**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.

**Array before implementing bucket sort:**

**[0.496189, 0.473741, 0.291860, 0.129880, 0.940070, 0.694384, 0.536453, 0.261176, 0.390510, 0.562679]**

**Array after implementing bucket sort:**

**[0.129880, 0.261176, 0.291860, 0.390510, 0.473741, 0.496189, 0.536453, 0.562679, 0.694384, 0.940070]**

(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 | 42500 | 527799 | 32500 |
| 100 | 685100 | 265300 | 377400 |
| 1000 | 5260500 | 16254900 | 979600 |
| 10000 | 31186699 | 6801000 | 5542500 |
| 100000 | 92179199 | 31907499 | 31275199 |
| 1000000 | 542055401 | 387975900 | 290885500 |
| 100000000 | 14281595000 | 4138480500 | 3844282100 |

Which algorithm performs best at which input size?

**The results showed that heap sort performs best at all input sizes when compared to merge sort and bucket sort.**

(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 | Bucket sort RAM | Merge sort RAM | Heap sort RAM |
| 100000000 | 625,344 | 557,660 | 547,460 |

Bucket sort uses higher amount of RAM when compared to heap sort and merge sort because it uses memory for each index of array list.. And merge sort uses more RAM than heap sort because, it uses additional arrays in order to merge two sorted subarrays.