Project 5. February 12th: 11:59pm.   100 points.

This project has three parts: theory, suspense :), C part and an optional extra credit.

Point distribution:

Theoretical (25 points)

- Problem 1: 5 points
- Problem 2: 6 points
- Problem 3: 4 points
- Problem 4: 10 points

Suspense (Sort Detective) (60 points) (Order below is random)

- Bubble: 10 points
- MergeSort: 10 points
- QuickSort: 10 points
- CheckSort: 10 points
- Insertion: 10 points
- Selection: 10 points

C problem (10 points)

- Find maximum: 15 points

Extra credit problems (8 points)

- Implement an insertion sort that sorts numbers in the stack.

Files to submit:

- HW5_1.pdf
- HW5_2.txt
- HW5_C.c
- InsertionSort.java (optional Extra credit part)

Part 1

Create **HW5_1.pdf** file and write all your answers there

Problem 1. For a given array A = (1, 5, 3, 2, 0, 4, 6, 8) and a partition method **below**, show the resulting array after making a call to Partition(A, 0, 7). Show each iteration step in detail in a table of the following format:

j	Action	Array contents

```
PARTITION($A, p, r$)
$x = A[r]$
$i = p - 1$
for $j = p$ to $r - 1$
    if $A[j] \le x$
        $i = i + 1$
        exchange $A[i]$ with $A[j]$
exchange $A[i + 1]$ with $A[r]$
return $i + 1$
```

Problem 2. You need to sort numbers {5, 7, 1, 9, 45, 23, 7} in DECREASING order. Show the resulting array after each step of the algorithm (each outer loop) in a neat table format using:

a. Selection sort
b. Insertion sort
c. Bubble sort

Problem 3. Sorting question

Give an **efficient** O(n) algorithm to rearrange an array of *n* keys so that all the negative keys precede all the nonnegative keys. Your algorithm must be in-place, meaning you cannot allocate another array to temporarily hold the items.

Problem 4. For some problems, it is convenient to have data in already sorted order. For each of the questions below explain whether sorting the list helps to solve a problem faster. If yes, explain how sorting helps performance in terms of running time. If not, describe a faster alternative algorithm without sorting along with its runtime complexity. (Note that you must take into consideration the runtime for sorting the list first)

1. Finding the n*th* smallest number in a list
2. Computing the mode of a set of integers (e.g., the value that occurs most often).
3. Computing the median of two arrays. (let me rephrase: You have some numbers in both arrays, you need to find a median for all the numbers combined).
4. Checking whether a particular item exists more than once in the array.
5. Find the minimum value in an array.

Part 2. Sort Detective

(Developed from http://nifty.stanford.edu/2002/LevineSort/labWriteUp.htm and Prof. Alvarado's homework)

You will be given a program (download from TED) which is designed to measure comparisons, data movements, and execution time for the six sorting algorithms discussed in class and in your reading assignments:

- Bubble sort
- Check Sort
- Insertion Sort
- Selection Sort
- Merge Sort
- QuickSort

Slight problem: The buttons are not labeled properly. You must apply your understanding of the general properties of the algorithms (and in some cases, of the code used to implement them) to determine the proper labeling of the buttons.

Files for this assignment are provided:
- hw5.jar
- SortingAlgorithms.java

File To Submit
- HW5_2.txt

As you know from class, if you double the size of the input data that you give to a quadratic algorithm, it will do four times the work; by contrast, an O(nlogn) algorithm will do a bit more than twice as much; and, a linear algorithm will do only twice as much work. As you also know, the properties of the input data set can affect the expected performance of many of our sorting algorithms. Before you begin the homework, you should review the expected performance of the algorithms on various data sets.

How to start

First, calculate your *Screte* (secret) number. It is the last ***6 numeric DIGITS of your student ID.***

Examples:

Student ID	Screte Number
A123456789	456789
FG78367052BA	367052

Execute hw5.jar (to execute the program type 'java -jar hw5.jar' at the command line) and enter your *Screte* number in the upper right and start playing with the sorting algorithms a bit. **And we will grade based on your *Screte* number as well**. Notice that the button names do not give any indication which sort they will execute. Create a list with some size and property of your choice by clicking "Create The List".

Warning:

Checksort is O(n!) time complexity. So if your input size is large it will make your program freeze for a long long time. Please be alert and do not think that your computer is dead. Enter smaller numbers manually in the box to find the check sort. Also be sure to click "Create the List" to change the size or order of the list.

