**Introduction**

In this lab you will compute the running time of bucket sort, which is also a linear sorting algorithm. Submit your answers to the questions below in a text file (e.g. Word document). Name your file in name\_surname.docx format. Submit your solution document and Java codes as a zip folder in name\_surname format to Canvas.

You can use the code templates in linear\_2.java in this lab.

**Problem Statement**

Given an array of real-valued numbers sort the numbers in ascending order.

**Assignment**

1. (a) Implement a Java method for the bucket sort algorithm given below. Note that the indices for array start from 1 while those of start from 0. You may need to do necessary adjustments in the indexing of these arrays. You can use the template class for singly linked lists.

|  |  |
| --- | --- |
|  |  |

(b) Test your algorithm by choosing an array of size 10. Initialize your array by random real numbers from 0 to 1. For this purpose, you can use the nextFloat method in Random class of Java. Make sure your program sorts the array correctly. Include the output of your program for this sample input in your report.

🡪 [0,172210, 0,759426, 0,231027, 0,334355, 0,360610, 0,903920, 0,795991, 0,478275, 0,254408, 0,022550]: randomly generated array with the size of 10

🡪 [0,022550, 0,172210, 0,231027, 0,254408, 0,334355, 0,360610, 0,478275, 0,759426, 0,795991, 0,903920] :sorted array with bucket sort

🡪 Elapsed time in nanoseconds for bucket sort: 901200

(c) Choose input sizes in the table below, which are multiples of 10, and initialize the values in your array by random numbers from 0 to 1. Compute the running time of merge sort, heap sort and bucket sort in nanoseconds for each of these input sizes and include them to the table below. The codes for merge sort and heap sort are available in the code template but only for sorting integers. You need to update these methods so that they can sort floating point values. Write a for loop that performs these operations automatically. Do not run them one at a time.

|  |  |  |  |
| --- | --- | --- | --- |
| Input size | Bucket sort running time | Merge sort running time | Heap sort running time |
| 10 | 29200 | 13500 | 15000 |
| 100 | 114100 | 90700 | 149500 |
| 1000 | 1536900 | 723900 | 270600 |
| 10000 | 5607800 | 2591900 | 2516600 |
| 100000 | 44230100 | 23080400 | 17067800 |
| 1000000 | 462283000 | 164342600 | 213950200 |
| 100000000 | 5706031000 | 17932226300 | 43417991800 |

Which algorithm performs best at which input size?

**🡪** For the smaller input sizes merge sort runs better than others.

(d) Set the input size to 100000000 and initialize the array with random numbers from 0 to 1. Run bucket sort, merge sort and heap sort one at a time for this input size. Open a terminal window and type top. Find the processes for the sorting algorithm you executed and record the RAM usage in MEM column. Include the RAM usage of these algorithms into the table below. Compare and comment on the RAM usage of these sorting algorithms.

|  |  |  |  |
| --- | --- | --- | --- |
| Input size | Buck sort RAM | Merge sort RAM | Heap sort RAM |
| 100000000 | 820.213 | 819.774 | 818.224 |

* The bucket sort has the biggest RAM usage. And the heap sort is least one. But the Ram usage of these three sorts are close to each other.