Question: Which disk algorithm is the most efficient?

Introduction:

Operating systems employ disk scheduling to manage the arrival of input/output requests to the disk. I/O scheduling is another name for disk scheduling. The relevance of disk scheduling can be summed up as follows:

- Hard drives are one of the slowest components in a computing device, therefore they must be accessible quickly.
- Multiple I/O requests may arrive from distinct processes, but the disk controller can only service one I/O request at a time. As a result, further I/O requests must wait in the queue and be scheduled.
- Two or more requests may be separated by a significant distance, resulting in increased disk arm movement.

Experiment :

We compare six disk algorithms i.e. First Come First Serve (FCFS), Shortest Seek Time First (SSTF), SCAN, Circular SCAN (C-SCAN), LOOK, Circular LOOK (C-LOOK) in terms of total number of head movements and total time taken.

Implementation details:

The program runs on a 10,000-cylinder disk with numbers ranging from 0 to 9999. It produces a random request array of 5000 requests and processes them using each of the above techniques. The disk head's beginning location (start index) is given to the program as a command line argument, and it calculates the total number of head movements and the time required by each algorithm. I ran the program with several starting indexes, each incrementing by 500, and documented the results.

1. **First Come First Serve (FCFS):** The FCFS disk scheduling algorithm is by far the simplest of all the disk scheduling algorithms. Requests are handled in FCFS in the order they appear in the request queue. In our software, we start from the index following the initial index and add to the head movement in the sequence of requests received, starting with the starting index.

2. **Shortest Seek Time First(SSTF):** In SSTF, requests with the least seek time are processed first. As a result, each request's seek time is computed in advance in the queue, and then they are scheduled based on that determined seek time. As a result, the request that is closest to the disk arm will be processed first. To run SSTF, I identified the shortest distance to the left and right of the starting index, added it to an integer that tracks our total head movement, and then updated the head at whatever request we read from. This is done in a loop that terminates when we've read all of the requests in our array.

3. **SCAN :** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait. In program, head starts from one left of start, and then goes down to zero even if it’s not included in the random requests of array. Then starts at one higher than start and then goes up to highest value.

4. **Circular-SCAN(C-SCAN) :** In C-SCAN algorithm, the disk arm in place of reversing its direction like SCAN goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion. This algorithm works by beginning at the starting index and then reading each request until we reach the very end of our array. We must then travel all the way to the beginning of our array i.e. index 0. From there, we read every single request until we reach the one that is closest to our starting index, which continues until we read all requests in the array.

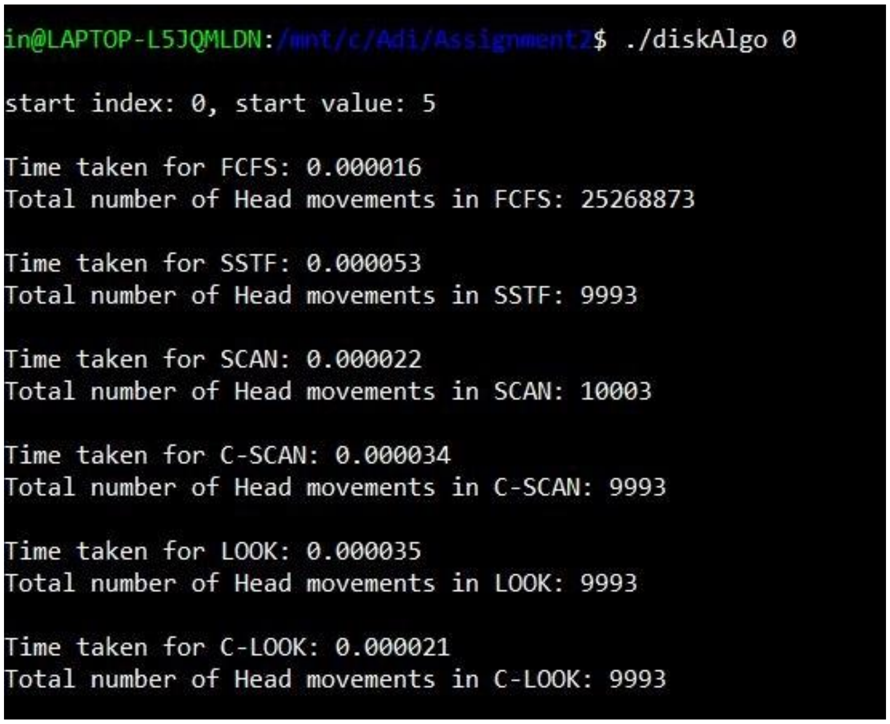
5. **LOOK :** It's similar to the SCAN disk scheduling algorithm, except and instead of continuing to the end of the disk, its disk arm only goes to the last request to be handled in front of the head and then reverses course from there. As a result, the additional latency caused by the

