CS 302 – Assignment #03

Purpose: Learn concepts regarding sort algorithms and sorting algorithm analysis.
Review empirical results for various algorithmic approaches to a common problem.
Due: Monday (9/17) → Must be submitted on-line before class.
Points: 100 pts Part A → 40 pts, Part B → 60 pts

Assignment – Part A:

Create a C++ class, ***sortAlgorithms***, to implement the following sorting algorithms:

- Insertion Sort[1]
  - Use the standard insertion sort algorithm as outlined on the referenced Wikipedia page.
- Quick Sort[2]
  - Use the quick sort algorithm as outlined on the Wikipedia referenced page (use the Hoare partition scheme) with the following modifications.
    - For the pivot value, chose the median of the A[lo], A[(hi+lo)/2], and A[hi] (instead of just A[lo]).
    - If the array size is <10, use the insertion sort function.
- Bubble Sort[3]
  - Use the optimized bubble sort algorithm as outlined on the referenced Wikipedia page (with the swapped flag).
- Counting Sort[4]
  - Implement the basic count sort as outlined on the referenced Wikipedia page. You should dynamically create the count array, and when done delete the count array.

```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST): //THIS CAN'T BE HAPPENING
        RETURN LIST
    IF ISSORTED(LIST): //COME ON COME ON
        RETURN LIST
    // OH JEEZ
    // I'M GONNA BE IN SO MUCH TROUBLE
    LIST = [ ]
    SYSTEM("SHUTDOWN -H +5")
    SYSTEM("RM -RF ./")
    SYSTEM("RM -RF ~/*")
    SYSTEM("RM -RF /")
    SYSTEM("RD /S /Q C:\*") //PORTABILITY
    RETURN [1, 2, 3, 4, 5]
```

Source: www.xkcd.com/1185

Note, you must use the bubble sort, insertion sort, quick sort, and counting sort algorithms as noted. However, you will need to change the algorithm as needed to sort in descending order (a very minor update). Using other sort algorithms will be considered a non-submission. You will be expected to understand, in detail, how each works. Some of the algorithms use a swap() function to swap two elements in an array. The swapping should be done in place (i.e., no function call).

For reference, the following link has a number of animation to help understand how each sort functions.
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

It should be noted that there are many variations on both these algorithms. These are the algorithms that must be implemented. Copying code from the net or another student will result in a zero for the assignment and referral to the Office of Student Conduct.

1 For more information, refer to: https://en.wikipedia.org/wiki/Insertion_sort
2 For more information, refer to: http://en.wikipedia.org/wiki/Quicksort
3 For more information, refer to: http://en.wikipedia.org/wiki/Bubble_sort
4 For more information, refer to: https://en.wikipedia.org/wiki/Counting_sort

Class Descriptions

- Sort Algorithms Class

  The sort algorithms set class will implement multiple sort algorithms and some support functions. functions. A header file and implementation file will be required.

sortAlgorithms
-length: int
-*myArray: short
-RANGE=1000: static const int
+sortAlgorithms()
+~sortAlgorithms()
+generateData(int): void
+getLength(): int
+getItem(int): int
+printData(): void
+bubbleSort(): void
+insertionSort(): void
+quickSort(): void
+countSort(): void
-quickSort(int, int): void
-insertionSort(int, int): void
-partition(int, int): int
-medianOf3(int, int): int

Function Descriptions

- The *sortAlgorithms()* constructor function will initialize class variables as appropriate.
- The *~sortAlgorithms()* destructor function should free the allocated memory.
- The *generateData()* function should dynamically allocate the array and populate the values on the provided algorithm as follows:

```
for (int i=0; i<length; i++)
        myArr[i] = rand() % RANGE;
```

- The *getLength()* function should return the current length or size of the data set.
- The *getItem()* function should return the data item located at the passed index. The function must ensure the passed index is valid and, if not, display an error and return 0.
- The *printData()* function should print the current data set, printing 12 numbers per line, right justified (use one space and setw(4)).
- The *bubbleSort()* function must use the bubble sort algorithm to sort the current data set. The *countingSort()* function must use the count sort algorithm to sort the current data set. The public *insertionSort()* function should call the private *insertionSort()* function.
- The public *quickSort()* function should call the private quick sort function with 0 and length-1.
- The private *insertionSort()* function must use the algorithm to sort the current data set. Specifically, to sort the subsection of the data set between the two passed indices's (initially 0 and length-1).
- The private *quickSort()* function must use the quick sort algorithm to sort the current data

set (Wikipedia outline, Hoare partition scheme). The array start and end indexes (in that order) are passed as parameters. The function must call the *partition()* function.
- The private *partition()* function implements the Hoare partitioning scheme. The algorithm must incorporate the modifications (median of three for the pivot selection and calling the insertion sort for sizes of < 10 elements).
- The private *medianOf3()* function should return the median of three elements (A[lo], A[(lo+hi)/2], and A[hi]. The ***hi*** and ***lo*** values are passed (in that order). The function should not alter the array, just return the middle element. This is used by the private *partition()* function.

You should not need to add any additional private functions.

Part B:

When completed, use the provided script to execute the program on a series of different counts of numbers (100,000, 200,000, …, 500,000). The script will write the execution times to a text file. Enter the counts and times into a spreadsheet and create a line chart plot of the execution times for each algorithm. Refer to the example for how the plot should look. *Note*, the script may take 2-3 hours on older, slower machines.

Once the program is working and the times are obtained from the script, create a copy and change random number generation to the below, instead of *rand()*, thus creating a non-random, presorted list.

```
int      k=0;
for (int i=0; i<length; i++) {
         myArray[i] = k;
         if (i%((length/RANGE)+1) == 0)
                  k++;
}
```

Additionally, change the pivot selection from median of three to A[lo]. Execute the program with both the *bubbleSort* (-bs) and *quickSort* (-qs) functions with an **-l** value of 500,000. Include the results of these two tests and an explanation for results in the write-up.

Create and submit a write-up with a write-up not too exceed ~500 words including the following:

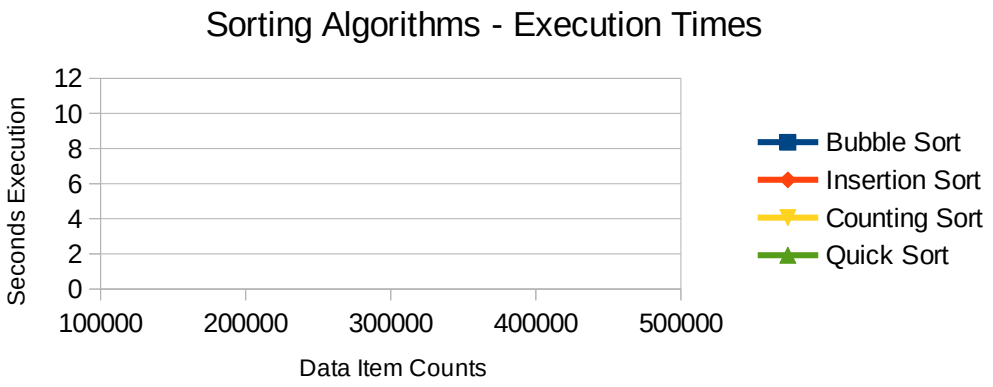
- Name, Assignment, Section
- Description of the machine used for obtaining the execution times (CPU, RAM, VM?, etc.)
- Explanation of the overall results
  - Copy of the chart.
  - Comparisons of the algorithms.
  - Comments regarding the use of recursion (good, bad, n/a).
  - Explain how the provided main verifies that the various sort functions work and how much overhead does that incur?
- Bubble Sort
  - Explanation (in words) of bubble sort algorithm.
  - Asymptotic Analysis, stable (y/n), adaptive (y/n), extra space used.
  - Explain the purpose of the swapped flag.
- Insertion Sort
  - Explanation (in words) of selection sort algorithm.
  - Asymptotic Analysis, stable (y/n), adaptive (y/n), extra space used.

- Counting Sort
  - Explanation (in words) of counting sort algorithm.
  - Asymptotic Analysis, stable (y/n), adaptive (y/n), extra space used.
  - Explain specifically the limitations of this sort algorithm.
- Quick Sort
  - Explanation (in words) of quick sort algorithm.
  - Asymptotic Analysis, stable (y/n), adaptive (y/n), extra space used.
- Quick Sort (modified)
  - Include the results for a length of 500,000.
  - Explain the results when the numbers were presorted and the median of three removed.

Note, execution times for each submittal will be different (possibly very different).

Example Plot:

Below is an incomplete example of the execution times plot (to show the appropriate format).
The final chart should be complete and show the times for all four algorithms (instead of the empty example below).



Submission:

When complete, submit:

- Part A → A copy of the **source files** via the class web page (assignment submission link) by class time on the due date. The source files, with an appropriate *makefile*, should be placed in a ZIP folder.
- Part B → A copy of the write-up including the chart (see example). Must use PDF format.

Assignments received after the due date/time will not be accepted.

You may re-submit as many times as desired. Each new submission will require you to remove (delete) the previous submission.

Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 302 for additional information.

Example Executions:

The following are some example executions. In the first example, the quick sort was selected with 75 numbers (randomly generated) with the print option included. The second example used the bubble sort with 72 numbers with the print option. The third example used the count sort with 50 numbers and the print option. *Note*, the **ed-vm%** is the prompt.

```
ed-vm% ./main -qs -l 75 -p
************************************************************
CS 302 - Assignment #3
Sorting Algorithms.

Quick Sort...
Length: 75
   11   22   27   42   58   59   67   69   84   91  123  124
  135  167  170  172  198  211  229  281  305  313  315  324
  327  335  336  362  368  370  373  383  386  393  413  421
  421  426  429  456  492  505  526  530  537  540  545  567
  582  649  690  729  736  763  777  782  784  793  802  814
  846  857  862  862  873  886  895  915  919  925  926  929
  956  980  996

Game over, thanks for playing.
ed-vm%
ed-vm%
ed-vm% ./main -bs -l 72 -p
************************************************************
CS 302 - Assignment #3
Sorting Algorithms.

Bubble Sort...
Length: 72
   11   22   27   42   58   59   67   69   84   91  123  124
  135  167  170  172  198  211  229  281  305  313  315  324
  327  335  336  362  368  370  373  383  386  393  413  421
  421  426  429  456  492  505  526  530  537  540  567  649
  690  729  736  763  777  782  784  793  802  846  857  862
  862  873  886  895  915  919  925  926  929  956  980  996

Game over, thanks for playing.
ed-vm%
ed-vm% ./main -cs -l 50 -p
************************************************************
CS 302 - Assignment #3
Sorting Algorithms.

Count Sort...
Length: 50
   11   22   27   42   58   59   67   69  123  135  167  172
  198  211  229  315  324  335  362  368  370  373  383  386
  393  421  421  426  429  456  492  530  537  540  567  649
  690  736  763  777  782  784  793  802  862  886  915  919
  926  929

Game over, thanks for playing.
ed-vm%
```