CS590 B homework 2 – Recurrences and sorting

The due date for this assignment is Monday, October 16th, at 11.59pm. This assignment is worth 10% of your final grade.

Any sign of collaboration will result in a 0 and being reported to the Graduate Academic Integrity Board. The programming assignment will be done individually. No collaboration is allowed between students. No code from online resources is allowed to be used besides the code that I will share with you. Late submission policy described in the syllabus will be applied.

Part 1 (20 points)

Recurrences: Solve the following recurrences using the substitution method. Subtract off a lower-order term to make the substitution proof work or adjust the guess in case the initial substitution fails.

1. $T(n) = T(n-3) + 3\lg n$. Our guess: $T(n) = O(n \lg n)$.
   Show $T(n) \leq cn \lg n$ for some constant $c > 0$.
   (Note: $\lg n$ is monotonically increasing for $n > 0$)
2. $T(n) = 4T\left(\frac{n}{3}\right) + n$. Our guess: $T(n) = O(n^{\log_3 4})$.
   Show $T(n) \leq cn^{\log_3 4}$ for some constant $c > 0$.
3. $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{8}\right) + n$. Our guess: $T(n) = O(n)$.
   Show $T(n) \leq cn$ for some constant $c > 0$.
4. $T(n) = 4T(\frac{n}{2}) + n^2$. Our guess: $T(n) = O(n^2)$.
   Show $T(n) \leq cn^2$ for some constant $c > 0$.

Part 2: Radix-sort on strings (80 points)

The class random generator has been updated (*random_generator.h* and *random generator.cpp*) by a member function which generates random strings of up to a given length with characters between "a" and "z".

- *char* random_string(int max, int& m)*
  The function picks a string length *m* at random in between 1 and *max* (*1<=m<= max*). The length *m* of the random string is stored in the second parameter. The function then allocates *m + 1* characters. The first *m* characters (*0……m-1*) are chosen at random in between the characters "a" and "z". The *m*-th character is set to 0 in order to mark the end of the string.

A string in C is represented by an array of characters (e.g. *char *st*). Each character is an ASCII representation of a specific integer value. The character "a" is a representation of the value 97 according to the ASCII table. The individual characters of the array/string can therefore be viewed as a digit.

1. Implement an insertion sort algorithm for strings that sorts a given array of strings according to the character at position *d*. It is necessary to include the length of each string (*array A len*) as it is unclear whether or not the digit *d* exists. A non-existing digit *d* is interpreted as a 0 in the sorting process. Add a parameter *int d* and *int* A_len* to the algorithm arguments and modify the given insertion sort algorithm accordingly. This algorithm should be implemented in the method below:

```
void insertion_sort_digit(char** A, int* A_len, int l, int r, int d)
```

Notes:

- Do not copy the characters itself, but only the pointer to the string/array of characters instead.
- Do not fill the strings with 0's to make them of equal size.
- Do not compute the length of a string with *strlen*. Use the int array *A_len* instead.
- If you move a string within the array of strings then you must update the array containing the length of the strings accordingly.

2. Use this modified insertion sort algorithm to implement radix sort for an array of strings. Measure the runtime performance for arrays of random strings 10 times for every combination of array size n = 10000; 25000; 50000; 75000; 100000 and length of the random strings m = 25; 50; 75. Discuss your results. (You might have to adjust the value for *n* dependent on your computers speed, but allow each test to take up to a couple of minutes). This algorithm should be implemented in the method below:
   ```
   void radix_sort_is(char** A, int* A_len, int n, int m)
   ```

3. Develop a counting sort algorithm for strings that sorts a given array of strings according to the character at position *d*. As for the insertion sort on digit *d*, a non-existing digit *d* is treated as a 0 throughout the counting sort. This algorithm should be implemented in the method below:
   ```
   void counting_sort_digit(char** A, int* A_len, char** B, int* B_len, int n, int d)
   ```

Notes:

- Do not copy the characters itself, but only the pointer to the string/array of characters, instead.
- Do not fill the strings with 0's to make them of equal size.
- Do not compute the length of a string with *strlen*. Use the int array *A_len* instead.
- When moving the string into its correct position you also have to move the length of the string into its correct position.
- You can use *int C[256]* to store the counters/positions.

4. Use this new counting sort algorithm to implement radix sort for an array of strings. Measure the runtime performance for arrays of random strings 10 times for every combination of array size n = 1E+5; 2.5E+5; 5E+5; 7.5E+5; 1E+6, 2.5E+6, 5E+6 and length of the random strings m = 50; 70; 90,100. Discuss your results. (You might have to adjust the value for *n* dependent on your computers speed, but allow each test to take up to a couple of minutes). This algorithm should be implemented in the method below:
   ```
   void radix_sort_cs(char** A, int* A_len, int n, int m)
   ```

Remarks:

- You are not allowed to use code from online resources. Your submission will be tested against that, and will receive a 0, and a report to the Graduate Academic Integrity Board if it is detected.
- Your report has to be typed, and submitted in a pdf file. You should provide figures that plot the running times in relation to the input size parameters.
- You might have to adjust the value for n depending on your computers speed, but allow each test to take up to a couple of minutes.

- Start with smaller values of n and m and stop if one instance of the algorithm takes more than 5 min to complete (the insertion sort implementations will hit that limit early on).
- Discuss your results means that in your report you have to present the results in a table, and analyze and interpret your findings properly. What do the experiments tell you?
- The programming, testing and the experimentation will take some time. Start early.
- Feel free to use the provided source code for your implementation. You have to document your code.
- A Makefile is provided to build the code in the Virtual Box provided.
- Your code has to compile, and run at the Virtual box shared with you in order to be graded.
- Your methods should be added in the sort.h/sort.cpp files. You must keep the same file structure, makefile that is provided in the skeleton code.