CME 2204 Assignment 1

ShellSort and DualPivotQuickSort

30.03.2024 23:55 (Sharp deadline – No extension)

Rules:

- The submissions will be checked for the code similarity. Plagiarism will be graded as zero.
- You are required to upload two different files. One is the source code you have written in Java programming language and the report.
- The files you are required to upload are given below with explanations(single zip file of project folder):
  (STUDENT_NUMBER)_(STUDENT_NAME)_HW1_Code.zip
  Example = 2050901815_Augusta_Ada_HW1_Code.zip
  (STUDENT_NUMBER)_(STUDENT_NAME)_HW1_Report.pdf
  Example = 2050901815_Augusta_Ada_HW1_Report.pdf

Implementation:

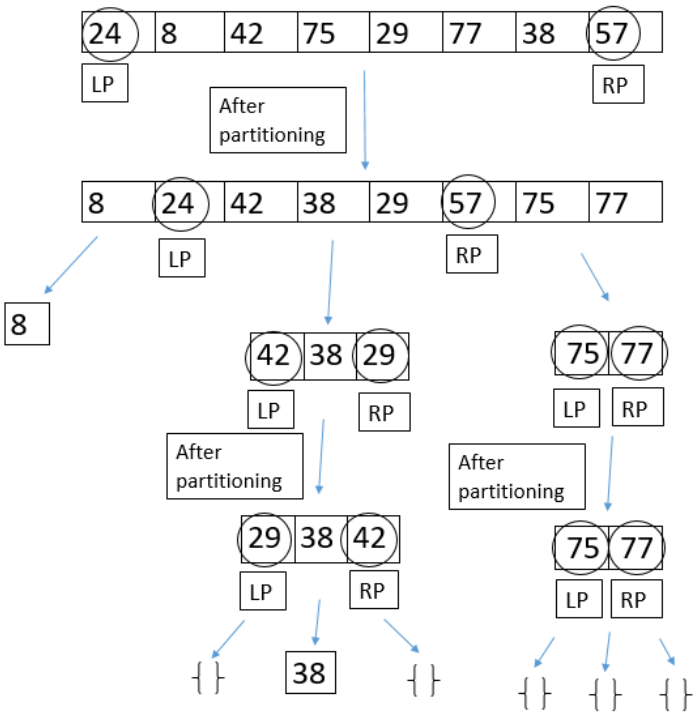
In this assignment you will implement and test two sorting algorithms: **Shellsort** and **DualPivotQuickSort**. You will write a class and own methods. Sorting algorithm methods will be invoked from this class;

```
public class SortingClass {
        dualPivotQuickSort (int[] arrayToSort) { .... }
        shellSort (int[] arrayToSort) { .... }
}
```

In the ***dualPivotQuickSort*** method, you must implement a different version of quicksort. This version has two pivots: one in the left end and one in the right end of the array. The left pivot (LP) must be less than or equal to the right pivot (RP).

$< LP$	LP	$LP \leq \ \& \ \leq RP$	RP	$RP <$

Input array is divided into three parts. In the first part, all elements will be less than LP, in the second part all elements will be greater or equal to LP and will be less than or equal to RP, and in the third part all elements will be greater than RP. Until the existing pivots have stopped their moving, sorting process continues recursively.



In the ***shellSort*** method, you must implement an improved version of the insertion sort algorithm. Unlike classical insertion sort, instead of comparing contiguous items, shellSort breaks the main array into several subarrays using an interval "i" (which is known as the gap), then sorts the subarrays by insertion sort. These steps are repeated until the gap reaches to 1.

An example run of Shellsort with gaps 4, 2, and 1 is shown below.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
Input Data	3	22	14	1	31	10	7	25
4-gap Sorting	3	10	7	1	31	22	14	25
2-gap Sorting	3	1	7	10	14	22	31	25
1-gap Sorting	1	3	7	10	14	22	25	31

The first pass, 4-gap sorting, performs insertion sort on four separate subarrays (*a1*, *a5*), (*a2*, *a6*), (*a3*, *a7*), (*a4*, *a8*). For instance, it changes the subarray (*a2*, *a6*) from (22, 10) to (10, 22). The next pass, 2-gap sorting, performs insertion sort on two subarrays (*a1*, *a3*, *a5*, *a7*), (*a2*, *a4*, *a6*, *a8*). The last pass, 1-gap sorting, is an ordinary insertion sort of the entire array (*a1* ,..., *a8*).

Researchers have published different approaches for gap sequences. You can use Shell's original sequence gap-length, i.e., ⌊N/2⌋ , ⌊N/4⌋ , ⌊N/8⌋ , …, 1 where N is the input size.

Runtime Analysis:

You are expected to compare each sorting algorithm according to their running times (in milliseconds) for different inputs and fill the following table accordingly.

	EQUAL INTEGERS			RANDOM INTEGERS			INCREASING INTEGERS			DECREASING INTEGERS		
	1.000	10.000	100.000	1.000	10.000	100.000	1.000	10.000	100.000	1.000	10.000	100.000
dualPivot QuickSor												
shellSort												

- You must create the input arrays by yourself in the Test class according to the specified rules (1.000, 10.000, 100.000 of each equal, random, increasing, and decreasing order).

- Note that, the elapsed time of creating the arrays should not be considered during the calculation of the total running time of sorting algorithms. You only must check how long the sorting algorithm is running.

- One of the methods you can use to measure the time elapsed in Java is given below:

```
 n=1000     // array size
int arrayToSort = new int[n];
 for (int i = 0 ; i < n ; i++)
      arrayToSort[i] = i;       // generate numbers in increasing order

long startTime = System.currentTimeMillis();
shellSort(arrayToSort);          // run one of the sorting methods
long estimatedTime = System. currentTimeMillis() - startTime;
```

Scientific Report:

Prepare a scientific report in the light of the results you obtained from your program and the knowledge you learned in the lectures. Establish a connection between the asymptotic running time complexity (in Θ notation) and the results (in milliseconds) of your experiments. Your report should include a comparison table (as shown in the previous page). Additionally, you are expected to determine the appropriate sorting algorithm for the following scenarios. Which algorithm (shellSort or dualPivotQuickSort) would you choose and why? Explain your thoughts and reasons clearly.

You must add all the code and discussions you have prepared with the help of public internet resources or AI software into your report as a reference list.
(Continued on next page).

Scenario A:
The goal is to place students at universities according to their central exam (YKS) grades and preferences. If there are 3 million of students in the exam, which sorting algorithm would you use to do this placement task faster?

Scenario B:
You have a Turkish-English dictionary with a single word translation for each word in alphabetical order. You aim to translate it into an English-Turkish dictionary. For example; if your Tur-Eng dictionary contains [ayna : mirror, bardak : glass, elma : apple, kitap : book] then your new Eng-Tur dictionary should contain [apple : elma, book : kitap, glass : bardak, mirror: ayna]. Assume that there are thousands of words in your dictionary, which sorting algorithm do you use to perform this translation faster?

Grading Policy:

Task	**%**
dualPivotQuickSort	30
shellSort	20
Runtime Analysis	30
Scientific Report (Scenario A+B)	20