CS431 Homework 2
Disk Arm Scheduling Algorithm
Due May 16th (MW class) or May 17th (TuTh class)

For this homework, you will be investigating the disk arm scheduling algorithms, plus you will make a new algorithm of your own design. What you will turn in is a zip file of your code, plus a report describing your algorithm and giving a chart comparing the various algorithms.

You can use either Java or C++ for this project.

We will be modelling disk arm scheduling algorithms. The sample disk we will be using has 100 cylinders. To simplify things, assume that it takes '1 time unit' to change from cylinder *i* to *i + 1*, or from *i* to *i - 1*. We will ignore rotational delays, and we will ignore the time it takes to transfer a block of data: we are only interested in the sum of the seek delays.

Your program should do the following:

1. Ask for the number of seeks to be generated. As you are testing your program, you may want to use smaller counts, such as 10 or 20. For the final report, use a count of 1,000. Generate a *master list* containing that number of random cylinder seeks, from 1 to 100. So while you are testing, your list might contain twenty values that are between 1 and 100, but for the final report, your list will contain a thousand values between 1 and 100. Clearly there will be duplicate numbers in the list!
2. Write a routine that simulates the action of the First Come First Served (FCFS) scheduling algorithm. This routine will assume that the disk arm started on cylinder 50. It would then move to the first value in your master list, recording the number of steps taken to move the heads to that cylinder. For example, if your first value was 75, then it would take 25 time units to get to that cylinder, so set your running sum to 25. Then move to your second value. Let's say your second value was 37. To move from 75 to 37 takes 38 steps, so add 38 to your running sum. The output from this routine will be that running sum, the total number of time units to perform all of the seeks.
3. Write a routine that simulates the action of the Shortest Seek First (SSF) scheduling algorithm. This routine is a little bit trickier, since we can be changing the order of the seeks. Not only that, but we don't want all of the requests to enter the system at the same time! To implement this routine, do the following:

   a. Have a counter, *time*, which starts at zero. We will in essence be incrementing this number once for each step up or down we move the disk heads.
   b. Have a value, *cylinder*, which indicates the current cylinder of the disk. Initialize this value to 50.
   c. Make a class, *seqReq*, which represents one seek request. The class should have a property indicating the time at which the request entered the system, the time at which the request was satisfied, and the cylinder for this request.
   d. Have a method of *seqReq* which gives the delay for this request, which is simply the end time minus the start time.
   e. Have a second method which gives the *score* for this request, which is simply the delay times the square root of the delay. We want to minimize the overall scores, and this score function penalizes us for having requests sitting in the queue for too long.
   f. Grab the first 10 values from your master list, creating a list, *queue*, of 10 *seqReq* instances. These will all have a start time of 0, since *time* is currently 0. As you are running this simulation, you will be removing instances from the queue. As you remove the instance, you will move the disk head to that instance's cylinder number. Whenever the number of instances in the queue drops below 5, add 10 more values from the master list into the queue (but these next instances will have a different starting time value, since we will be increasing the time as the simulation goes on.
   g. When you process a *seqReq* (as it is removed from the *queue*), move the *cylinder* to the number of the request. As you do, compute the absolute value of the difference between the current cylinder and the new cylinder, adding this to *time* (so *time* will be incremented once for each cylinder the heads have to pass through to get to the new point). Once you have updated *time*, you can set the end time for the *seqReq*.
   h. As the request is completed, get its delay and score, using the methods from above. For the final report, we want to know the average delay for all the requests, the maximum delay, the average score, and the maximum score.
4. Write a third algorithm, the Elevator algorithm, which is similar to SSF, but uses the elevator algorithm. Assume that the head starts on cylinder 50, and is moving *up*.
5. Write your own algorithm for doing the scheduling. It should be similar to both SSF and Elevator in that it keeps an internal list of about 10 requests that it is

processing at a time. However, we want to minimize the average score and the maximum score, we don’t want any request to be waiting too long to be handled. After you have your program working well for the first three algorithms, design your own, see what you can come up with!

6. Once your program has generated the master list, it should pass the list to each of the algorithms. Take the average and maximum delays and scores from each algorithm, printing the results in a little chart.
7. Finally, write a short report which presents your findings. Include the chart, and also include a description of your algorithm. Your description should be detailed enough that a fellow classmate (or me!) can understand what your algorithm does.
8. Don’t forget to turn in your report and a zip of your source files.