCFS() {
    Waiting_time[0] = 0;

    for (i=1; i<n; i++){
        Waiting_time[i] = 0;
        for (j=0; j<i; j++){
            Waiting_time[i] += Burst_time[j];
        }
    }

printf("\n Process \t\t Burst Time \t\t
        Waiting Time \t\t Turnaround Time");

for (i=0; i<n; i++){
    Turn_around_time[i] =
    Burst_time[i] + Waiting_time[i];
    avg_Waiting_time += Waiting_time[i];
    avg_Turn_around_time += Turn_around_time[i];
    printf("\n P[%d] \t\t %d \t\t\t %d \t\t\t %d",
            i+1, Burst_time[i],
            Waiting_time[i], Turn_around_time[i]);
}
```

```
avg_waiting_time = (float)(avg_waiting_time)
                                /(float)i;
avg_Turn_around_time = (float)(avg_Turn_
    around_time)/(float)i;
printf("\nAverage Waiting Time : %.2f",
    avg_waiting_time);
printf("\nAverage Turnaround Time : %.2f\n",
    avgTurn_around_time);
return 0;
}

int SJF(){
  for(i=0; i<n; i++){
    pos = i;
    for(j=i+1; j<n; j++){
      if(Burst_time[j] < Burst_time[pos])
        pos = j;
    }
    temp = Burst_time[i];
    Burst_time[i] = Burst_time[pos];
    Burst_time[pos] = temp;

    temp = process[i];
    process[i] = process[pos];
    process[pos] = temp;
  }
  waiting_time[0] = 0;

  for(i=1; i<n; i++){
    waiting_time[i] = 0;
    for(j=0; j<i; j++)
      waiting_time[i] += Burst_time[j];
```

```
    total += waiting_time[i];
}
avg_waiting_time = (float) total/n;
total = 0;

printf("\n Process \t\t Burst time \t\t
    Waiting Time \t\t Turn around Time");

for (i=0; i<n; i++){
    Turn_around_time[i] = Burst_time[i]+
                          waiting_time[i];
    total += Turn_around_time[i];
    printf("\n P[%d] \t\t %d \t\t %d \t\t
        \t\t %d", process[i],
        Burst_time[i], waiting_time[i],
        Turn_around_time[i]);
}

avg_Turn_around_time = (float) total/n;
printf("\n\n Average Waiting Time = %f",
        avg_waiting_time);
printf("\n Average Turnaround Time = %f\n",
        avg_Turn_around_time);
}

int main(){
    printf("Enter the total no. of processes:");
    scanf("%d", &n);

    printf("\n Enter Burst Time:\n");
```

```
for (i=0; i<n; i++){
    printf("P[%d]: ", i+1);
    scanf("%d", &burst_time[i]);
    process[i] = i+1;
}

while (1){
    printf("\n ---MAIN MENU----\n");
    printf("1. FCFS Scheduling\n
            2. SJF Scheduling\n");

printf("\n Enter your choice:");
scanf("%d", &choice);

switch (choice){
    case 1: FCFS();
    break;

    case 2: SJF();
    break;

