[CENG 315 All Sections] Algorithms

Navigation

- Dashboard
  - Site home
  - Site pages
  - My courses
    - 571 - Computer Engineering
      - [CENG 351 All Sections]
      - CENG 300 All Sections
      - CENG 300 Section 4
      - CENG 315 All Sections
        - Participants
        - Badges
        - Competencies
        - Grades
        - General
        - October 13 - October 19
        - October 20 - October 26
        - October 27 - November 2
        - November 3 - November 9
        - November 10 - November 16
        - November 17 - November 23
          - THE3 Discussion Forum
          - THE3
            - Description
            - Submission view
          - THE3_IO_B
          - THE3_IO_Officials
          - THE3 Solution
        - November 24 - November 30
        - December 1 - December 7
        - December 8 - December 14
        - December 15 - December 21
        - December 22 - December 28
        - December 29 - January 4
        - January 5 - January 11
        - January 12 - January 18
        - January 19 - January 25
      - CENG 315 Section 3
      - CENG 331 All Sections
      - CENG 331 Section 2
      - CENG 351 Section 3
    - 651 - Music and Fine Arts
    - 612 - Modern Languages (Persian)
    - 642 - Turkish Language

Description | Submission view

THE3

Available from: Friday, November 19, 2021, 11:59 AM
Due date: Friday, November 19, 2021, 11:59 PM
Requested files: test.cpp, the3.cpp (Download)
Type of work: Individual work

Specifications:

- There are 1 **task** to be solved in **12 hours** in this take home exam.
- You will implement your solutions in **the3.cpp** file.
- You are free to add other functions to **the3.cpp**
- Do **not** change the first line of **the3.cpp**, which is **#include "the3.h"**
- *<iostream>, <climits>* , *<cmath>* , *<string>* are included in "the3.h" for your convenience.
- Do **not** change the arguments and return value of the functions **radixSort()** in the file **the3.cpp**
- Do **not** include any other library or write include anywhere in your **the3.cpp** file (not even in comments).

- You are given **test.cpp** file to **test** your work on **Odtuclass** or your **locale**. You can and you are encouraged to modify this file to add different test cases.
- If you want to **test** your work and see your outputs you can **compile** your work on your locale as:

```
>g++ test.cpp the3.cpp -Wall -std=c++11 -o test

> ./test
```

- You can test your **the3.cpp** on virtual lab environment. If you click **run**, your function will be compiled and executed with **test.cpp**. If you click **evaluate**, you will get a feedback for your current work and your work will be **temporarily** graded for **limited** number of inputs.
- The grade you see in lab is **not** your final grade, your code will be reevaluated with **completely different** inputs after the exam.

The system has the following limits:

- a maximum execution time of 32 seconds (your functions should return in less than 1 seconds for the largest inputs)
- a 192 MB maximum memory limit
- an execution file size of 1M.
- Solutions with longer running times will not be graded.
- If you are sure that your solution works in the expected complexity constrains but your evaluation fails due to limits in the lab environment, the constant factors may be the problem.

```
int radixSort(std::string arr[], bool ascending, int n, int l)
```

In this exam, you are asked to sort the given array *arr* with Radix Sort ascending or descending depending on the boolean variable *$ascending$* and return the number of iterations done in the loops of the Counting Sort algorithm (you need to use Counting Sort as a subroutine in the Radix Sort). *n* is the number of elements. You are expected to use Counting Sort for *l* digits at each time.

IMPORTANT: Different than the algorithm for Counting Sort in your book, initialize count array as *int* C = new int[k]* and use the fourth loop for copying the array back. Otherwise the return value of the function (as the number of iterations) will not be evaluated as correct.

Constraints:

- Maximum array size will be **1000000**.
- Array elements will be strings each of which can contain only the characters as **uppercase English letters (i.e. from 'A' to 'Z').**
- For the sake of simplicity, it is guaranteed that the strings in the array will always have the **same** length. This length can be at most **12**.
- ***l*** may take values **1**,**2**,**3**,**4** or **6**.

Evaluation:

- After your exam, black box evaluation will be carried out. You will get full points if you fill the ***arr*** variable as stated **and** return the number of iterations correctly for the cases that will be tested.

