**CSE 316-OPERATING SYSTEMS**

**NAME**: MYLA SRICHAND **REGNO**:11806981

**ROLLNO**:16  **SEC**:K18BY

**EMAIL**:chennakesavaraomyla@gmail.com(for git hub) or srichandmyla@gmail.com

**GithubLink**:https://github.com/srichandmyla/SCAN-disk-scheduling-project.git

**Code**:

#include<stdio.h>

int main()

{

int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],

temp1=0,temp2=0;

float avg;

printf("Enter the max range of disk\n");//input should be 4999

scanf("%d",&max);

printf("Enter the initial head position\n");//input should be 143

scanf("%d",&head);

printf("Enter the size of queue request\n");//input should be 9

scanf("%d",&n);

printf("Enter the queue of disk positions to be read\n");/\*input should be 86 1470 913 1774 948 1509 1022 1750 130\*/

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

{

scanf("%d",&temp);

if(temp>=head)

{

queue1[temp1]=temp;

temp1++;

}

else

{

queue2[temp2]=temp;

temp2++;

}

}

for(i=0;i<temp1-1;i++)

{

for(j=i+1;j<temp1;j++)

{

if(queue1[i]>queue1[j])

{

temp=queue1[i];

queue1[i]=queue1[j];

queue1[j]=temp;

}

}

}

for(i=0;i<temp2-1;i++)

{

for(j=i+1;j<temp2;j++)

{

if(queue2[i]>queue2[j])

{

temp=queue2[i];

queue2[i]=queue2[j];

queue2[j]=temp;

}

}

}

for(i=1,j=0;j<temp1;i++,j++)

queue[i]=queue1[j];

queue[i]=max;

queue[i+1]=0;

for(i=temp1+3,j=0;j<temp2;i++,j++)

queue[i]=queue2[j];

queue[0]=head;

for(j=0;j<=n+1;j++)

{

diff=abs(queue[j+1]-queue[j]);

seek+=diff;

printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

printf("Total seek time is %d\n",seek);

avg=seek/(float)n;

printf("Average seek time is %f\n",avg);

return 0;

}

1. **DESCRIPTION:**

Given an array of disk track numbers and initial head position, our task is to find the total number of seek operations done to access all the requested tracks if SCAN disk scheduling algorithm is used.

**SCAN (Elevator) algorithm**  
In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the **elevator algorithm**. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait

Advantages:

* High throughput
* Low variance of response time
* Average response time

Disadvantages:

* Long waiting time for requests for locations just visited by disk arm

1. **ALGORITHM:**
2. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. ‘head’ is the position of disk head.
3. Lets take the maximum range of the disk.
4. In the direction in which head is moving service all tracks one by one.
5. Calculate the absolute distance of the track from the head.
6. Increment the total seek count with this distance.
7. Currently serviced track position now becomes the new head position.
8. Go to step 3 until we reach at one of the ends of the disk.
9. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.
10. **COMPLEXITY:**

|  |  |  |
| --- | --- | --- |
| **METHOD** | **Amortized Complexity** | **Amortized Complexity**  **(m--->ꝏ)** |
| **SCAN** | (m-1)/m.Q/W+Q/m | Q/W |
| **1-StepSCAN(FCFS)** | (m-1)/m.Q+Q/m | Q |
| **N-StepSCAN(N<W)** | (m-1)/m.Q/N+Q/m | Q/N |
| **N-StepSCAN(N>=W)** | (m-1)/m.Q/W+Q/m | Q/W |
| **Lower Bound** | Q/W | Q/W |

SCAN and N-Step SCAN with N>=W are optimal in amortized complexity, but N-Step SCAN is not optimal when N<W.We conclude that SCAN is the best disk scheduling algorithm among SCAN and N-Step SCAN in amortized sense.The 1-Step SCAN(FCFS) is the worst among them.

6.

