CHOOSE **ONE** OF THE FOLLOWING PROJECTS. ANSWER ALL QUESTIONS REGARDING THE RESULTS OF YOUR PROGRAMS.

1. **Comparisons of Clock Time for Sorting Algorithms (large arrays):**

a) generate an array of 10,000 random integers, with values between 1 and 1,000,000.
b) "Sort" the array using:
InsertionSort
MergeSort
QuickSort (basic version of Partition)
making sure to use the **original random** array for each algorithm.
c) insert a timer into each of the sorting algorithms to measure the exact clock time to run each algorithm, and keep an "accumulator" for each algorithm. (System.currentTimeMillis() will show the differences.)
d) RUN the sorting algorithms a total of 100 times each, and display the **average clock time** used by each of the algorithms.

Answer the following questions:
How do the results of the clock time compare to the expected results based on the known runtimes for these algorithm?

Did you encounter any problems when running the algorithms?

Run the same programs, but increase the array size to 100,000. How did the results compare to the original arrays of size 10,000?

For one "run" only, run QuickSort and MergeSort on the ***sorted array from InsertionSort.*** *Did you encounter any problems?*

2. **Comparisons of Clock Time for Sorting Algorithms (small arrays)**:

a) generate an arrays of random integers, with values between 1 and 1000, with sizes 16, 32, 64, 128, and 256.

b) "Sort" the arrays using:

InsertionSort

MergeSort (basic version)

QuickSort (basic version of Partition)

making sure to use the **original random** array for each algorithm.

c) insert a timer into each of the sorting algorithms to measure the exact clock time (ms) to run each algorithm, and keep an "accumulator" for each algorithm.

d) RUN the sorting algorithms a total of 100 times each, and display the **average clock time** used by each of the algorithms. (You will need to use System.nanoTime() to see the differences!)

At what array size is there a change in the "best" clock time for the algorithms?

3. Heaps:

a) Trace through, by hand, the process to "BuildHeap" using the "BuildHeap" algorithm, using an array of 15 random integers between 1 and 100. **Draw your heaps as trees!**

b) Trace through, by hand, the process the "build a heap" by repeatedly inserting 15 random integers (between 1 and 100) into an initially empty heap. **Draw your heaps as trees!**

c) *Discuss the number of "swaps" ("bubbles") used by both (a) and (b).*
Does this match the expected "runtimes"?
Are heaps unique?

d) Write a program that will generate an array of 100 random integers between 1 and 1000 and then *build a heap* using the "BuildHeap" method and the "BuildByInsert" method. Include a timer for each to measure the actual clock time for each method.