Assessment 2 – Report Upload Algorithms and Complexity Weronika Wawrzyniak 17643904

Implementation of the task

My program is built of 4 Sorting Methods (Quick, Heap, Bubble and Merge) and 2 Searching Methods (Interpolation and Binary). Each sorting algorithms has a twin sorting in different order. Each searching method has additional method that shows location of searched value. There are also “loadArray” method that loads array from file of user's choice, “neighbourValues” method, that in case of value not being in the array shows the values and indexes of the closest value, “mergeArrays” method, which merges two arrays of user's choice.

At the beginning User is asked which text file would he like to read data from. I am aware that in the given instructions program was supposed to load each 128 file into individual array but I believe it is a waste of memory. The code uses "loadArray" method that contains File.ReadAllText method to read all text from the file into a string and then uses the string.Split method to put each number into a string array and by using foreach loop parse it into double array. I also applied try catch in case data is not a number. My main problem was using polish version of Visual Studio Player which reads decimal numbers with commas, not dots, so I needed to use item.Replace(('.'), (',')) method.

In step 1 User inputs file name and is asked to choose which sorting algorithm wants to use. To avoid confusion all questions contain in brackets examples of answers and I also used ToLower() tool to improve User's experience and make sure program runs smoothly. After choosing sorting algorithm User can decide whether to sort it ascending or descending.

For Step 2, I wrote sorting algorithms and then added another methods with slightly changed code (I only reversed greater/lower-than symbol) so they would sort ascendingly or descendingly. I also considered just reversing sorted arrays but it is not a very efficient solution.

In step 3 I decided to go with Interpolation Search (for ascending arrays) and Binary Search (for descending ones). Algorithms return index in case of finding the value and -1 in case of not. After this, my program uses “showLoaction” method that declares integer ‘location’ as a result of searching algorithm. If the result was -1, method prints out that the value was not found in the array and goes to “neighbourValues”. Otherwise by using two for loops, one starting at location and going up, second going down the array , it checks which values in the array equals searched value and print out its indexes. If the value is not equal, the loop breaks. At first I went with one for loop checking every number in the array but the I realized it is inefficient and modified it to compare only the closest values.

For task 4 I have written “neighbourValues” method, which contains a for loop checking by if statement if valueToFind is between values next to each other. If it is it print out indexes. The method also sets in a for loop variable distance as absolute value of substraction of searched value and array[i]. Then if statements checks in distance < minDistance and sest minDistance = distance minIndex = i. At the end it prints out the closest index.

When it comes to task 5, I did not do anything. My program was capable of running 256 and 1024 files without me changing anything in the code.

In task 6 I created method “mergeArrays” which takes two strings as arguments. Strings are names of files User wants to merge and are included in paths. Method reads files and puts data into two arrays the same way as “loadArray”. I have created third array of the size of added sizes of two arrays. I made two for loops. First loop was starting index 0 and finishing at thirdArray.length/2 and put values from first array into third array. Second loop was starting at thirdArray.length/2 and finishing at thirdArray.length-1 and put values from second array indexes [i-(thirdArray.Length)/2] into the third one (looks complicated but I needed to find a way to keep filling higher indexes in third array and at

the same time taking values from regular array starting at index 0). Repeating Tasks 2 to 4 was not a problem, because I just used previously written methods.

Again, for task 7 I did not change anything in my code because it worked for both 256 and 1024 files.

Description of algorithm choices

For my program I have chosen Quick Sort, modified Bubble Sort, Merge Sort and Heap Sort. I used Merge and Heap sorts because of their polylogarithmic average, best and worst running time (which is the fastest compering to other sorting algorithms). Unfortunately, but for their excellent work they have some flaws. When it comes to merge sort, its virtue is that it is stable (it doesn't change the order of equal items like heap sort often does), but its main problem is that it requires a second array with the same size as the array to be sorted, doubling the memory requirements. On the other hand in the nearly sorted case, the heapify method destroys the original order, so none of those algorithms is perfect. I also decided to go with Quick Sort because it is slow (quadratic) only in its worst case but average and best are polylogarithmic. Choosing last algorithm, I could not decide between Insertion Sort and Bubble Sort. Insertion Sort best running time might be linear, but only in a case of already sorted array and is good only for sorting small arrays (usually less than 100 items). Taking into consideration data from the assessment 2 I realized that this algorithm does not suit my needs and on top of that will only run on its worst or average time which is quadratic. On the other side, regular bubble sort has quadratic best, worst and average running time, but after modifying the code so that it would stop after reaching a sorted array best time becomes linear. I never took into consideration using Selection Sort because even a perfectly sorted input requires scanning the entire array and its running time is always quadratic.

In my code I used Binary Search and Interpolation Search. I decided to use for descendingly sorted arrays Binary Search because of its logarithmic work thanks to searching by divide-and-conquer. With second searching algorithm, tempted by the fact that in average-case, Interpolation Search on equally distributed data requires only O(log(log(N)) comparison and in the worst case its linear (so still better than Linear Search which I was also considering), I decided to go with this improved variant of binary search to search in ascendingly sorted arrays.

Number of steps in Sorting Algorithms

NAME	NUMBER OF STEPS ASCENDING	NUMBER OF STEPS DESCENDING
Merge Sort Change_128	1792	1792
Merge Sort Change_256	4096	4096
Merge Sort Change_1024	20480	20480
Heap Sort Change_128	1977	2031
Heap Sort Change_256	4676	4663
Heap Sort Change_1024	23710	23524
Quick Sort Change_128	1259	1331
Quick Sort Change_256	2923	3053
Quick Sort Change_1024	12877	12425
Modified Bubble Sort Change_128	8151	7878
Modified Bubble Sort Change_256	32881	32596
Modified Bubble Sort Change_1024	523474	523719

YOUTUBE VIDEO : https://youtu.be/esO-E6YD8wU