Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	[@KLWKS_BOT THANOS]

Experiment Title: DISK SCHEDULING ALGORITHMS

Aim/Objective: Students should be able to understand the concepts of Disk Scheduling. It helps in techniques like coordinating execution of First Come First Serve (FCFS) Disk Scheduling, Shortest Seek Time First (SSTF) Disk Scheduling, SCAN Disk Scheduling, LOOK Disk Scheduling, and C-SCAN Disk Scheduling.

Description:

Disc scheduling is an important process in operating systems that determines the order in which disk access requests are serviced. The objective of disc scheduling is to minimize the time it takes to access data on the disk and to minimize the time it takes to complete a disk access request. Disk access time is determined by two factors: seek time and rotational latency. Seek time is the time it takes for the disk head to move to the desired location on the disk, while rotational latency is the time taken by the disk to rotate the desired data sector under the disk head. Disk scheduling algorithms are an essential component of modern operating systems and are responsible for determining the order in which disk access requests are serviced. The primary goal of these algorithms is to minimize disk access time and improve overall system performance.

Prerequisite:

- **Basic functionality of Disk Scheduling Algorithms.**
- **Complete idea of FCFS, SCAN, and C-SCAN.**

Pre-Lab Task:

Disk Scheduling Parameters	FUNCTIONALITY
Seek time	Time to move the disk arm to the requested track.

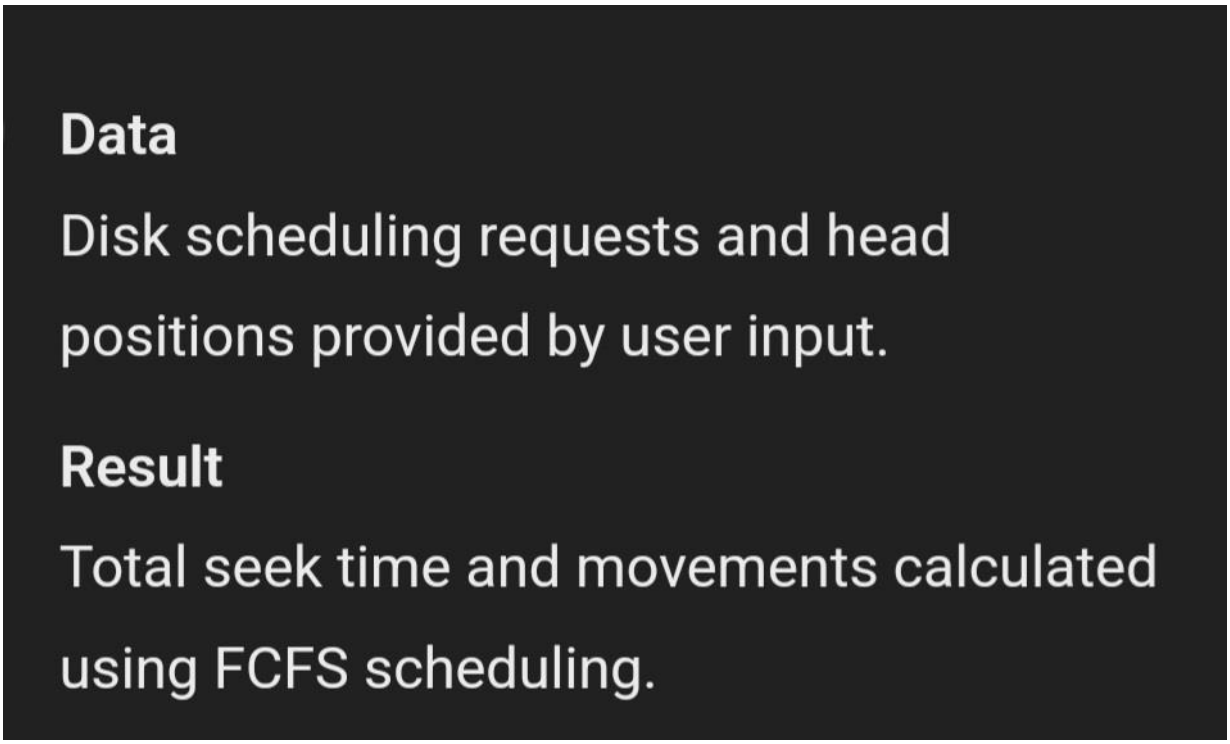
Disk Scheduling Parameters	FUNCTIONALITY
Transfer time	Time to transfer data from disk to memory.
Disk Access time	Time taken to locate and retrieve data from disk.
Rotational Latency	Time for the disk to rotate the desired sector under head.