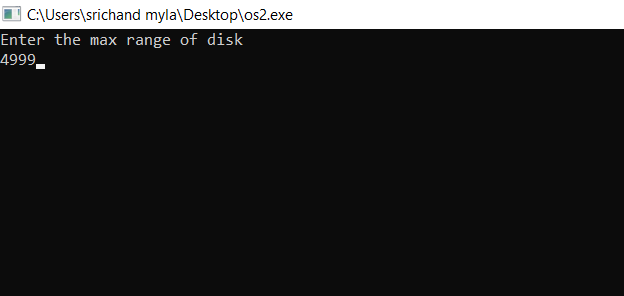
Starts at 143 ,scans up to 4999 ,jumps back to 0 ,then scans up again to serve what remains

Total distance travelled=

(913-143)+(948-913)+(1022-948)+(1470-1022)+(1509-1470)+(1750-1509)+(1774-1750)+(4999-0)+(130-86)= 9769

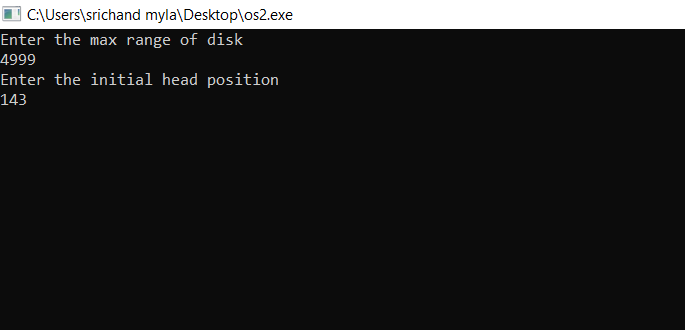
**Output:**

**Step1:Enter the Max range of Disk = 4999**

****

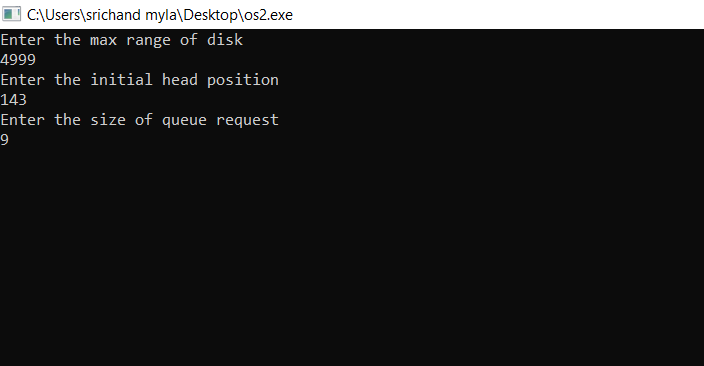
**Step2:**

**Enter the initial head position = 143**

****

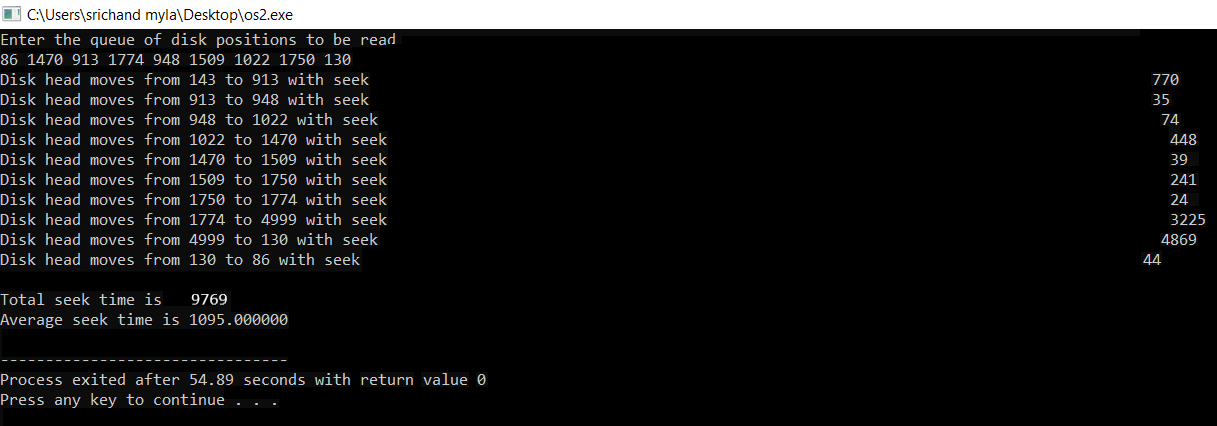
**Step3:**

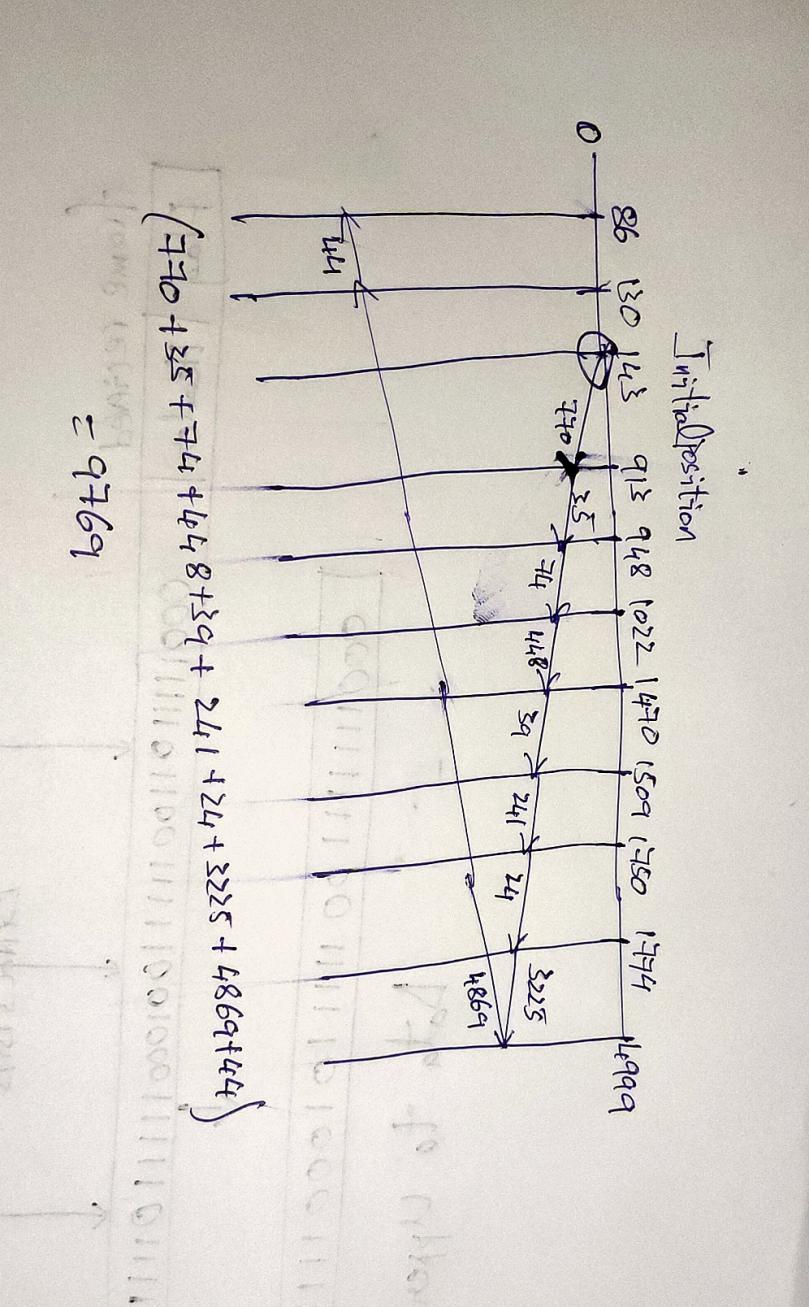
**Enter the size of queue you need = 9**

****

**Step4:**

**Enter the queue of disk positions to be read = 86 1470 913 1774 948 1509 1022 1750 130 and press enter you will get the Total seek time and average seek time.**





7.

**Test Cases:**

|  |  |  |
| --- | --- | --- |
| **Initial state** | **Input** | **Output** |
| Enter max range of disk | 4999 | Asks to enter the initial head position |
| Initial head position | 143 | Asks to enter length of queue you need |
| Queue length | 9 | Prints enter the queue of disk positions |
| Enter queue of disk positions | 86 1470 913 1774 948 1509 1022 1750 130 | Prints the path it moves and prints Total seek time and average seek time |

**8.**

I have made 4 revisions of solutions on git hub