unneeded traverse to the disk's end is avoided. In the program, we begin with a value that is higher than the start value and work our way up to the highest value. Then it decreases until it reaches the least value in the request array.

6. **Circular-LOOK (C-LOOK) :** C-LOOK is comparable to the C-SCAN disk scheduling method in the same way that LOOK is related to the SCAN algorithm. Instead of going to the end, the disk arm in C-LOOK only goes to the last request to be fulfilled in front of the head, and then to the other end's last request. As a result, it avoids the additional time caused by unneeded traversal to the disk's end. C-LOOK is a more advanced version of C-SCAN. We only reach the requests that are closest to the upper and lower borders of our array, rather than reaching them all the time. As a result, we were able to reduce our overall head movement and wait time by a small amount, which is still a significant improvement.

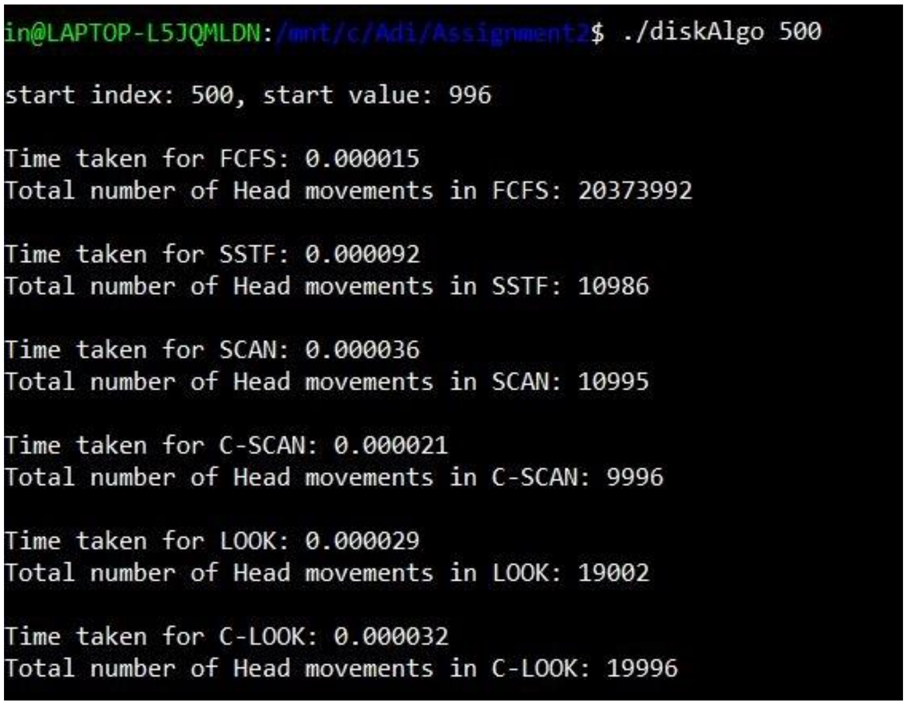
Program Output:

1. Start index = 0



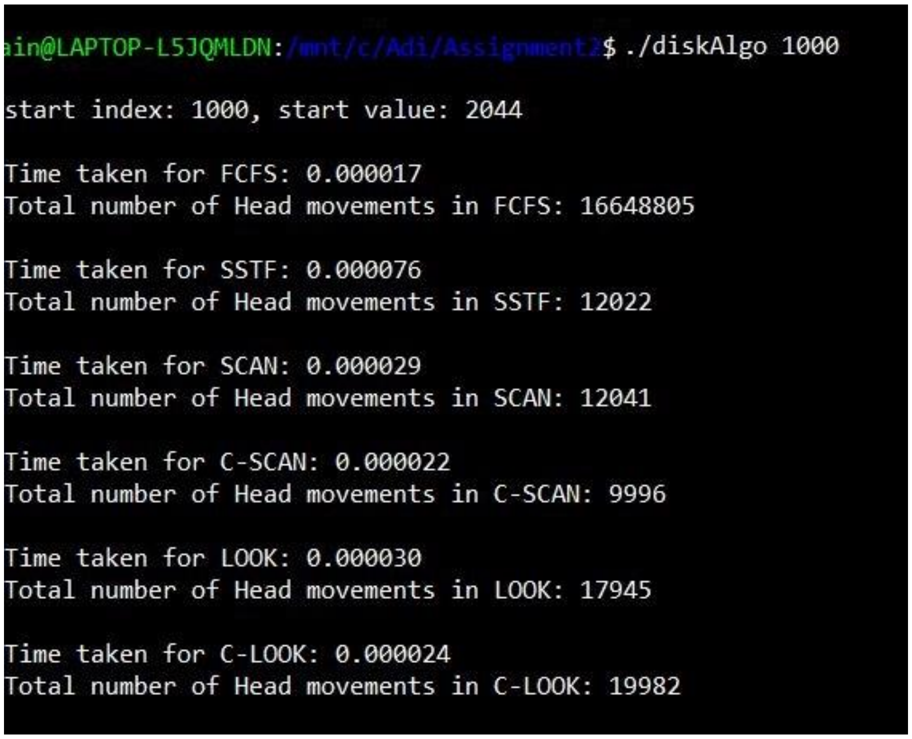
```
in@LAPTOP-L5JQMLDN:/mnt/c/Adi/Assignment2$ ./diskAlgo 0

start index: 0, start value: 5

Time taken for FCFS: 0.000016
Total number of Head movements in FCFS: 25268873

Time taken for SSTF: 0.000053
Total number of Head movements in SSTF: 9993

Time taken for SCAN: 0.000022
Total number of Head movements in SCAN: 10003

Time taken for C-SCAN: 0.000034
Total number of Head movements in C-SCAN: 9993

Time taken for LOOK: 0.000035
Total number of Head movements in LOOK: 9993

Time taken for C-LOOK: 0.000021
Total number of Head movements in C-LOOK: 9993
```

2. Start index = 500



```
in@LAPTOP-L5JQMLDN:/mnt/c/Adi/Assignment2$ ./diskAlgo 500

start index: 500, start value: 996

Time taken for FCFS: 0.000015
Total number of Head movements in FCFS: 20373992

Time taken for SSTF: 0.000092
Total number of Head movements in SSTF: 10986

Time taken for SCAN: 0.000036
Total number of Head movements in SCAN: 10995

Time taken for C-SCAN: 0.000021
Total number of Head movements in C-SCAN: 9996

Time taken for LOOK: 0.000029
Total number of Head movements in LOOK: 19002

Time taken for C-LOOK: 0.000032
Total number of Head movements in C-LOOK: 19996
```