In Lab Task:

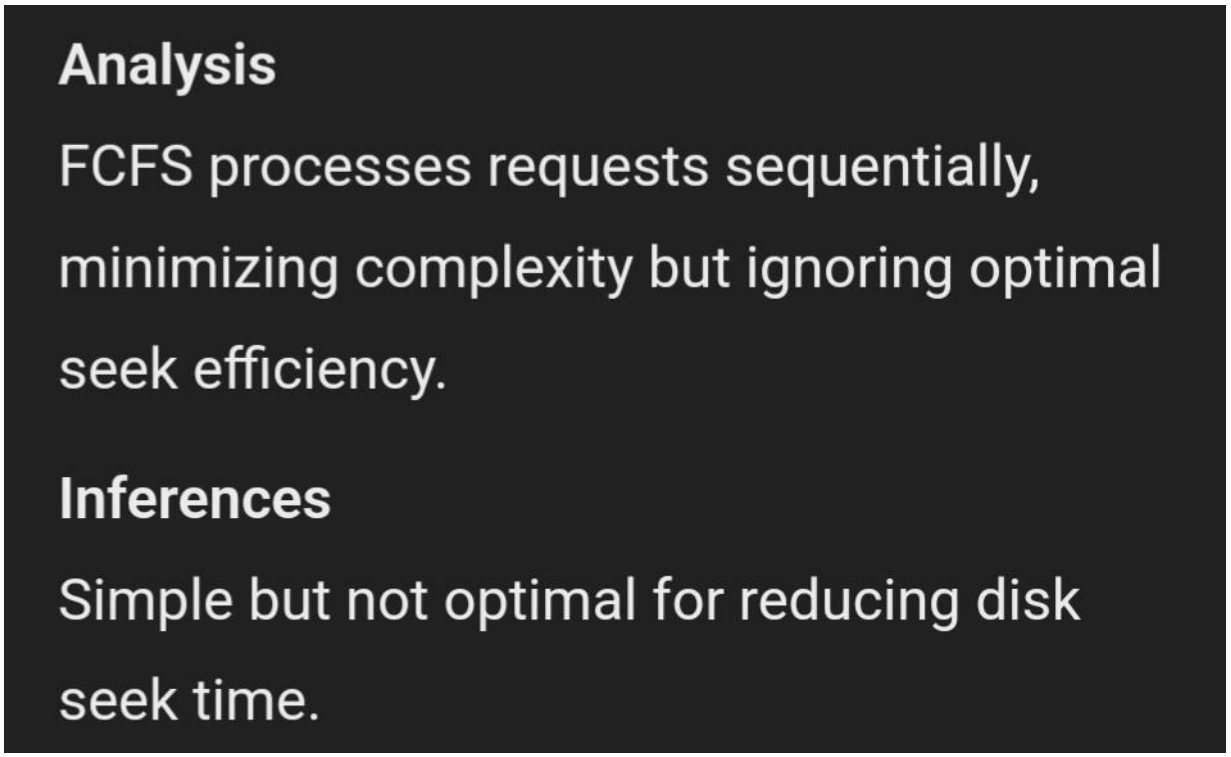
1. Write a C program to implement the FCFS Disk Scheduling Algorithm.

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

int main() {
   int n, i;

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

   int requests[n];

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

   int current_head, total_seek_time;

   printf("Enter the current position of the disk head: ");
   scanf("%d", &current_head);

   total_seek_time = 0;

   for (i = 0; i < n; i++) {
      int seek_distance = abs(current_head - requests[i]);
      total_seek_time += seek_distance;
      printf("Move from %d to %d (seek time: %d)\n", current_head, requests[i], seek_distance);
      current_head = requests[i];
   }

