S.No	Title	Date Allotted	Date Checked	Sign/ Remarks
17	Disk Scheduling (FCFS)	11/04/23		
18	Disk Scheduling (Shortest Seek Time)	11/04/23		
19	Disk Scheduling (Scan)	11/04/23		
20	Disk Scheduling (Look /Peek)	11/04/23		
21	Disk Scheduling (C-Scan)	11/04/23		
22	Disk Scheduling (C-Look)	11/04/23		
23	Open Ended Experiment	11/04/23		

Disk Scheduling (FCFS)

```
#include <stdio.h>

int main() {
    int n;
    int diskQueue[100];
    int head, seekTime = 0;
    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    printf("Enter the disk request queue: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &diskQueue[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &head);

    printf("\nFCFS Disk Scheduling:\n");
    printf("Head Movement\tSeek Time\n");
    printf("---------------------------\n");

    for (int i = 0; i < n; i++) {
        printf("%d -> %d\t\t%d\n", head, diskQueue[i], abs(head - diskQueue[i]));
        seekTime += abs(head - diskQueue[i]);
        head = diskQueue[i];
    }

    printf("---------------------------\n");
    printf("Total Seek Time: %d\n", seekTime);

    return 0;
}
```

Output:

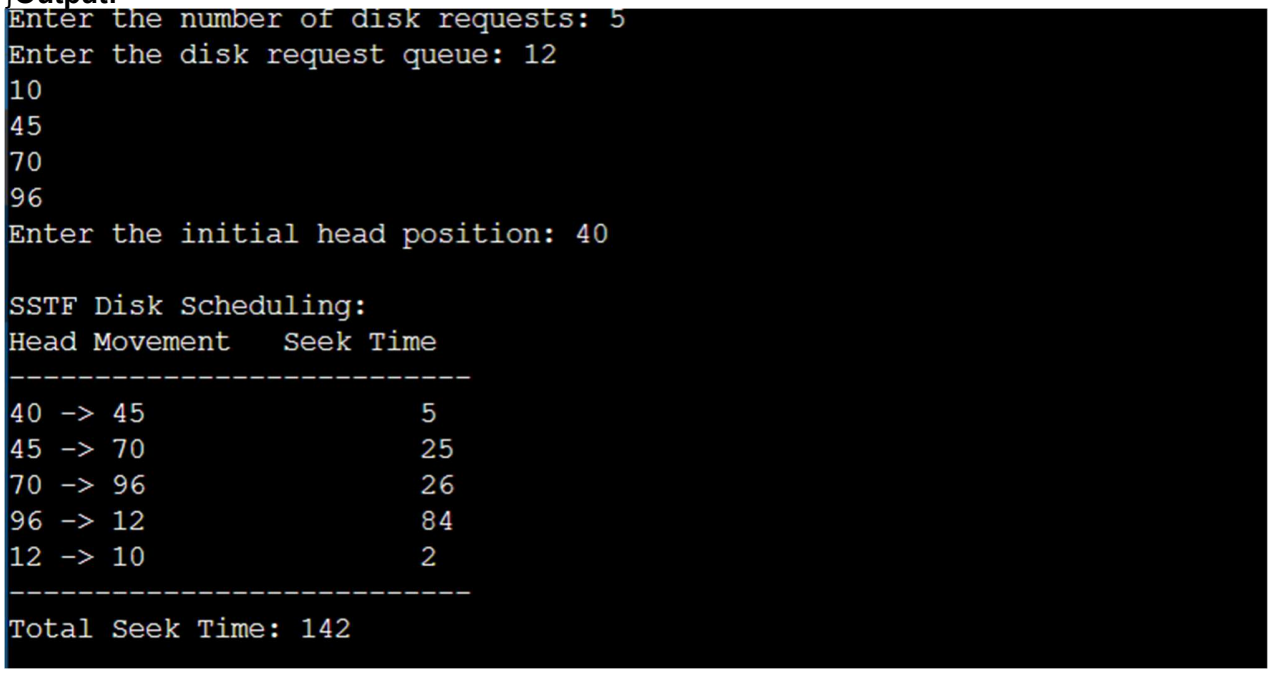
```
Enter the number of disk requests: 5
Enter the disk request queue: 12
72
50
41
89
Enter the initial head position: 40

FCFS Disk Scheduling:
Head Movement   Seek Time
---------------------------
40 -> 12                28
12 -> 72                60
72 -> 50                22
50 -> 41                9
41 -> 89                48
---------------------------
Total Seek Time: 167
```