    default: printf("Invalid Input!!!");
    }
}
    return 0;
}
```

Output:

Enter the total no. of processes : 3

Enter Burst Time :
P[1] : 10
P[2] : 5
P[3] : 8

---- MAIN MENU ----
1. FCFS Scheduling
2. SJF Scheduling

Enter your choice : 1

Process	Burst Time	Waiting Time	Turnaround Time
P[1]	10	0	10
P[2]	5	10	15
P[3]	8	15	23

Average Waiting Time : 8.33
Average Turnaround Time : 16.00

---- MAIN MENU ----
1. FCFS Scheduling
2. SJF Scheduling

Enter your choice : 2

Date / /
Page 6

Process	Burst Time	Waiting Time	Turnaround Time
P[2]	5	0	5
P[3]	8	5	13
P[1]	10	13	23

Average Waiting Time = 6.000000
Average Turnaround Time = 13.666667

SJF
15/5/24

Date 22/5/24
Page 7

Lab-3

Q. Write a C program to simulate the following CPU scheduling algorithm to find turnaround time & waiting time.

a) Priority (pre-emptive & Non-pre-emptive)

b) Round Robin (Experiment with different quantum sizes for RR algorithm).

a)

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

struct process{
    int process_id;
    int burst_time;
    int priority;
    int waiting_time;
    int turnaround_time;
};

void find_average_time(struct process[], int);

void priority_scheduling(struct process[], int);

int main(){
    int n, i;
    struct process proc[10];

    printf("Enter the no. of processes: ");
    scanf("%d", &n);
```

```
for (int i=0; i<n; i++){
    printf("\n Enter the process ID: ");
    scanf("%d", &proc[i].process_id);
    printf("Enter the burst time: ");
    scanf("%d", &proc[i].burst_time);
    printf("Enter the priority: ");
    scanf("%d", &proc[i].priority);
}
priority_scheduling(proc, n);
return 0;
}

void find_waiting_time(struct process
        proc[], int n, int wt[]){
int i;
wt[0] = 0;

for(i=1; i<n; i++){
    wt[i] = proc[i-1].burst_time
          + wt[i-1];
}
}

void find_average_time(struct proces
        proc[], int n){
int wt[0], tat[10], total_wt=0,
    total_tat=0, i;

find_waiting_time(proc, n, wt);
find_turnaround_time(proc, n, wt,
```

```
printf("\n Process ID\t Burst Time \n Priority
    \t Waiting Time \t Turnaround Time");

for(i=0; i<n; i++){
    total_wt = total_wt + wt[i];
    total_tat = total_tat + tat[i];
    printf("\n %d\t\t %d\t\t %d\t\t %d\t\t
        %d", proc[i].process_id,
        proc[i].burst_time);
printf("\n\n Average Waiting Time =
    %f", (float) total_wt/n);
printf("\n Average Turnaround Time =
    %f\n", (float) total_tat/n);
}

void priority_scheduling(struct process
        proc[], int n){
    int i, j, pos;
    struct process temp;
    for(i=0; i<n; i++){
        pos = i;
        for(j=i+1; j<n; j++){
            if(proc[j].priority < proc[pos].priority)
            pos = j;
        }
    temp = proc[i];
    proc[i] = proc[pos];
    proc[pos] = temp;
    }
    find_average_time(proc, n);
}
```

b) Round Robin (Non-pre-emtive).

```
# include <stdio.h>
# include <stdbool.h>

int turnaroundtime(int process[],
    int n, int bt[], int wt[],
    int tat[]){
    for (int i=0; i<n; i++){
        tat[i] = bt[i] + wt[i]
        return 1;
    }
}

int waitingtime (int process[], int n,
    int bt[], int wt[], int quantum){
    int rem_bt[n];
    for (int i=0; i<n; i++){
        rem_bt[i] = bt[i];
    int t=0;

    while (1){
        bool done = true;
        for (int i=0; i<n; i++){
            if (rem_bt[i]>0){
                done = false;
                if (rem_bt[i]> quantum){
                    t+= quantum;
                    rem_bt[i]-= quantum;
                }
```

```
        else {
            t = t + rem_bt[i];
            wt[i] = t - bt[i];
            rem_bt[i] = 0;
        }
    }
}
    if (done == true)
        break;
}
    return 1;
}

int findavgTime(int processes[], int n,
        int bt[], int quantum){
    int wt[n], tat[n], total_wt = 0,
        total_tat = 0;
    waiting time(processes, n, bt, wt,
        quantum);
    turnaround time(processes, n, bt,
        wt, tat);
    printf("\n\n Processes \t\t Burst time \t\t
        \t\t Waiting Time \t\t turnaround time\n");
    for(int i=0; i<n; i++){
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("\n\t %d \t\t %d \t\t\t %d \t\t\t %d
            \t\t\t %d \n", i+1, bt[i],
            wt[i], tat[i]);
    }
```

```
int main(){
    int n, processes[n], burst_time[n],
    quantum;
    printf("Enter the Number of
            processes:");
    scanf("%d", &n);

    printf("\n Enter the quantum
            time:");
    scanf("%d", &quantum);

    int i=0;
    for(i=0; i<n; i++){
        printf("\n Enter the process:");
        scanf("%d", &processes[i]);
        printf("Enter the Burst Time:");
        scanf("%d", &burst_time[i]);
    }

    findavgTime(processes, n, burst_time,
                quantum);
    return 0;
}
```

