**Disk Sorting Algorithms**

The code consists of three main sorting algorithms and one method used to generate 1000 random requests for the cylinders,

6 **static** **void** randPick(**int** arr[], **int** arr1[])

14 **static** **void** FCFS(**int** pos, **int** arr[])

32 **static** **void** SSTF(**int** pos, **int** arr[], **int** arr1[])

58 **static** **void** SCAN(**int** pos, **int** arr[], **int** arr1[])

The method randPick takes in two arrays, one to input the random requests to and one to take in and pick for the random request. This uses the random built in java.util to create a random number from 0 to 4999, representing all available cylinders. This is then run through a for loop from arr[0] to arr[1000] filling it with random values.

First Come First Serve, Shortest Seek Time First, and SCAN all takes in a user input for the current head with **int** pos, an array of random requests **int** arr[], and with the exception of FCFS, also take in the cylinder array for its length with **int** arr1[].

Each algorithm consists of the same general variables, **int** count which is used count the total amount of head movement, **int** head which is used to update the algorithm to a new head location, and **int** distance which stores the distance between the current head and the next head.

For FCFS, the method finds the distance between the current head and the next head 22 distance = Math.*abs*(head - pos), updates the head movement count with the distance 24 count = count + distance, and updates the head to the new position 26 pos = head. This follows the same format for both SSTF and SCAN.

For SSTF, a 33 **boolean**[] check = **new** **boolean**[1000]; is used to check if each index for the request is checked with a loop. It then runs through a second loop to check for the distance and if the index is not true. If this is the case, the head is then updated to the second instance variable in this case j. The math is calculated by finding the current head with the request location. Then it marks the current head as true and follows the same update as FCFS.

For SCAN, there is also 62 String direction = "left", which takes in the direction, in this case it is set to left. It is also followed by three vectors 62 Vector<Integer> left = **new** Vector<Integer>(), right = **new** Vector<Integer>() and Vector<Integer> seekSeq = **new** 64 Vector<Integer>().The first two are used for both directions left and right, while the last is used to check the sequence of the head. The method then checks for edge cases for the last values in both directions, and reverses the direction for both left and right. They are then put through a sort using 83 Collections.*sort*(left) and 84 Collections.*sort*(right). It then goes through a loop for both left and right side, adding the head to the sequence vector and then following the same FCFS update. The string direction is then swapped at the end of the loop.

The main function creates and initializes both array of 124 **int**[] cylinders = **new** **int**[5000] and array of 126 **int**[] cylinders = **new** **int**[5000]. It then allows for user input for the initial head location which is then used and passed into the three algorithm methods right after, printing the head movement count for FCFS, SSTF and SCAN respectively.

The code consists of three main sorting algorithms and one method used to generate 1000 random requests for the cylinders,

6 **static** **void** randPick(**int** arr[], **int** arr1[])

123 **static** **void** CSCAN(**int** pos, **int** arr[], **int** arr1[])

176 **static** **void** LOOK(**int** pos, **int** arr[], **int** arr1[])

234 **static** **void** CLOOK(**int** pos, **int** arr[], **int** arr1[])

The method randPick takes in two arrays, one to input the random requests to and one to take in and pick for the random request. This uses the random built in java.util to create a random number from 0 to 4999, representing all available cylinders. This is then run through a for loop from arr[0] to arr[1000] filling it with random values.

CSCAN, LOOK and CLOOK all takes in a user input for the current head with **int** pos, an array of random requests **int** arr[], and the cylinder array for its length with **int** arr1[].

Each algorithm consists of the same general variables, **int** count which is used count the total amount of head movement, **int** head which is used to update the algorithm to a new head location, and **int** distance which stores the distance between the current head and the next head.

For CSCAN, the method follows the format for SCAN on assignment 4. It is followed by three vectors 127 Vector<Integer> left = **new** Vector<Integer>(), right = **new** Vector<Integer>() and 128 Vector<Integer> seekSeq = **new** Vector<Integer>().The first two are used for both directions left and right, while the last is used to check the sequence of the head. The method then checks for edge cases for the last values in both directions, and reverses the direction for both left and right. They are then put through a sort using 83 Collections.*sort*(left) and 84 Collections.*sort*(right). It then goes through a loop for the left, adding the head to the sequence vector and then following finds the distance between the current head and the next head 152 distance = Math.*abs*(head - pos), updates the head movement count with the distance 154 count = count + distance, and updates the head to the new position 156 pos = head. The head is then set to 0, which follows a circular loop which is then continued on in the next format to go through the right side. This follows the same format for both LOOK and CLOOK.

For LOOK, there is also 62 String direction = "left", which takes in the direction, in this case it is set to left. Following the same format as SCAN, the only difference is that instead of checking for edge cases for the values in both direction and reversing it or both left and right side, they are just put through a loop which adds values from both left and right directions from the current head position. It then follows the same collection sort and left and right loop that CSCAN follows. The difference is that after the left loop, the head is not set back to 0, but kept at its last position before continuing to the right direction.

For CLOOK, it follows the same method and format as LOOK, however there is no need for a string direction as it loops circularly. As such after going through the same initial format of adding values and sorting them, we start with the left side and after completing that, we also get the distance from the current head and its next head position and then set the current head back to 0 to make a circular loop. Then it follows the same format for the right side.

The main function creates and initializes both array of 286 **int**[] cylinders = **new** **int**[5000] and array of 288 **int**[] cylinders = **new** **int**[5000]. It then allows for user input for the initial head location which is then used and passed into the three algorithm methods right after, printing the head movement count for CSCAN, LOOK and CLOOK respectively.