Disk Scheduling (Shortest Seek Time)

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

int abs(int a) {
    return (a < 0) ? -a : a;
}

int findClosest(int diskQueue[], bool visited[], int head, int n) {
    int minDist = INT_MAX;
    int closest = -1;

    for (int i = 0; i < n; i++) {
        if (!visited[i] && abs(diskQueue[i] - head) < minDist) {
            minDist = abs(diskQueue[i] - head);
            closest = i;
        }
    }

    return closest;
}

int main() {
    int n; // number of disk requests
    int diskQueue[100]; // array to store disk requests
    int head, seekTime = 0; // head position and seek time
    bool visited[100] = { false }; // array to keep track of visited disk requests

    printf("Enter the number of disk requests: ");
    scanf("%d", &n);

    printf("Enter the disk request queue: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &diskQueue[i]);
    }

    printf("Enter the initial head position: ");
    scanf("%d", &head);

    printf("\nSSTF Disk Scheduling:\n");
    printf("Head Movement\tSeek Time\n");
    printf("----------------------------\n");

    int totalRequests = 0;
    while (totalRequests < n) {
        int closest = findClosest(diskQueue, visited, head, n);

        if (closest == -1) {
            break;
        }

        printf("%d -> %d\t\t%d\n", head, diskQueue[closest], abs(head - diskQueue[closest]));
        seekTime += abs(head - diskQueue[closest]);
```

```
        head = diskQueue[closest];
        visited[closest] = true;
        totalRequests++;
    }

    printf("---------------------------\n");
    printf("Total Seek Time: %d\n", seekTime);

    return 0;
}
```

Output:

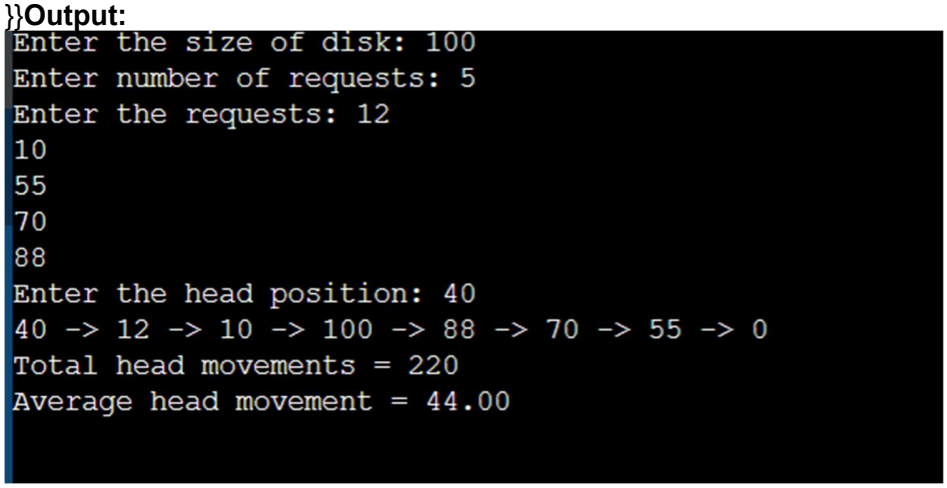


Disk Scheduling (Scan)

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

// Comparison function for qsort()
int cmpfunc(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

int main() {
    int i, j, k, n, m, sum = 0, x, y, h;
    printf("Enter the size of disk: ");
    scanf("%d", &m);
    printf("Enter number of requests: ");
    scanf("%d", &n);
    printf("Enter the requests: ");
    int a[n], b[3 * n];
    for (i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    for (i = 0; i < n; i++) {
        if (a[i] > m) {
            printf("Error, Unknown position %d\n", a[i]);
            return 0;
        }
    }
    printf("Enter the head position: ");
    scanf("%d", &h);
    int temp = h;
    a[n] = h;
    a[n + 1] = m;
    a[n + 2] = 0;
    qsort(a, n + 3, sizeof(int), cmpfunc);
    for (i = 0; i < n + 3; i++) {
        if (h == a[i])
            break;
    }
    k = i;
    if (k < n / 2) {
        for (i = k; i < n + 3; i++) {
            b[i - k] = a[i];
        }
        for (i = k - 1; i >= 0; i--) {
            b[n + 2 - k + i] = a[i];
        }
    } else {
        for (i = k; i >= 0; i--) {
            b[k - i] = a[i];
        }
        for (i = k + 1; i < n + 3; i++) {
            b[n + 2 - i + k] = a[i];
        }
    }
    temp = b[0];
```

```
    printf("%d", temp);
    for (i = 1; i < n + 3; i++) {
        printf(" -> %d", b[i]);
        sum += abs(b[i] - temp);
        temp = b[i];
    }
    printf("\n");
    printf("Total head movements = %d\n", sum);
    printf("Average head movement = %.2f\n", (float)sum / n);
    return 0;
}
```