OUTPUT:

a) Enter the no of processes : 2

Enter the process ID : 1
Enter the burst time : 2
Enter the priority : 3

Enter the process ID : 4
Enter the burst time : 5
Enter the priority : 6

Process ID	Burst Time	Priority	Waiting Time	Turnaround Time
1	2	3	0	2
4	5	6	2	7

Average Waiting Time : 1.000000
Average Turnaround Time : 4.500000

b) Ente the no. of Processes : 3

Enter the quantum time : 2
Enter the Process : 1
Enter the Burst time : 5
Enter the Process : 2
Enter the Burst time : 7
Enter the Process : 3
Enter the Burst time : 3

Date / /
Page 14

Processes	Burst Time	Waiting Time	Turnaround Time
1	5	7	12
2	7	8	15
3	3	8	11

Average waiting time = 7.66667
Average turnaround time = 12.66667

SJF
22/5/24

Q1) Write a C program to simulate producer-customer problem using semaphores.

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

int mutex=1, full=0, empty=3, x=0;

int main(){
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1. Producer \n2. Consumer
                \n3. Exit");

while(1){
    printf("\n Enter your choice:");
    scanf("%d", &n);
    switch(n){
        case1: if((mutex==1)&&(empty!=0)
                producer();
                else;
                printf("Buffer is full!!");
                break;
        case2: if((mutex==1)&&(full!=0))
                consumer();
                else
                printf("Buffer is empty!!");
                break;
```

```
        case 3: exit(0);
                break;
        }
    }
    return 0;
}

int wait (int s){
    return (--s);
}

int signal (int s){
    return (++s);
}

void producer (){
    mutex = wait (mutex);
    full = signal (full);
    empty = wait (empty);
    x++;
    printf ("\n Producer produces the
            item %d", x);
    mutex = signal (mutex);
}

void consumer(){
    mutex = wait (mutex);
    full = wait (full);
    empty = signal (empty);
    printf ("\n Consumer consumes
            item %d", x);
    x--;
    mutex = signal (mutex);
}
```

Date __/__/__
Page _____

Output:

1. Producer
2. Consumer
3. Exit

Enter your choice: 1

Producer producer the item 1
Enter your choice: 2

Consumer 1 consumes item 1
Enter your choice: 3

Q Write a C program to simulate the concept of Dining-Philosophers problem.

Sol
```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (i+4)%N
#define RIGHT (i+1)%N

int state[N];
int phil[N] = {0,1,2,3,4};

sem_t mutex;
sem_t S[N];

void test(int i){
    if(state[i] == HUNGRY &&
     state[LEFT] != EATING &&
     state[RIGHT] != EATING){
      state[i] = EATING;
      sleep(2);
      printf("Philosopher %d takes
       fork %d & %d\n", i+1,
       LEFT+1, i+1);
      printf("Philosopher %d is
          Eating\n", i+1);
```

```
        sem_post(&S[i]);
    }
}

void * philosopher(void * num){
    while(i){
        int * i = num;
        sleep(1);
        take_fork(*i);
        sleep(0);
        put_fork(*i);
    }
}

int main(){
    int i;
    pthread_t thread_id[N];

    sem_init(&mutex, 0, 1);

    for(i=0; i<N; i++)
        sem_init(&S[i], 0, 0);

    for(i=0; i<N; i++)
        pthread_create(&thread_id[i],
        NULL, philosopher, &phil[i]);
        printf("Philosopher %d is thinking
            \n", i+1);
    }
    for(i=0; i<N; i++){
        pthread_join(thread_id[i], NULL);
    }
}
```

Output:

Philosopher 1 is thinking
Philosopher 2 is thinking
3 is thinking
4 is thinking
5 is thinking
1 is hungry
1 takes for fork 2 & 1
1 is eating
2 is hungry
2 takes for 3 & 2
2 is eating
3 is hungry
3 takes for fork 4 & 3
3 is eating
4 is hungry
4 take for 5 & 4
4 is eating
1 putting for fork 2 & 1 down
1 is thinking
1 is thinking
2 putting for fork 3 & 2 down
2 is thinking
3 putting fork 4 & 3 down
3 is thinking
4 putting fork 5 & 4 down
4 is thinking
5 putting fork 1 & 5 down
5 is thinking

Date 3/7/24
Page

Lab-7

Q) Write a C program to simulate the following contiguous memory allocation techniques:

a) Worst-fit
b) Best-fit
c) First-fit

Sol.