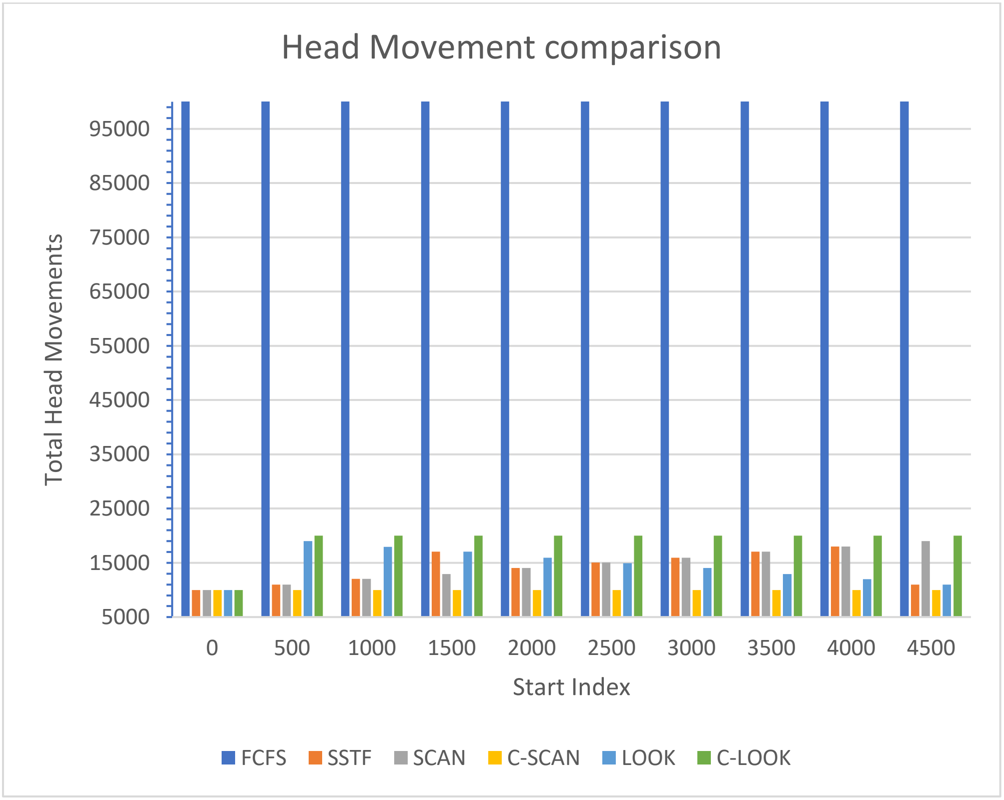
3. Start index = 1000



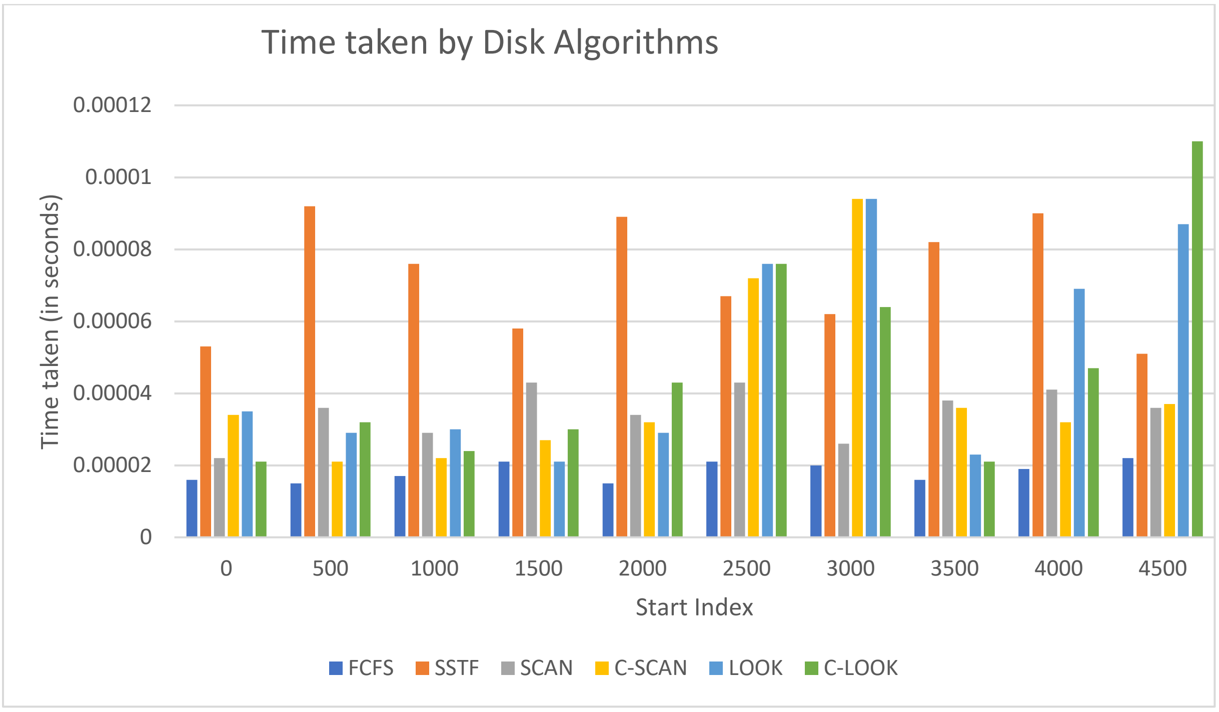
```
ain@LAPTOP-L5JQMLDN:/mnt/c/Adi/Assignment2$ ./diskAlgo 1000

start index: 1000, start value: 2044

Time taken for FCFS: 0.000017
Total number of Head movements in FCFS: 16648805

Time taken for SSTF: 0.000076
Total number of Head movements in SSTF: 12022

Time taken for SCAN: 0.000029
Total number of Head movements in SCAN: 12041

Time taken for C-SCAN: 0.000022
Total number of Head movements in C-SCAN: 9996

Time taken for LOOK: 0.000030
Total number of Head movements in LOOK: 17945

Time taken for C-LOOK: 0.000024
Total number of Head movements in C-LOOK: 19982
```