Output:



Disk Scheduling (Look /Peek)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests 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 direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for look disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for(j=0;j<n-i-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 movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
```

```
        for(i=index-1;i>=0;i--)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
           TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
           initial=RQ[i];
        }

        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 number of Requests
5
Enter the Requests sequence
10
12
45
70
88
Enter initial head position
40
Enter total disk size
100
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 126
```

Disk Scheduling (C-Scan)

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;
    printf("Enter the number of Requests\n");
    scanf("%d",&n);
    printf("Enter the Requests 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 direction for high 1 and for low 0\n");
    scanf("%d",&move);

    // logic for C-Scan disk scheduling

        /*logic for sort the request array */
    for(i=0;i<n;i++)
    {
        for( j=0;j<n-i-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 movement is towards high value
    if(move==1)
    {
        for(i=index;i<n;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];
        }
```

```
        //  last movement for max size
        TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);
        /*movement max to min disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial=0;
        for( i=0;i<index;i++)
        {
            TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
            initial=RQ[i];

        }
    }
    // if movement is towards low value
    else
    {
        for(i=index-1;i>=0;i--)
        {
           TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);
           initial=RQ[i];
        }
        //  last movement for min size
        TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);
        /*movement min to max disk */
        TotalHeadMoment=TotalHeadMoment+abs(size-1-0);
        initial =size-1;
        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 number of Requests
5
Enter the Requests sequence
12
10
45
70
88
Enter initial head position
40
Enter total disk size
100
Enter the head movement direction for high 1 and for low 0
1
Total head movement is 170
```

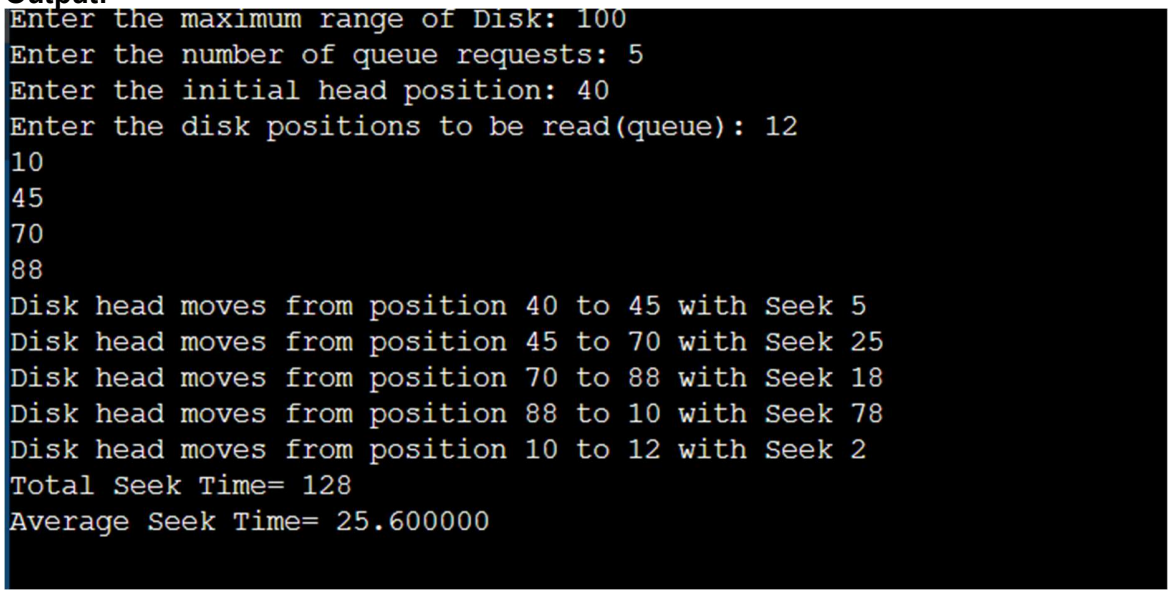
Disk Scheduling (C-Look)

```
#include<stdio.h>
int absoluteValue(int); // Declaring function absoluteValue

void main()
{
   int queue[25],n,headposition,i,j,k,seek=0, maxrange,
   difference,temp,queue1[20],queue2[20],temp1=0,temp2=0;
   float averageSeekTime;

   // Reading the maximum Range of the Disk.
   printf("Enter the maximum range of Disk: ");
   scanf("%d",&maxrange);

   // Reading the number of Queue Requests(Disk access requests)
   printf("Enter the number of queue requests: ");
   scanf("%d",&n);

   // Reading the initial head position.(ie. the starting point of execution)
   printf("Enter the initial head position: ");
   scanf("%d",&headposition);

   // Reading disk positions to be read in the order of arrival
   printf("Enter the disk positions to be read(queue): ");
   for(i=1;i<=n;i++)   // Note that i varies from 1 to n instead of 0 to n-1
   {
      scanf("%d",&temp);  //Reading position value to a temporary variable

      //Now if the requested position is greater than current headposition,
      //then pushing that to array queue1
      if(temp>headposition)
      {
         queue1[temp1]=temp; //temp1 is the index variable of queue1[]
         temp1++; //incrementing temp1
      }
      else    //else if temp < current headposition,then push to array queue2[]
      {
         queue2[temp2]=temp; //temp2 is the index variable of queue2[]
         temp2++;
      }
   }

   //Now we have to sort the two arrays
   //SORTING array queue1[] in ascending order
   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;
         }
```

```
    }
}

//SORTING array queue2[] in ascending order
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;
        }
    }
}

//Copying first array queue1[] into queue[]
for(i=1,j=0;j<temp1;i++,j++)
{
    queue[i]=queue1[j];
}

//Moving Disk head to the inner most requested cylinder,
//because this is Circular LOOK.
queue[i]=queue2[0];

//Copying second array queue2[] after that first one is copied, into queue[]
for(i=temp1+1,j=0;j<temp2;i++,j++)
{
    queue[i]=queue2[j];
}

//At this point, we have the queue[] with the requests in the
//correct order of execution as per C-LOOK algorithm.
//Now we have to set 0th index of queue[] to be the initial headposition.
queue[0]=headposition;

// Calculating SEEK TIME. seek is initially set to 0 in the declaration part.

for(j=0; j<n; j++) //Loop starts from headposition. (ie. 0th index of queue)
{
    // Finding the difference between next position and current position.
    difference = absoluteValue(queue[j+1]-queue[j]);

    // Adding difference to the current seek time value
    seek = seek + difference;

    // Displaying a message to show the movement of disk head
    printf("Disk head moves from position %d to %d with Seek %d \n",
    queue[j], queue[j+1], difference);
}

// Calculating Average Seek time
```

```
    averageSeekTime = seek/(float)n;

    //Display Total and Average Seek Time(s)
    printf("Total Seek Time= %d\n", seek);
    printf("Average Seek Time= %f\n", averageSeekTime);
}

// Defining function absoluteValue
int absoluteValue(int x)
{
    if(x>0)
    {
        return x;
    }
    else
    {
        return x*-1;
    }
}
```

Output:



Open Ended Experiment

a) Repository/Directory Synchronizer: Client-server application which is capable to synchronize the local changes to a remote folder.

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

// The main function, which takes two command-line arguments, the source directory and the destination
directory
int main(int argc, char *argv[])
{
// Check if the user has provided both the source and the destination directories
if (argc < 3) {
printf("Usage: %s source_dir dest_dir\n", argv[0]);
return 1;
}

// Set the path to the rsync executable
char *rsync = "C:\\ProgramData\\chocolatey\\bin\\rsync.exe";
// Set the source and destination directories from command-line arguments
char *source_dir = argv[1];
char *dest_dir = argv[2];

// Define the arguments to be passed to the rsync command
char *args[7] = {
    rsync, // path to rsync executable
    "-avz", // rsync command line option
    "--delete", // rsync command line option
    source_dir, // source directory to sync
    dest_dir, // destination directory to sync
    "--log-file=sync.log", // rsync command line option to specify log file
    NULL // end of argument list
};

// Create process structures to store process information
STARTUPINFO si = { sizeof(si) };
PROCESS_INFORMATION pi;

// Create a command line string from the arguments array
char cmd_line[1024];
int i;
size_t offset = 0;
for (i = 0; args[i] != NULL; i++) {
    offset += snprintf(cmd_line + offset, 1024 - offset, "%s ", args[i]);
}

// Print the command to be run
printf("Running command: %s\n", cmd_line);

// Create the child process to run the rsync command
if (!CreateProcess(NULL, cmd_line, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi)) {
    // If the process creation fails, print the error message and exit the program
    fprintf(stderr, "CreateProcess failed (%d).\n", GetLastError());
    return 1;
```

```
}

// Wait for the child process to exit
WaitForSingleObject(pi.hProcess, INFINITE);

// Get the exit code of the child process
DWORD exit_code;
GetExitCodeProcess(pi.hProcess, &exit_code);

// Close process and thread handles to avoid resource leaks
CloseHandle(pi.hProcess);
CloseHandle(pi.hThread);

// Print the exit code of the child process
printf("Command exited with code %d.\n", exit_code);

// Return 0 to indicate success
return 0;
}
```

Output:

```
[Running] cd "c:\Users\dmana\My_work\LeetCode\Python\Java\Main\" && gcc tempCodeRunnerFile.c -o tempCodeRunnerFile &&
"c:\Users\dmana\My_work\LeetCode\Python\Java\Main\"tempCodeRunnerFile
Usage: c:\Users\dmana\My_work\LeetCode\Python\Java\Main"tempCodeRunnerFile source_dir dest_dir

[Done] exited with code=1 in 2.138 seconds
```

b) WAP for your own Signal Handler which will execute when you type CTRL+D

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

// Signal handler function
void ctrlDHandler(int signum) {
    printf("Received CTRL+D signal (EOF)\n");
    // Additional actions to be taken on receiving CTRL+D signal
    // ...
}

int main() {
    // Set up signal handler for CTRL+D
    if (signal(SIGINT, ctrlDHandler) == SIG_ERR) {
        printf("Failed to set up signal handler for CTRL+D\n");
        return 1;
    }

    // Loop indefinitely until CTRL+D is received
    while (1) {
        printf("Enter input (or press CTRL+D to exit): ");
        int ch = getchar();
        if (ch == EOF) {
            break; // Exit loop on receiving EOF
        }
        // Additional actions to be taken with input
        // ...
        getchar(); // Consume newline character
    }

    printf("Exiting...\n");
    return 0;
}
```

Output:

```
Enter input (or press CTRL+D to exit): 20
Enter input (or press CTRL+D to exit): 12
Enter input (or press CTRL+D to exit): Enter input (or press CTRL+D to exit): 10
Enter input (or press CTRL+D to exit): 24
Enter input (or press CTRL+D to exit): Enter input (or press CTRL+D to exit): 10
Enter input (or press CTRL+D to exit): Enter input (or press CTRL+D to exit): Exiting...
```