```
#include <stdio.h>
#define max 25

void firstFit(int b[], int nb, int f[], int nf);
void worstFit(int b[], int nb, int f[], int nf);
void bestFit(int b[], int nb, int f[], int nf);

int main(){
    int b[max], f[max], nb, nf;
    printf("Memory Management Schemes\n");
    printf("\n Enter the no. of blocks:");
    scanf("%d", &nb);
    printf("Enter the no. of files:");
    scanf("%d", &nf);
    printf("\n Enter the size of the blocks\n");

    for(int i=1; i<=nb; i++){
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }
```

```
printf("\n Memory Management
        Scheme - First Fit");
firstFit(b, nb, f, nf);

printf("\n\n Memory Mongement
    Scheme - Worst Fit");
worstFit(b, nb, f, nf);

printf("\n\n Memory Mongement
    scheme - Best fit");
bestFit(b, nb, f, nf);

return 0;
}

void firstFit(int b[], int nb,
    int f[], int nf){
    int bf[max] = {0};
    int ff[max] = {0};
    int frag[max], i, j;

    for(i=1; i<=nf; i++){
      for(j=1; j<=nb; j++){
        if(bf[j]!=1 && b[j]>=f[i]){
          ff[i] = j;
          bf[j] = 1;
          frag[i] = b[j]-f[i];
          break;
        }
      }
    }
```

```
printf("\n File_no : \t File_size : \t
  Block_no : \t Block_size : \t Fragment");

for (i=1; i<=nf; i++){
  printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d",
          i, f[i], ff[i], b[ff[i]], frag[i]);
  }
}

void worstfit (int b[], int nb, int f[],
               int nf){
  int bf[max] = {0};
  int ff[max] = {0};
  int frag[max], i, j, temp,
      highest = 0;
  for (i=1; i<=nf; i++){
    for (j=1; j<=nb; j++){
      if (bf[j] != 1){
        temp = b[j] - f[i];
        if (temp >=0 && highest <temp){
          ff[i] = j;
          highest = temp;
        }
      }
    }

  frag[i] = highest;
  bf[ff[i]] = 1;
  highest = 0;
}
```

Date ___/___/___
Page _____

```
void bestFit (int b[], int nb,
              int f[], int nf){
  int bf[max] = {0};
  int ff[max] = {0};
  int frag[max], i, j, temp,
  lowest = 10000;

  for (i=1; i<=nf; i++){
    for (j=1; j<=nb; j++){
      if (bf[j] != 1){
        temp = b[j] - f[i];
        if (temp>=0 && lowest>
                    temp){
          ff[i] = j;
          lowest = temp;
        }
      }
    }
    frag[i] = lowest;
    bf[ff[i]] = 1;
    lowest = 10000;
  }

  printf("\n File_no :\t File_size :\t
    Block_no :\t Block_size :\t Fragment");

  for (i=1; i<=nf && ff[i] != 0; i++){
    printf("\n %d\t\t %d\t\t\t %d\t\t\t %d",
      i, f[i], ff[i], b[ff[i]], frag[i]);
  }
}
```

Output:

```
Enter the no. of blocks
5
Enter the no. of processes
8
Enter the block size
100 500 200 300 600
Enter the process size
212 415 63 124 23 89 73 13

1. First-fit
2. Best-fit
3. Worst-fit
Enter your choice
1
```

Process No.	Process size	Block no.
1	212	2
2	415	5
3	63	1
4	124	2
5	23	1
6	89	2
7	73	2
8	13	1

Date __/__/__
Page _____

Q2 Write a C program to simulate page replacement algorithms:
a) FIFO
b) LRU
c) Optimal

Sol
```
#include <stdio.h>
int n, f, i, j, k;
int in[100];
int p[50];
int hit=0;
int pgfaultcnt=0;

void getData(){
    printf("\n Enter length of page reference sequence: ");
    scanf("%d", &n);
    printf("\n Enter the page reference sequence: ");
    for(i=0; i<n; i++){
        scanf("%d", &in[i]);
    printf("\n Enter no. of frames: ");
    scanf("%d", &f);
}