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

   return 0;
}
```

Data and Results



Analysis and Inferences:

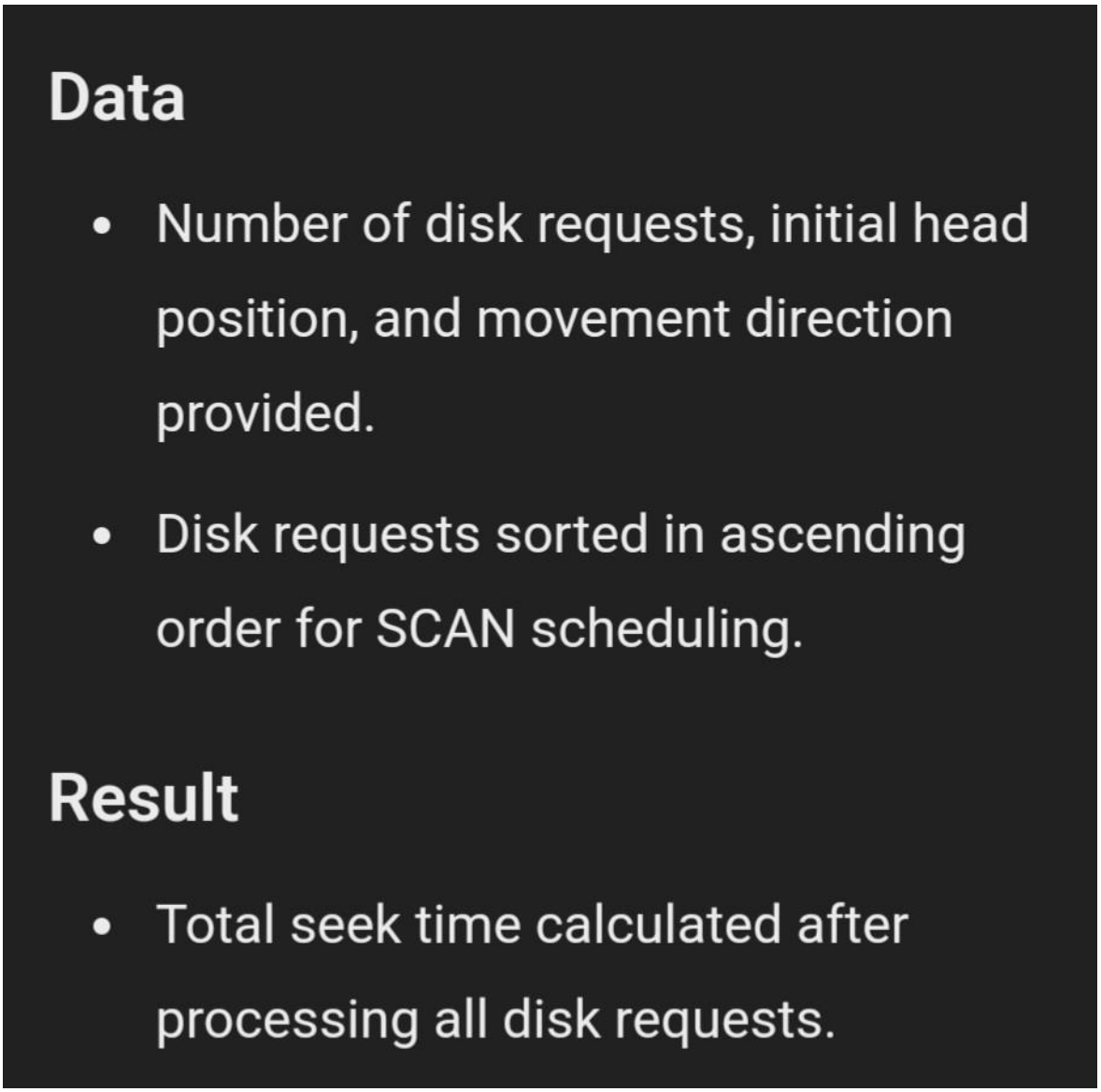


2. Write a C program to implement the SCAN Disk scheduling algorithm.

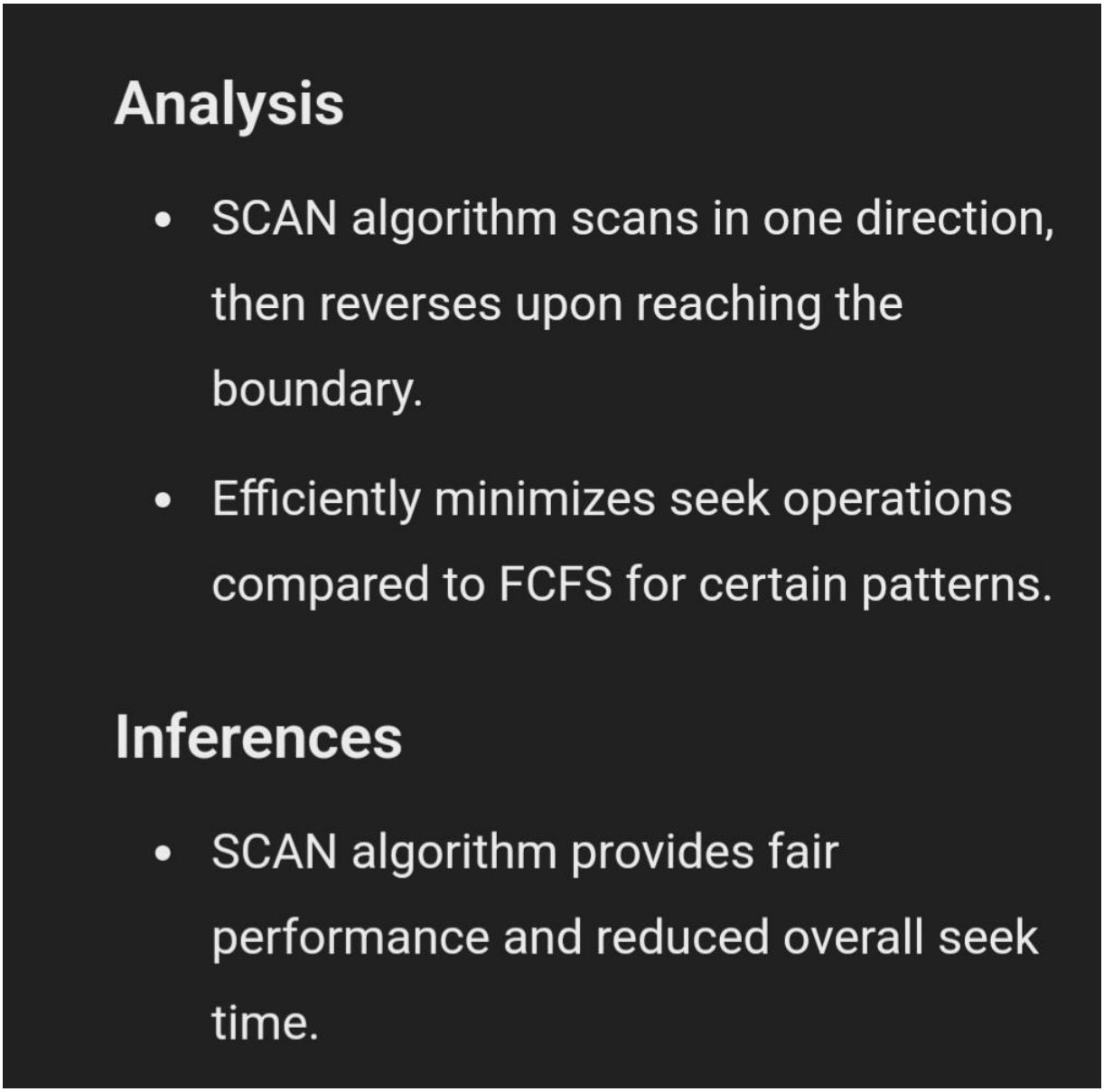
```
#include <stdio.h>
#include <stdlib.h>

