CS 302
Homework, Asst. #03

Purpose: Learn concepts regarding sort algorithms and sorting algorithm analysis.
Review empirical results for various algorithmic approaches to a common problem.
Due: Tuesday (9/19) → 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:

```
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]
```

- 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 Wikipedai referenced page (Hoare partition scheme).
- 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.

Note, you must use the bubble sort, insertion sort, quick sort, and counting sort algorithms as noted. Using other sort algorithms will be considered a non-submission. You will be expected to understand, in detail, how each works.

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 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
-LIMIT=5000: static const int
+sortAlgorithms()
+~sortAlgorithms()
+generateData(int): void
+getLength(): int
+getItem(int): short
+printData(): void
+bubbleSort(): void
+insertionSort(): void
+quickSort(): void
+countSort(): void
-quickSort(int, int): void
-partition(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 based and populate the values on the provided algorithm as follows:

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

- The *getLength()* function should return the current length or size of the data set.
- The *getItem(int)* 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 10 number per line, right justified (use one space and setw(6)).
- The *bubbleSort()* function must use the bubble sort algorithm to sort the current data set. The basic algorithm should be updated to sort in descending order (large to small).
- The *countingSort()* function must use the count sort algorithm to sort the current data set. The basic algorithm should be updated to sort in descending order (large to small).
- The *insertionSort()* function must use the insertion sort algorithm to sort the current data set.
- The public *quickSort()* function should call the private quick sort function with 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 should call the partition() function.

- The private *partition()* function implements the Hoare partitioning scheme. The basic algorithm should be updated to sort in descending order (large to small).

You should not need 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, …, and, 1,000,000). 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=LIMIT;
for (int i=0; i<length; i++) {
          myArray[i] = k;
          if (i%((length/LIMIT)+1) == 0)
                    k--;
}
```

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.

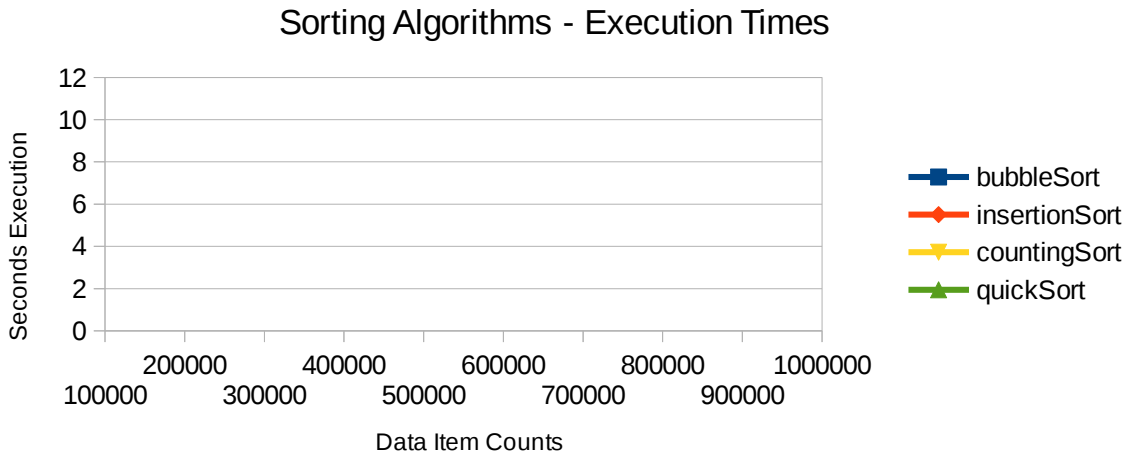
Create and submit a write-up with a write-up not too exceed ~500 words including the following:

- 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).
- Bubble Sort
  - Explanation (in words) of bubble sort algorithm.
  - Asymptotic Analysis.
  - Explain the purpose of the swapped flag.
- Insertion Sort
  - Explanation (in words) of insertion sort algorithm.
  - Asymptotic Analysis.
- Counting Sort
  - Explanation (in words) of counting sort algorithm.
  - Asymptotic Analysis.
  - Explain specifically the limitations of this sort algorithm.
- Quick Sort
  - Explanation (in words) of quick sort algorithm.
  - Asymptotic Analysis.
- Quick Sort (modified)
  - Include the results for a length of 500,000.

- Explain the results when the numbers were pre-sorted.

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 incomplete example above).

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 bubble sort was selected with 100 numbers (randomly generated) with the print option included. The second example used the count sort with 100 numbers and no print. The third example used the quick sort with 150 numbers and the print option. *Note*, the **ed-vm%** is the prompt.

```
ed-vm% ./main -bs -l 100 -p
************************************************************
CS 302 - Assignment #3
Sorting Algorithms.

Bubble Sort...
   4956   4932   4919   4802   4676   4582   4421   4383   4370   4324
   4172   4170   4067   4043   4022   3980   3929   3926   3895   3814
   3784   3750   3690   3586   3584   3537   3526   3456   3426   3368
   3367   3335   3315   3167   3135   3094   3069   3058   3042   2862
   2793   2777   2763   2754   2739   2651   2567   2539   2399   2373
   2362   2305   2281   2276   2178   2084   1996   1915   1873   1862
   1808   1729   1649   1530   1505   1429   1421   1413   1393   1327
   1313   1229   1226   1124   1091   1087    925    886    857    846
    788    782    736    545    540    492    434    403    386    368
    364    336    211    198    123     60     59     27     12     11


Game over, thanks for playing.
ed-vm%
ed-vm% ./main -cs -l 100
************************************************************
CS 302 - Assignment #3
Sorting Algorithms.

Count Sort...

Game over, thanks for playing.

ed-vm%
ed-vm% ./main -qs -l 100 -p
************************************************************
CS 302 - Assignment #3
Sorting Algorithms.

Quick Sort...
   4956   4932   4919   4802   4676   4582   4421   4383   4370   4324
   4172   4170   4067   4043   4022   3980   3929   3926   3895   3814
   3784   3750   3690   3586   3584   3537   3526   3456   3426   3368
   3367   3335   3315   3167   3135   3094   3069   3058   3042   2862
   2793   2777   2763   2754   2739   2651   2567   2539   2399   2373
   2362   2305   2281   2276   2178   2084   1996   1915   1873   1862
   1808   1729   1649   1530   1505   1429   1421   1413   1393   1327
   1313   1229   1226   1124   1091   1087    925    886    857    846
    788    782    736    545    540    492    434    403    386    368
    364    336    211    198    123     60     59     27     12     11


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