Void initialize(){
    pgfaultcnt=0;
    for(i=0; i<f; i++){
        p[i]=9999;
    }
}
```

```
int getHitIndex(int data){
    int hitind;
    for(k=0; k<f; k++){
        if(p[k]==data){
            hitind = k;
            break;
        }
    }
    return hitind;
}

void dispPages(){
    for(k=0; k<f; k++){
        if(p[k]!=9999){
            printf(" %d", p[k]);
        }
    }
}

void dispPgFaultCnt(){
    printf("\n Total no. of page faults: %d",
            pgfaultcnt);
}

void fifo(){
    getData();
    initialize();
    for(i=0; i<n; i++){
        printf("\n For %d: ", in[i]);

        if(isHit(in[i])==0){
            for(k=0; k<f-1; k++)
                p[k] = p[k+1];
```

```
        p[k] = in[i];
        pgfaultcnt++;
        dispPages();
    }
    else
        printf("No page fault");
}
    dispPgFaultCnt();
}

void Iru()
{
    initialize();
    int least[50];
    for(i=0; i<n; i++)
    {
        printf("\n For %d :", in[i]);

        if(isHit(in[i])==0){
        for(j=0; j<nf; j++){
            int pg = p[j];
            int found = 0;
            for(k=i-1; k>=0; k--){
                if(pg == in[k]){
                    least[j] = k;
                    found = 1;
                    break;
                }
            else
                found = 0;
        }
```

```
        if(!found){
            least[j] = -9999;
        }

        int min = 9999;
        int repindex;

        for(j=0; j<nf; j++){
            if(least[j] < min){
                min = least[j];
                repindex = j;
            }
        }

        p[repindex] = in[i];
        pgfaultcnt++;

        dispPages();
    }
    else
        printf("No Page fault!");
}
dispPgFaultCnt();
}

int main(){
    int choice;
    while(1){
    printf("\n Page Replacement Algorithms\n
    1. Enter data \n2. FIFO \n3. optimal \n4.
    LRU. 5. Exit \n Enter your choice:");
```

```
scanf("%d", &choice);
switch(choice){
  case 1: getData();
          break;
  case 2: fifo();
          break;
  case 3: Optimal();
          break;
  case 4: Lru();
          break;
  default: return 0;
          break;
  }
 }
}
```

Output:

```
Replacement Algorithms
1. Enter data
2. FIFO
3. Optimal
4. LRU
5. Exit

Enter your choice: 1
Enter length of a page reference sequence: 12
Enter the page reference sequence:
1 2 3 4 1 2 5 1 2 3 4 5
Enter no. of frames: 2
```

Date __/__/____
Page _____

Page Replacement Algorithms
1. Enter data
2. FIFO
3. Optimal
4. LRU
5. Exit
Enter your choice: 2
For 1: 1
For 2: 13
For 3: 23
For 4: 34
For 1: 41
For 2: 12
For 5: 25
For 1: 51
For 2: 12
For 3: 23
For 4: 34
For 5: 45
Total no. of page faults: 12

Date 10/7/24
Page

Lab-8

Q. Write a C program to simulate disk scheduling algorithm:

a) FCFS
b) SCAN
c) C-SCAN

a)

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

int main(){
    int RQ[100], i, n, TotalHeadMoment
    =0, initial;
    printf("Enter the no. of Requests\n");
    scanf("%d", &n);
    printf("Enter the Request Sequence\n");
    for(i=0; i<n; i++)
        scanf("%d", &RQ[i]);
    printf("Enter initial head position\n");
    scanf("%d", &initial);

    for(i=0; i<n; i++){
        TotalHeadMoment = TotalHeadMoment
            + abs(RQ[i]-initial);
        initial = RQ[i];
    }
    printf("Total head moment is
        %d", TotalHeadMoment);
        return 0;
    }
```