Similarly running for different start index. I incremented start index by a value of 500 and checked total head movements and time taken by different disk algorithms.

Start Index	Total Head Movements					
	FCFS	SSTF	SCAN	C-SCAN	LOOK	C-LOOK
0	25268873	9993	10003	9993	9993	9993
500	20373992	10986	10995	9996	19002	19996
1000	16648805	12022	12041	9996	17945	19982
1500	14676129	17075	12910	9997	17075	19989
2000	13139102	14061	14062	9998	15935	19998
2500	12488382	15052	15058	9997	14938	19995
3000	12920710	15951	15962	9998	14030	19992
3500	14372296	17042	17051	9997	12941	19991
4000	16881212	17987	17997	9998	11983	19980
4500	20389054	11002	18993	9998	11002	19994



Start Index	Time Taken in seconds(in seconds)					
	FCFS	SSTF	SCAN	C-SCAN	LOOK	C-LOOK
0	0.000016	0.000053	0.000022	0.000034	0.000035	0.000021
500	0.000015	0.000092	0.000036	0.000021	0.000029	0.000032
1000	0.000017	0.000076	0.000029	0.000022	0.00003	0.000024
1500	0.000021	0.000058	0.000043	0.000027	0.000021	0.00003
2000	0.000015	0.000089	0.000034	0.000032	0.000029	0.000043
2500	0.000021	0.000067	0.000043	0.000072	0.000076	0.000076
3000	0.00002	0.000062	0.000026	0.000094	0.000094	0.000064
3500	0.000016	0.000082	0.000038	0.000036	0.000023	0.000021
4000	0.000019	0.00009	0.000041	0.000032	0.000069	0.000047
4500	0.000022	0.000051	0.000036	0.000037	0.000087	0.00011



Observations from the experiment:

To do the comparisons in the performance of these six algorithms(FCFS, SSTF, SCAN, C-SCAN, LOOK, and C-LOOK), I ran each algorithm against different starting indexes that increment by 500 and compared the time taken and total head movements against each other. Observations made can be list in following points.

- FCFS is the fastest algorithm in terms of time taken to service all requests, but it also results in huge amount of total head movement as compared to other algorithms. To understand if other algorithms are taking between 10,000 to 20,000 head movements to service all requests. FCFS takes in the range of 15 to 25 million head movements, which is more than 100o times.
  It results in less starvation but if we use this algorithm in heavily loaded system, it can corrupt the hardware in some time. So, we should only use this algorithm in lightly loaded systems where quicker servicing of requests is of more importance.
- SSTF is good at minimizing the total head movement for our hard drives. However, it is important to note that this algorithm also takes near or more than double the time than C-SCAN

or C-LOOK so it may starve some requests if we used a bigger array. So we can conclude that SSTF should be used for lightly loaded systems.

- C-SCAN is the best at minimizing the total head movement and also time taken for this algorithm is lesser than SSTF, although more than FCFS.
- SCAN, C-SCAN, LOOK and C-LOOK result overall in best performance with almost similar head movements and time taken. Starvation is very less and so these algorithms can be used for larger arrays or heavily loaded systems.
- C-SCAN and C-LOOK are the optimization of SCAN and LOOK, but it is noticed if the initial head position of disk is placed at middle point or towards the end of the disk, these algorithms result in a greater number of head movements and the total time taken is also more.

References :

1. Peter Bare Silberschatz, Greg Galvin, Gagne
   Operating System Principles, volume 9
2. Andrew S Tanenbaum
   Modern operating system, 2nd edition, 2014
3. https://www.researchgate.net/publication/301890138_A_New_Approach_of_Disk_Scheduling_Algorithm
4. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3349013