Example IO:

1) Given array arr = {"AAA", "ABC", "ABA", "CCB"}, size = 4, l = 1, ascending = true:

- fill arr = { "AAA", "ABA", "ABC", "CCB" }
- return 114

2) Given array arr = {"BAAA", "AABC", "CDBA", "CACB", "ABAB", "ACAB", "CBCB"}, size = 7 l = 1, ascending = true:

- fill arr = { "AABC", "ABAB", "ACAB", "BAAA", "CACB", "CBCB", "CDBA" }
- return 188

3) Given array arr = {"BAAA", "AABC", "CDBA", "CACB", "ABAB", "ACAB", "CBCB"}, size = 7 l = 2, ascending = true:

- fill arr = { "AABC", "ABAB", "ACAB", "BAAA", "CACB", "CBCB", "CDBA"}
- return 1394

4) Given array arr = {"BAAA", "AABC", "CDBA", "CACB", "ABAB", "ACAB", "CBCB"}, size = 7, l = 3, ascending = false:

- fill arr = { "CDBA", "CBCB", "CACB", "BAAA", "ACAB", "ABAB", "AABC" }
- return 17644

5) Given array arr = { "NWLRBBMQBHCD", "ARZOWKKYHIDD", "QSCDXRJMOWFR", "XSJYBLDBEFSA", "RCBYNECDYGGX", "XPKLORELLNMP", "APQFWKHOPKMC", "OQHNWNKUEWHS", "QMGBBUQCLJJI", "VSWMDKQTBXIX" }, size = 10, l = 3, ascending = true:

- fill arr = { "APQFWKHOPKMC", "ARZOWKKYHIDD", "NWLRBBMQBHCD", "OQHNWNKUEWHS", "QMGBBUQCLJJI", "QSCDXRJMOWFR", "RCBYNECDYGGX", "VSWMDKQTBXIX", "XPKLORELLNMP", "XSJYBLDBEFSA" }
- return 70424

TEST EVALUATION:
Due to the limitation of our programming environment, larger inputs can not be stored. Therefore, we create them when needed. The test evaluation has 2 phases. The first phase has the same inputs given here to check if your codes work fully correct on small inputs. If your code works perfectly on at least one of the first 4 tasks, it will also be tested on the second phase for the task(s) that works correct. The second phase on the other hand, creates and sorts larger arrays that are on boundaries. (Note that the tests give 50 pts for each phase. However, the real inputs will be like the ones on the second phase which means if your code works only on phase 1, it is possible for your real grade to be 0 afterwards).

Requested files

test.cpp

```cpp
// this file is for you for testing purposes, it won't be included in evaluation.

#include <iostream>
#include <random>
#include <ctime>
#include "the3.h"

char getRandomChar(){
    return 'A' + (rand() % 26);
}

std::string getRandomString(int length){
    char* result_array = new char[length];
    for(int i = 0; i < length; i++){
        result_array[i] = getRandomChar();
    }
    std::string result(result_array, length);
    return result;
}

void randomArray(std::string*& array, int size, int length)
{
  array = new std::string[size];
  for (int i = 0; i < size; i++)
  {
    array[i] = getRandomString(length);
  }
}

void printArrayInLine(std::string arr[], int arraySize){
    std::cout << "{ ";
    for(int i = 0; i < arraySize; i++){
        std::cout << "\"" << arr[i] << "\"";
        if (i == arraySize - 1){
            continue;
        }else{
            std::cout << ", ";
        }
    }
    std::cout << " }" << std::endl;
}

void test(){
    clock_t begin, end;
    double duration;
    int numberOfIterations;

    int size = 10;      // max 1000000
    int length = 5;     // max 12
    int l = 2;      // number of characters to be used in counting sort (1,2,3,4 or 6)

    std::string* arr;
    randomArray(arr, size, length);
    std::cout << "Array before sorting:" << std::endl;
    printArrayInLine(arr, size);
    if ((begin = clock() ) ==-1)
        std::cerr << "clock error" << std::endl;

    numberOfIterations = radixSort(arr, true, size, l);

    if ((end = clock() ) ==-1)
        std::cerr << "clock error" << std::endl;

    duration = ((double) end - begin) / CLOCKS_PER_SEC;
    std::cout << "Duration: " << duration << " seconds." << std::endl;

    std::cout << "Number of Iterations: " << numberOfIterations << std::endl;
    std::cout << "Array after sorting:" << std::endl;
    printArrayInLine(arr, size);
}

int main()
{
    srandom(time(0));
    test();
    return 0;
}
```

the3.cpp

```cpp
#include "the3.h"

// do not add extra libraries here


/*
    arr         : array to be sorted, in order to get points this array should contain be in sorted state before returning
    ascending : true for ascending, false for descending
    n           : number of elements in the array
    l           : the number of characters used in counting sort at each time

    you can use ceil function from cmath

*/
int radixSort(std::string arr[], bool ascending, int n, int l){

    return 0;

}
```

VPL