Date / /
Page

Output:

Enter the no. of Requests 5
Enter the Request sequence 98
183 37 122 14
Enter initial head position 53
Total head movement is 469

b)

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

int main(){
  int RQ[100], i, n, TotalHeadMovement
      = 0, initial, size, move;
  printf("Enter the no. of Requests\n");
  scanf("%d", &n);
  printf("Enter the Request sequence\n");
  for(i=0; i<n; i++)
    scanf("%d", &RQ[i]);
  printf("Enter initial head position\n");
  scanf("%d", &initial);
  printf("Enter total disk size\n");
  scanf("%d", &size);
  printf("Enter the head movement
    dir^n for high & for low 0\n");
  scanf("%d", &move);
  for(i=0; i<n; i++)
  {
    for(j=0; j<n-1; j++)
    {
      if(RQ[j]>RQ[j+1])
      {
        int temp;
        temp = RQ[j];
        RQ[j] = RQ[j+1];
        RQ[j+1] = temp;
      }
    }
  }
}
```

```
int index;
for (i=0; i<n; i++)
{
    if (initial < RQ[i])
    {
        index = i;
        break;
    }
}

if{
    for (i = index - 1; i >= 0; i--)
    {
        TotalHeadMoment = TotalHeadMoment +
            abs(RQ[i] - initial);
        initial = RQ[i];
    }
    TotalHeadMoment = TotalHeadMoment +
        abs(RQ[i+1] - 0);
    initial = 0;
    for (i=index; i<n; i++)
    {
        TotalHeadMoment = TotalHeadMoment
            + abs(RQ[i] - initial);
        initial = RQ[i];
    }
}

printf("Total head movement is %d",
        TotalHeadMoment);
return 0;
}
```

Output:

Enter the no. of Requests
8
Enter the Request sequence
98 183 37 122 14 124 65 67
Enter initial head position
53
Enter total disk size
200
Enter the head movement dirn for
high 1 & for low 0
1
Total head movement is 236

Date __/__/__
Page _____

c)

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

int main()
{
  int RQ[100], i, j, n, TotalHeadMovement
    =0, initial, size, move;
  printf("Enter the no. of Requests\n");
  scanf("%d", &n);
  printf("Enter the Request sequence\n");
  for(i=0; i<n; i++)
      scanf("%d", &RQ[i]);
  printf("Enter initial head position\n");
  scanf("%d", &initial);
  printf("Enter total Disk size\n");
  scanf("%d", &size);
  printf("Enter the head movement dir. for
  high 1 & for low 0\n");
  scanf("%d", &move);

  for(i=0; i<n; i++)
  {
    for(j=0; j<n; j++)
    {
      if(RQ[j] > RQ[j+1])
      {
        int temp;
        temp = RQ[j];
        RQ[j] = RQ[j+1];
        RQ[j+1] = temp;
      }
    }
  }
}
```

```
if (move == 1)
{
    for (i = index; i < n; i++)
    {
    TotalHeadMoment = TotalHeadMoment
        + abs(RQ[i] - initial);
    initial = RQ[i];
}

TotalHeadMoment = TotalHeadMoment
  + abs (size - RQ[i-1] - 1);
TotalHeadMoment = TotalHeadMoment
    + abs (size-1-0);
initial = 0;
for (i=0; i< index; i++)
{
  TotalHeadMoment = TotalHeadMoment
      + abs(RQ[i] - initial);
  initial = RQ[i];
  }
}

else {
  for (i = index; i>=0; i--)
  {
  TotalHeadMoment = TotalHeadMoment
  + abs(RQ[i] - initial);
  initial = RQ[i];
}
```

```
for (i = n-1; i >= index; i--)
{
    TotalHeadMoment = TotalHeadMoment
        + abs(RQ[i] - initial);
    initial = RQ[i];
}
}

printf("Total head movement is %d",
        TotalHeadMoment);
return 0;
}
```

Output:

```
Enter the no. of Requests
6
Enter the Request sequence
98
183
37
122
14
124
Enter inidial head position
53
Enter total disk size
199
Enter the head movement dir^n for high 1
& for low 0
1
Total head movement is 380
```