Computing Structures – Fall 2022
Project 6
Due: December 3rd 2022 at 11:59 PM

Objective:

You are going to be exploring two sorting algorithms – Shell sort and Bubble sort; and how the concept of *approximate sorting* works along with how we can measure, how good/bad an array of elements are sorted.

Description:

The two sorting algorithms that are given to you are Bubble and Shell sort. Each of these algorithms are in the order of $O(n^2)$ in the worst case. And they take certain number of comparisons of elements to get to the result (completely sorted array). But what if we don't have enough time to make all those comparisons?

In this project, we put a constraint on the number of comparisons that can be made for each of the sorting algorithms. The two sorting algorithm methods are given in the boiler plate file. You may see that there is a check before every comparison to see if we go past the given number of comparisons or not.

If the constraint on the number of comparisons is more than what is necessary by the algorithm, it would be completely sorted and we shall get the desired result. But if that is not the case, it would result in an approximately sorted array.

How would we measure the 'sortedness' of this approximately sorted array now? That is why we define quality measures. These quality measures are used to see how good/bad the array is sorted. To put it in a simple way, how far is this approximately sorted array from a completely sorted array. While we refer to sorting, we are assuming it as ascending order.

Quality Measures:

Number of Inversions:

An inversion in a given random set of numbers σ is a pair (σ(i),σ(j)) such that $i < j$ and $\sigma(i) > \sigma(j)$. Where i and j are index values and σ(i) and σ(j) are the numbers at those positions.

Maximum displacement/Chebyshev distance:

Given random set of numbers σ1, and the same set of numbers completely sorted σ2, the Chebyshev distance (dl) between σ1 and σ2 is

$$dl_{\infty}(\sigma 1,\sigma 2) = |\sigma 1 - \sigma 2|_{\infty} = max\ |\sigma 1(i) - \sigma 2(i)|.$$

The method to calculate the inv(σ) and dl for a given array of numbers is explained below with an example.

Examples:

Let us take this array as an example:

A = [1, 5, 4, 2, 3]

For each element (A[i]) in the array, you count the number of elements less than A[i] to the right of the element (i to n-1 indices). Let us look at those numbers and how they would turn out for each of them:

- For the element A[0] it is 0 because there are no elements to the right that are less than 1.
- For the element A[1] it is 3 because there are 3 elements to the right that are less than 5.
- For the element A[2] it is 2 because there are 2 elements to the right that are less than 4.
- For the element A[3] it is 0 because there are no elements to the right that are less than 2.
- For the element A[4] it is 0 because there are no elements to the right that are less than 3.

Now we add these numbers from above (0 + 3 + 2 + 0 + 0) = 5. This sum of 5 is the number of inversions necessary to go from this given array example to a completely sorted array.

Now let us take a similar example and look at our second quality measure –Chebyshev distance.

B = [10, 50, 40, 20, 30]

Let us refer to the sorted version of the array as B_s = [10, 20, 30, 40, 50]. Now looking at B and B_s, we first count the displacement of each of those elements.

- For B[0], the displacement is 0 because the element 10 is already at the correct position/index.
- For B[1], the displacement is 3 = |4 – 1|. 4 is the actual position where 50 has to be at B_s and 1 is the current position at B.
- For B[2], the displacement is 1 = |3 – 2|. 3 is the actual position where 40 has to be at B_s and 2 is the current position at B.
- For B[3], the displacement is 2 = |1 – 3|. 1 is the actual position where 20 has to be at B_s and 3 is the current position at B.
- For B[4], the displacement is 2 = |2 – 4|. 2 is the actual position where 30 has to be at B_s and 4 is the current position at B.

The above underlined numbers are the displacements for each of the elements in B. We now calculate the **maximum displacement/Chebyshev distance** which will be the maximum of all of the displacements which will be 3.

Input explanation:

The first line in the input file is the number of elements(numElements). The line following that has the seed followed by the lower range and the upper range for random number generation.
The seed is going to dictate the random numbers that are going to be generated. And is dependent on the computer that is being used. So the output that you may have on your computer maybe different to the one that is being provided. But when you upload your program to GradeScope, it will use the seed given and generate the same numbers as generated by the solution program on GradeScope. So, just stick to the output format and don't worry about the numbers being generated differently.

The last line of the input file will be the number of comparisons (D) that the sorting algorithms must go through with the generated random numbers.

Redirected input:

Redirected input instructions from the previous project need to be followed to set everything up.

Submission:

Submission will be through *GradeScope*. Your program will be autograded. You may submit how many ever times to check if your program passes the test cases provided. One test case will be released at the beginning and later other test cases will be released while grading. For the autograder to work, the program you upload must be named as ***project6.cpp***.

Along with the program, you are asked to submit a written ***report*** comparing the quality metrics (y-axis) and the number of comparisons (D) (x-axis). You'd have draw 2 line charts (one for each quality metric) for each 10000, 50000, 800000 elements (n). Each line graph will have two lines, one for bubble sort and another for shell sort.

You are required to change the given input file with the various number of elements and also the number of comparisons for experimentation. You may use any application/program (MS excel, python, R…) to generate the charts.

We expect a total of 6 line charts (with their tabulated results) and then a write up, explaining and exploring the results of the line graph. This report has to be in PDF format (project6_report.pdf). The report can be upto 2 pages long.

Your final submission must contain your source file named project6.cpp and project6_report.pdf.

Constraints:

1. In this project, the only header you will use is iostream, cstdlib (for random number generation), unordered_set (used for generating unique random numbers) and using namespace std.
2. None of the projects is a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.
    - Please review academic integrity policies in the modules.

How to ask for help:

Same as before.