CS 2336.004 Spring 2017

Assignment 2 – Recursion & Sorting

Feb 13th, 2017
Due Date: Feb 26th, 2017 – 11:59pm

Q1. A Recursive Puzzle

You have been given a puzzle consisting of a row of squares each containing an integer, like this:

3	6	4	1	3	4	2	5	3	0

The circle on the initial square is a marker that can move to other squares along the row. At each step in the puzzle, you may move the marker the number of squares indicated by the integer in the square it currently occupies. The marker may move either left or right along the row but may not move past either end. For example, the only legal first move is to move the marker three squares to the right because there is no room to move three spaces to the left.
The goal of the puzzle is to move the marker to the 0 at the far end of the row. In this configuration, you can solve the puzzle by making the following set of moves:

Starting Position:

3	6	4	1	3	4	2	5	3	0

Step 1: (move right)

3	6	4	1	3	4	2	5	3	0

Step 2: (move left)

3	6	4	1	3	4	2	5	3	0

Step 3: (move right)

3	6	4	1	3	4	2	5	3	0

Step 4: (move right)

3	6	4	1	3	4	2	5	3	0

Step 5: (move left)

3	6	4	1	3	4	2	5	3	0

Step 6: (move right)

3	6	4	1	3	4	2	5	3	0

Even though this puzzle is solvable—and indeed has more than one solution—some puzzles of this form may be impossible, such as the following one:

3	1	2	3	0

In this puzzle, you can bounce between the two 3's, but you cannot reach any other squares.

Write a function **bool Solvable(int start, int[] squares)** that takes a starting position of the marker along with the array of squares. The function should return **true** if it is possible to solve the puzzle from the starting configuration, and **false** if it is impossible. You may assume all the integers in the array are positive except for the last entry, the goal square, which is always zero.

Q2. Algorithm Efficiency and Sorting

Implement **Heap Sort and Radix Sort** Algorithm we discussed in class and **Bubble sort** we learned before. In addition, write a function to measure the sorting time. Use any random generator (e.g. https://www.random.org/integers/) to generate 10, 50, 100, 500, 1000, 5000, 10000 random integer numbers and sort these numbers using Heap Sort, Radix Sort, and Bubble Sort. Present a graph (a clear jpeg/png image) which plots the size of the array (N, x-axis) against the time taken to Sort (y-axis), and describe your observation from the results (e.g. Does the plot of running time reflect the big-O notation).

Deliverables:

Turn in your code file(s) (.java or .cpp) and description (plot) file (e.g. pdf).