void sort(int arr[], int n) {
   int i, j, temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

int main() {
   int n, i, current_head;
   printf("Enter the number of disk requests: ");
   scanf("%d", &n);

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

   printf("Enter the current position of the disk head: ");
   scanf("%d", &current_head);

   int total_seek_time = 0;
   sort(requests, n);

   int direction;
   printf("Enter the direction (0 for left, 1 for right): ");
   scanf("%d", &direction);

   if (direction == 0) {
```

```
  for (i = current_head; i >= 0; i--) {
        printf("Move from %d to %d (seek time: %d)\n", current_head, i, abs(current_head - i));
        total_seek_time += abs(current_head - i);
        current_head = i;
      }
      for (i = 0; i < n; i++) {
        printf("Move from %d to %d (seek time: %d)\n", current_head, requests[i], abs(current_head -
requests[i]));
        total_seek_time += abs(current_head - requests[i]);
        current_head = requests[i];
      }
    } else {
      for (i = current_head; i < 200; i++) {
        printf("Move from %d to %d (seek time: %d)\n", current_head, i, abs(current_head - i));
        total_seek_time += abs(current_head - i);
        current_head = i;
      }
      for (i = n - 1; i >= 0; i--) {
        printf("Move from %d to %d (seek time: %d)\n", current_head, requests[i], abs(current_head -
requests[i]));
        total_seek_time += abs(current_head - requests[i]);
        current_head = requests[i];
      }
    }

    printf("Total seek time: %d\n", total_seek_time);
    return 0;
}
```

Data and Results



Analysis and Inferences:

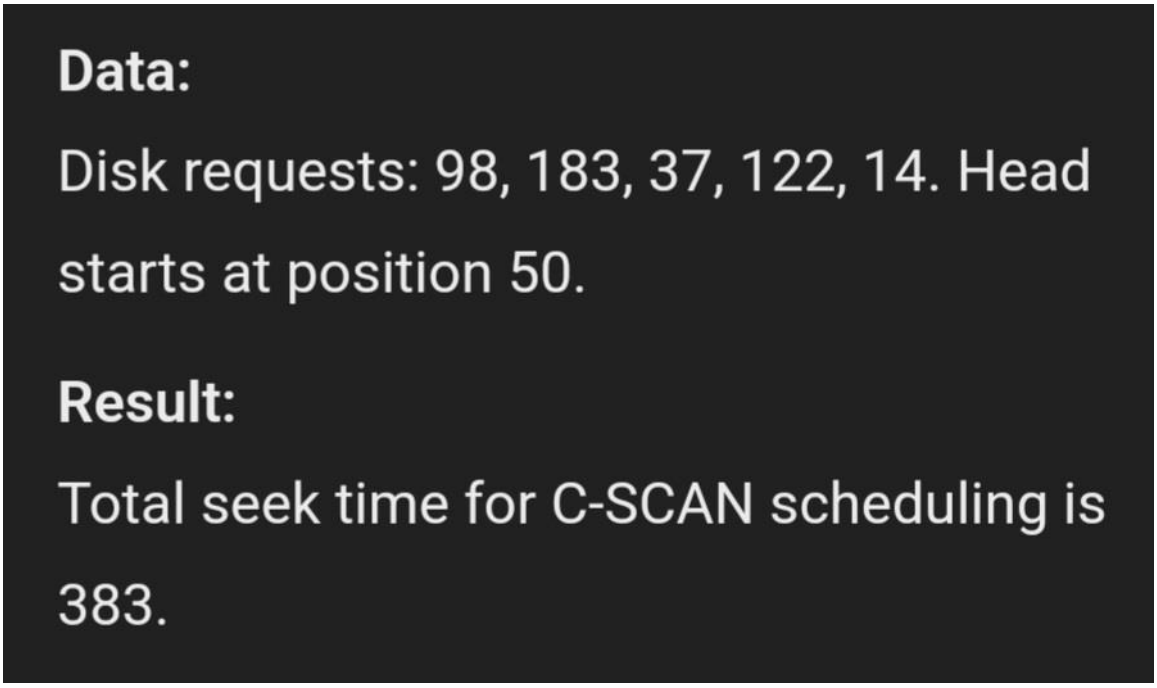


3. Write a C program to implement the C-SCAN Disk scheduling algorithm.

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

void sort(int arr[], int n) {
   int i, j, temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

int main() {
   int n, i, current_head;

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

   int requests[n];

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

   printf("Enter the current position of the disk head: ");
   scanf("%d", &current_head);

   int total_seek_time = 0;

   sort(requests, n);

   int current_index = -1;
```

```
  for (i = 0; i < n; i++) {
      if (requests[i] >= current_head) {
         current_index = i;
         break;
      }
   }

   for (i = current_index; i < n; i++) {
      printf("Move from %d to %d (seek time: %d)\n", current_head, requests[i], abs(current_head -
requests[i]));
      total_seek_time += abs(current_head - requests[i]);
      current_head = requests[i];
   }

   printf("Move from %d to 0 (seek time: %d)\n", current_head, current_head);
   total_seek_time += current_head;
   current_head = 0;

   for (i = 0; i < current_index; i++) {
      printf("Move from %d to %d (seek time: %d)\n", current_head, requests[i], abs(current_head -
requests[i]));
      total_seek_time += abs(current_head - requests[i]);
      current_head = requests[i];
   }

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

   return 0;
}
```