Other notes:

- Make sure you click on each button to see the number of comparisons, movements, and running time of each execution.
- Describe the results of your experiment in a summary document. Begin with your guess of each button and matching algorithm and then show the rationalization process that justifies it. Here is a sample of explaining a linear search algorithm for the hw5.txt report. You need to provide a good reasoning to support your guess.
- You can use SortingAlgorithms.java for reference and check how a number of comparisons, number of movements and execution time were generated.

Grading

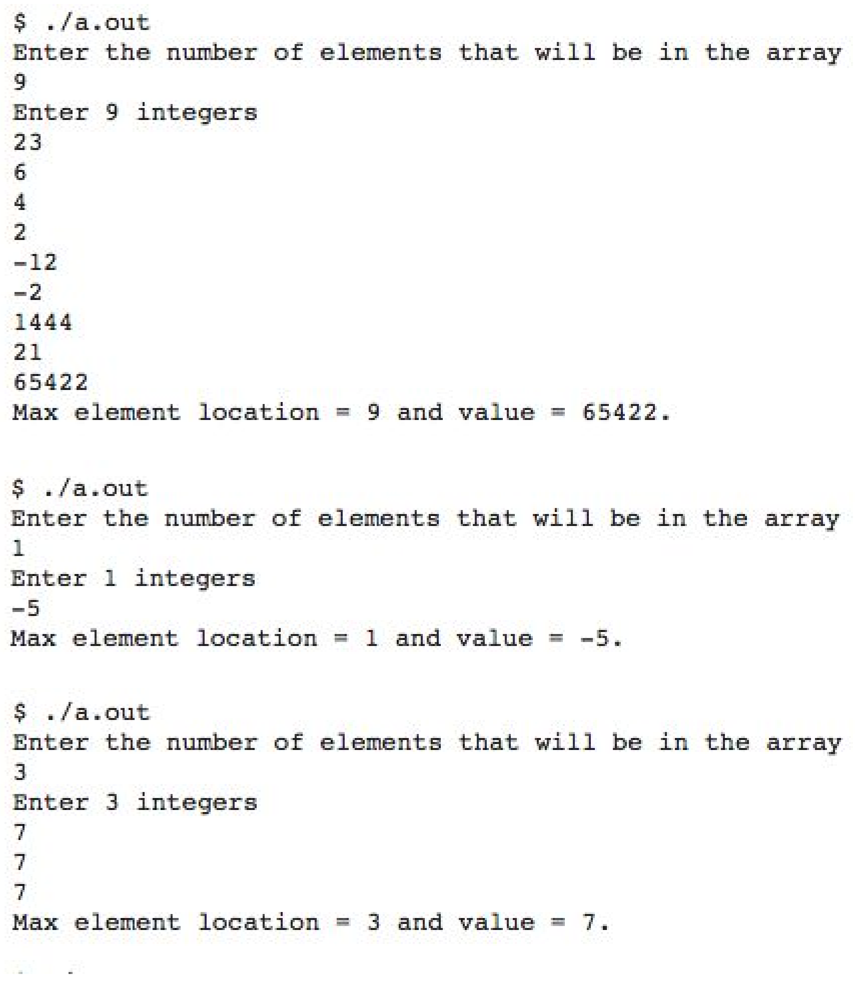
Your grade on this assignment will be a combination of both whether you correctly identified the sorts *and* how well you explained why each secret sort was a particular sorting algorithm.

Part 3. C problem: Maximum element in the array.

Suggested reading (you can use any other source).

- Scanf, printf: http://www.cprogramming.com/tutorial/c/lesson1.html
- arrays in C: http://www.cprogramming.com/tutorial/c/lesson8.html
- loops: http://www.cprogramming.com/tutorial/c/lesson3.html

Your code should find the maximum element present in an array. It also prints the location (index + 1) at which the maximum element occurs in the array. Sample run is shown below: Sample outputs are shown below:



```
$ ./a.out
Enter the number of elements that will be in the array
9
Enter 9 integers
23
6
4
2
-12
-2
1444
21
65422
Max element location = 9 and value = 65422.

$ ./a.out
Enter the number of elements that will be in the array
1
Enter 1 integers
-5
Max element location = 1 and value = -5.

$ ./a.out
Enter the number of elements that will be in the array
3
Enter 3 integers
7
7
7
Max element location = 3 and value = 7.
```

Do not change any print statement in the starter code except for the “??” mark at line 20.
You can assume an intelligent user:

- No negative number as the length of the array.
- