Programming Assignment 6
Searching / Sorting

Due Date : Wednesday November 20th , 2019
No Later Than @ 12:45 pm

Problem:

Write a C++ program that does the following :

Accepts a **positive *integer*** (***n***) from the keyboard . Create an character array of size
n. Using a random number generator, populate the array with characters between 33 – 126. Create 7
individual functions and perform the following

1. In the first function: display elements of the array. Display
   the first 20 elements If the size is > 20

2. In the second function : Using recursion, Search for a char (80
) in the array using **sequential search** and at the end display
   number of comparisons it makes.

3. In the third function : Sort the original array using **selection**
   Sort and at the end display the number of swaps it makes.
   Display elements of the array.

4. In the fourth function : Sort the original array using **insertion**
   Sort and at the end display the number of comparisons it makes.

5. In the fifth function : Sort the original array using **Quick** Sort
   and at the end display the number of recursion calls it makes.
   Use the next to the middle value as a pivot value.

6. In the sixth function : Sort the original array using **Quick** Sort
   and at the end display the number of recursion calls it makes.
   Use the first value as a pivot value. Display elements of the
   array.

7. In the last function : Sort the original array using **Quick** Sort
   and at the end display the number of recursion calls it makes.
   Use the last value as a pivot value. Display elements of the
   array.

8. For each of the preceding steps (2 thru 7), calculate and print
   the CPU time before each step starts and after each completed
   step then calculate actual time for the completion of each step.
   Time should be displayed in seconds and milliseconds Display
   elements of the array. Display the first 20 elements If the size
   is > 20

NOTES:

- Just one .cpp file with at least 7 individual functions (prototypes and definitions)
  plus main for testing.
- Do not use classes, structures , templates
- Do not use any global variables / global arrays or vector arrays.
- Validations on the size of the array.
- Do not use any sort library that is available with CodeBlocks or any other IDE.

Style Guidelines:

At the beginning of your program (and **before** the #include statement), include the
following :

Header comments (file documentation block) should be at the top of each file and
should contain: Author / s, Due Date, Assignment Number, Course number and
section, Instructor, and a brief description of the purpose of the code in the file. For
example :

//   Serial Number :   xxxxxxxxx
//
//   Author : (Your name here!!)

//   Due Date :
//   Programming Assignment Number 6
//
//   Fall   2019   -   CS 3358   -   Your Section Number
//
//   Instructor: Husain Gholoom.
//
//   <Brief description of the purpose of the program>

Variable names :

- Must be meaningful.
- The initial letter should be lowercase, following words should be capitalized, no other caps or punctuation (i.e. weightInPounds).
- Each variable must be declared on a separate line with a descriptive comment.

Named constants :

- Use for most numeric literals.
- All capitals with underscores (i.e. TX_STATE_SALES_TAX)
- Should occur at top of function, or global (only if necessary)

Line length of source code should be no longer than 80 characters (no wrapping of lines).

Indentation :

- Use 2-4 spaces (but be consistent throughout your program).
- Indent blocks, within blocks, etc.
- Use blank lines to separate sections.

Comments for variables :

All variable definitions should be commented as follows:

int gender;   // integer value for the gender,
                       // 1 = Male , 2 = Female ,

Rules   : In order to get a full mark   :

1. Your program **must compile** and run. We will test your program using the latest version of Codeblock that is available in our CS labs when testing this program.
2. Your program must be **documented according the style above . See the website for the sample programming style program.**
3. Must **use** at least **7** functions **(prototypes and definitions) for all searching / sorting and other functions of this program.** No menu selections are used.
4. You must use the appropriate libraries in writing this program.
5. Must properly format the output . Sample is provided .
6. You must name your program as :

    o **LastName_FirstName_F19_3358_1_PG6.cpp**

    Where LastName is your Last Name and FirstName is your First Name. For example , the file name should look something like : Gholoom_Husain_F19_3358_1_PG6.cpp (not .cbp)

7. Everyone must upload the electronic version of the program no later than the starting of class time on the due date.   **No late assignments will be accepted.   DO   NOT** send your assignment solution via email.

    Use TRACS To upload your program .

8. You must **also** turn in hard copy of your source code no later than the starting of class time on the due date . Should the hard copy consist of more than one page , then , the hard copy must be **stapled**.   If you are unable to turn in a printout during class, you can take the program to the computer science department and hand it to the front desk personal (Comal 211) before the deadline. Make sure that the front office stamps the program. Make sure that include the date and time. Finally ,make sure that they place the program in my mailbox.

    DO NOT slide your program under my office door − It will **NOT** be accepted

The following points will be deducted if :

- Incorrect file format such as uploading .cbp   instead of .cpp ,
  missing electronic copy , missing the hardcopy , using .h and
  .cpp files , Compilation Errors   **,**   Using global variables, global or
  fixed size arrays , Using global vectors or dynamic arrays, using
  classes or structures   … etc   **(- 10 points)**

- Other **(at least 1.25 point each)**   :

  - Logical Errors
  - Unable to read the source code due to unclear printing
  - Incorrect program file name.
  - Incorrect output format.
  - More than one hardcopy per group or Hard copy is not
    stapled.
  - Incorrect Style **such as but not limited to** Missing title ,
    footer , comments or program documentations , missing
    serial number , missing section number, more than one hard
    copy, missing function prototypes, not replacing my name
    with your name … etc

Sample run:

Searching / Sorting Benchmark

Using a random number generator, we are creating an array of n
elements of type char then performing the following   :

1. Displaying the first 20 numbers.

2. Using **recursion**, Searching for an element in the array using
   sequential search and at the end displaying number of comparisons
   it makes.

3. Sort the original array using **Selection** Sort   and at the end
   display the number of swaps it makes.

4. Sorting the array using **insertion**   Sort   and at the end
   displaying the number of comparisons it makes.

5. Sorting the array using **Quick** Sort and at the end displaying
   the number of recursion calls   it makes. Using the next to the
   middle value as a pivot value.

6. Sorting the same array using **Quick** Sort   and at the end
   displaying the number of recursion calls it makes. Using the
   first value as a pivot value.

7. Sorting the same array using **Quick** Sort   and at the end
   displaying the number of recursion calls it makes. Using the
   last value as a pivot value.

8. For each of the preceding steps (2 thru 7), calculating and
   printing the CPU time before each step starts and after each
   completed step then calculating actual CPU time for the
   completion of each step.

Enter the size of the array :   10

Array elements are : A a }   Q ~ %   5 < 0 }

Sequential Search

Searching for   80

80   Was Not found.
Start Time   :
End Time   :
Actual CPU Clock time :
Total Number of Comparisons :
Array Elements :

Selection Sort   :

Start Time   :   xxxxxxxxx
End Time   :   xxxxxxxxx
Actual CPU Clock time :   xxxxxx
Total Number of Swaps   :   xxxxxx
Sorted Elements :

Insertion Sort   :

Start Time   :   xxxxxxxxx
End Time   :   xxxxxxxxx
Actual CPU Clock time :   xxxxxx
Total Number of Comparisons   :   xxxxxx
Sorted Elements :

Quick Sort - Next to the middle element as a pivot :

Start Time   :   xxxxxxxxx
End Time   :   xxxxxxxxx
Actual CPU Clock time :   xxxxxx
Total Number of Recursive Calls :   xxxxxxxxx
Sorted Elements :

Quick Sort - First element as a pivot :

Start Time   :   xxxxxxxxx
End Time   :   xxxxxxxxx
Actual CPU Clock time :   xxxxxx
Total Number of Recursive Calls :   xxxxxxxxx
Sorted Elements :

Quick Sort - last element   as a pivot :

Start Time   :   xxxxxxxxx
End Time   :   xxxxxxxxx
Actual CPU Clock time :   xxxxxx
Total Number of Recursive Calls :   xxxxxxxxx
Sorted Elements :

11- 20 - 2019
Benchmark Algorithm Implemented by :
First Name , Last Name

Sample run:

Searching / Sorting Benchmark

Using a random number generator, we are creating an array of n elements of type char then performing the following :

1. Displaying the first 20 characters.

2. Using **recursion**, Searching for an element in the array using **sequential search** and at the end displaying number of comparisons it makes.

3. Sort the original array using **selection** Sort and at the end display the number of swaps it makes.

4. Sorting the array using **insertion** Sort and at the end displaying the number of comparisons it makes.

5. Sorting the array using **Quick** Sort and at the end displaying the number of recursion calls it makes. Using the next to the middle value as a pivot value.

6. Sorting the same array using **Quick** Sort and at the end displaying the number of recursion calls it makes. Using the first value as a pivot value.

7. Sorting the same array using **Quick** Sort and at the end displaying the number of recursion calls it makes. Using the last value as a pivot value.

8. For each of the preceding steps (2 thru 7), calculating and printing the CPU time before each step starts and after each completed step then calculating actual CPU time for the completion of each step.

Enter the size of the array : a

*** Error - Invalid input - Size must be > 0 ***

11- 20 - 2019
Benchmark Algorithm Implemented by :
First Name , Last Name

Sample run:

Searching / Sorting Benchmark

Using a random number generator, we are creating an array of n
elements of type char then performing the following   :

1. Displaying the first 20 numbers.

2. Using **recursion**, Searching for an element in the array using
   sequential search and at the end displaying number of comparisons
   it makes.

3. Sort the original array using **selection** Sort and at the end
   display the number of swaps it makes.

4. Sorting the array using **insertion** Sort and at the end
   displaying the number of comparisons it makes.

5. Sorting the array using **Quick** Sort and at the end displaying
   the number of recursion calls it makes. Using the next to the
   middle value as a pivot value.

6. Sorting the same array using **Quick** Sort and at the end
   displaying the number of recursion calls it makes. Using the
   first value as a pivot value.

7. Sorting the same array using **Quick** Sort and at the end
   displaying the number of recursion calls it makes. Using the
   last value as a pivot value.

8. For each of the preceding steps (2 thru 7), calculating and
   printing the CPU time before each step starts and after each
   completed step then calculating actual CPU time for the
   completion of each step.

Enter the size of the array :   -1
 ***   Error - Invalid input - Size must be > 0   ***

11- 20 - 2019
Benchmark Algorithm Implemented by :
First Name , Last Name