Data and Results



Analysis and Inferences

Analysis:

C-SCAN serves requests in one direction, reducing wait variability and ensuring cyclic scanning.

Inferences:

C-SCAN provides fair scheduling but increases seek time compared to other algorithms.

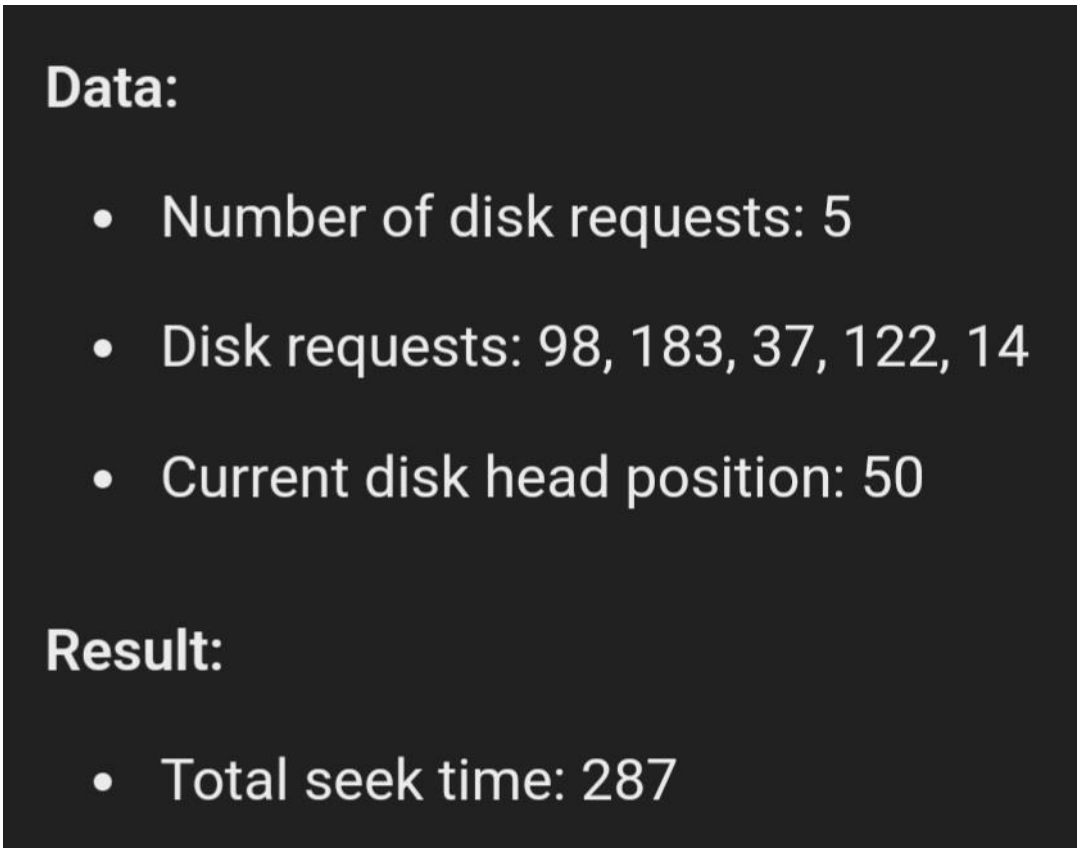
Post Lab:

1. Write a Program C-LOOK Disk Scheduling Algorithm in C

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

void sort(int arr[], int n) {
   int i, j, temp;
   for (i = 0; i < n - 1; i++) {
      for (j = 0; j < n - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

int main() {
   int n, i, current_head;
   printf("Enter the number of disk requests: ");
   scanf("%d", &n);

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

   printf("Enter the current position of the disk head: ");
   scanf("%d", &current_head);

   int total_seek_time = 0;
   sort(requests, n);

   int current_index = -1;
   for (i = 0; i < n; i++) {
      if (requests[i] >= current_head) {
         current_index = i;
```

```
  break;
      }
   }

   for (i = current_index; i < n; i++) {
      printf("Move from %d to %d (seek time: %d)\n", current_head, requests[i], abs(current_head -
requests[i]));
      total_seek_time += abs(current_head - requests[i]);
      current_head = requests[i];
   }

   for (i = 0; i < current_index; i++) {
      printf("Move from %d to %d (seek time: %d)\n", current_head, requests[i], abs(current_head -
requests[i]));
      total_seek_time += abs(current_head - requests[i]);
      current_head = requests[i];
   }

   printf("Total seek time: %d\n", total_seek_time);
   return 0;
}
```

Data and Results



1. Write a C Program to implement the SSTF Disk scheduling algorithm.

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

int findNearestRequest(int requests[], int n, int current_head, int visited[]) {
   int min_distance = INT_MAX;
   int nearest_index = -1;
   for (int i = 0; i < n; i++) {
      if (!visited[i]) {
         int distance = abs(requests[i] - current_head);
         if (distance < min_distance) {
            min_distance = distance;
            nearest_index = i;
         }
      }
   }
   return nearest_index;
}

int main() {
   int n, i, current_head;
   printf("Enter the number of disk requests: ");
   scanf("%d", &n);
   int requests[n];
   printf("Enter the disk requests:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &requests[i]);
   }
   printf("Enter the current position of the disk head: ");
   scanf("%d", &current_head);

   int total_seek_time = 0;
   int visited[n];
   for (i = 0; i < n; i++) {
      visited[i] = 0;
   }
```

```
 for (i = 0; i < n; i++) {
      int nearest_index = findNearestRequest(requests, n, current_head, visited);
      if (nearest_index != -1) {
         int distance = abs(requests[nearest_index] - current_head);
         total_seek_time += distance;
         visited[nearest_index] = 1;
         current_head = requests[nearest_index];
         printf("Move from %d to %d (seek time: %d)\n", current_head - distance, current_head,
distance);
      }
   }
   printf("Total seek time: %d\n", total_seek_time);
   return 0;
}
```

2. Write an algorithm to implement the SCAN Disk scheduling algorithm.

1. **Sort** the 'requests' array.

2. **Initialize** 'total_seek_time' and 'seek_sequence'.

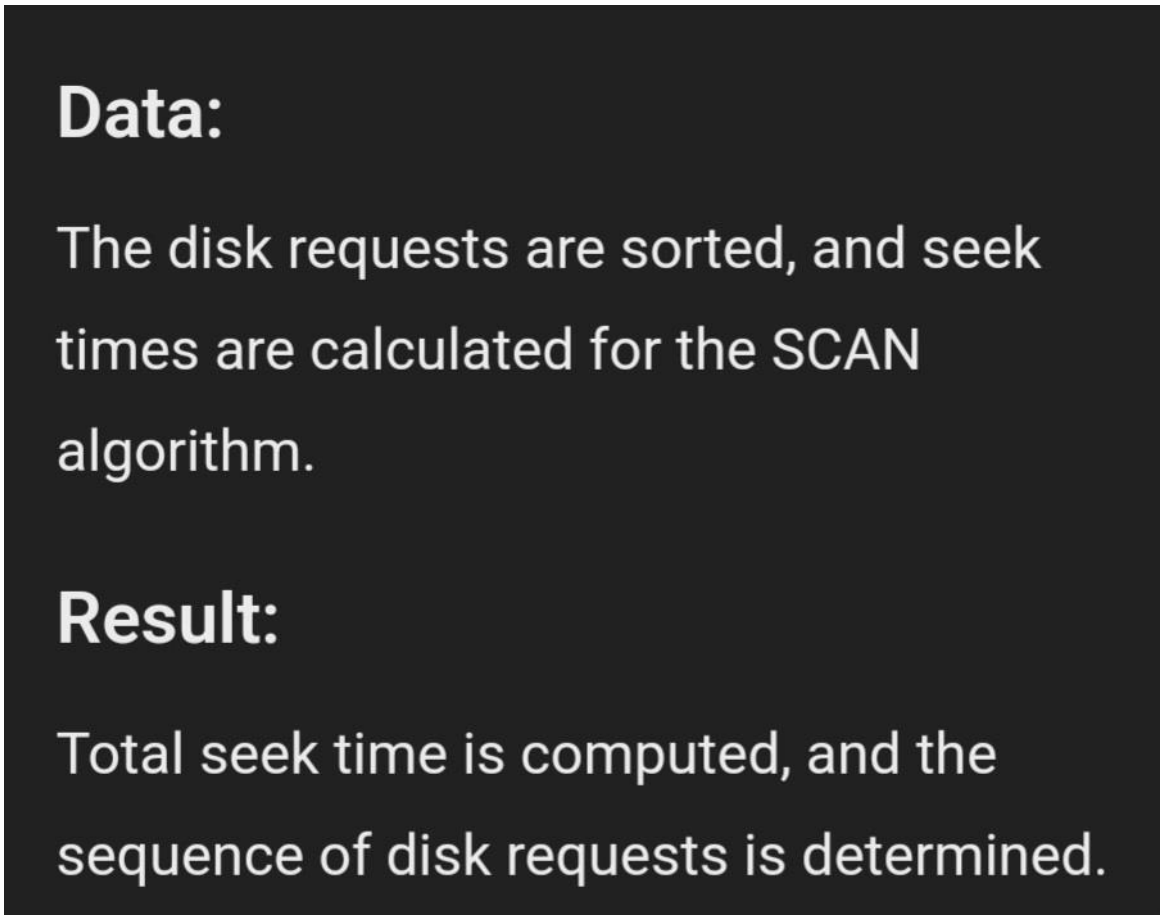
3. If **direction** is 0 (left):
    - **Iterate** from 'current_head' to 0, service requests.
    - **Reverse** direction.

4. If **direction** is 1 (right):
    - **Iterate** from 'current_head' to max, service requests.
    - **Reverse** direction.

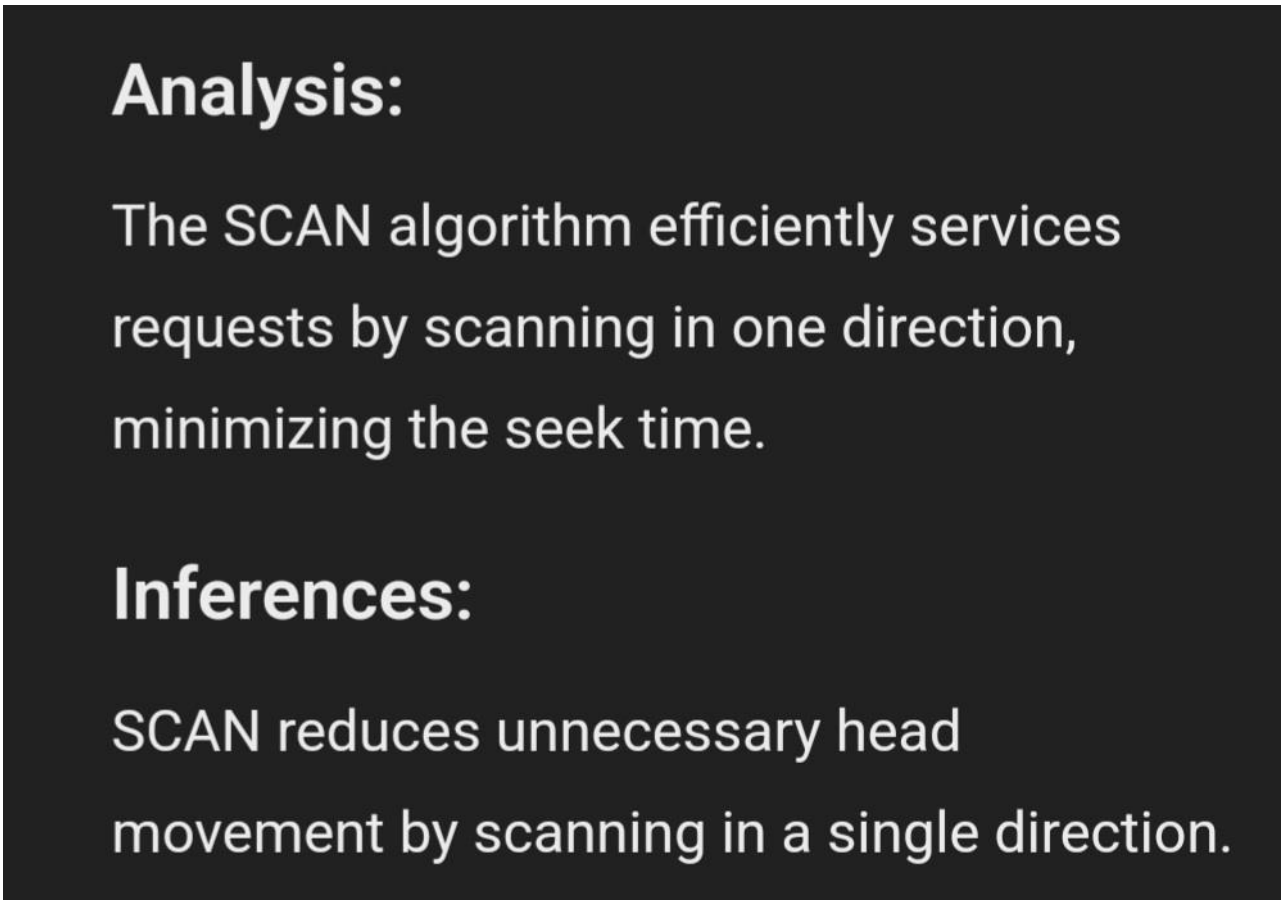
5. **Sum** seek times for total.

6. **Return** 'seek_sequence' and 'total_seek_time'.

Data and Results

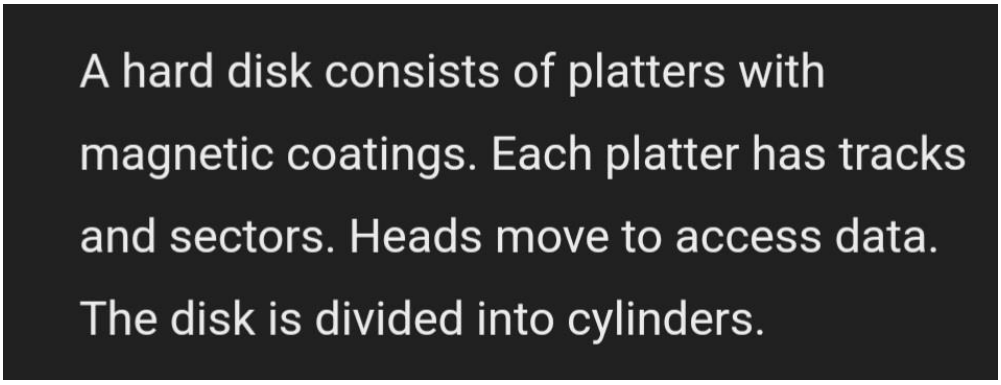


Analysis and Inferences

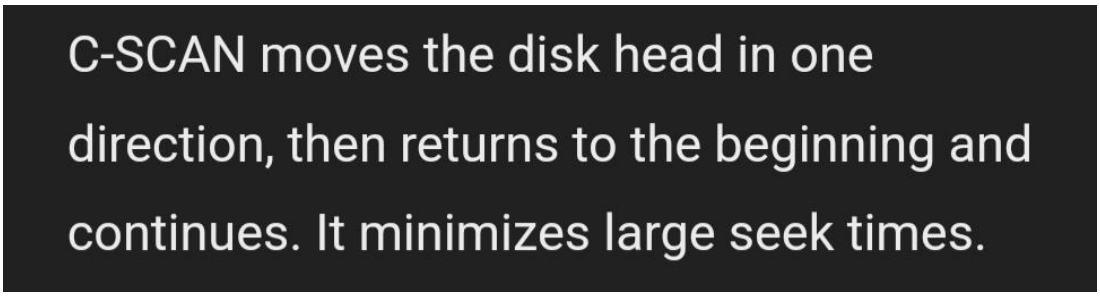


Sample VIVA-VOCE Questions (In-Lab):

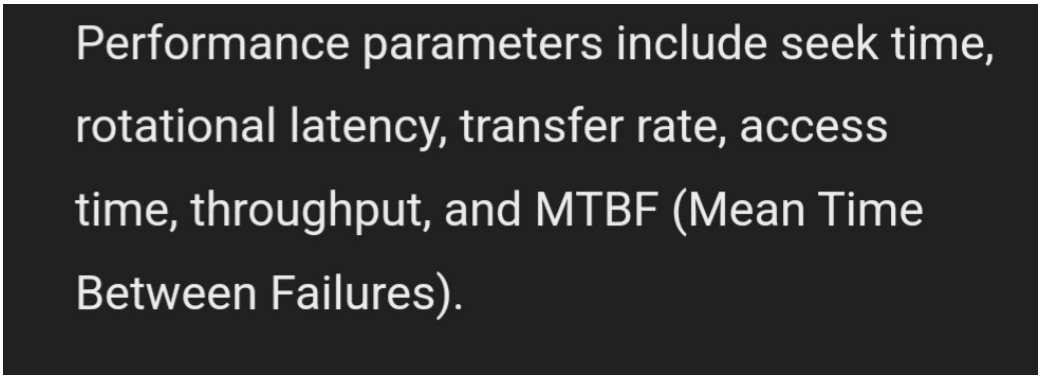
1. Explain in detail the Hard Disk Structure.



2. Explain in detail about C-SCAN.



3. Explain in detail Hard disk performance parameters and terminologies.



4. Explain in detail the Advantages and Disadvantages of FCFS disk scheduling.

Advantages: Simple, fair.

Disadvantages: High seek time, inefficient with scattered requests.

5. Explain in detail RAID (Redundant Array of Independent Disks)

RAID combines multiple disks for redundancy or performance. Common levels: 0 (striping), 1 (mirroring), 5 (parity), 10 (combination).

Evaluator Remark (if any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Note: Evaluator MUST ask Viva-voce before signing and posting marks for each experiment.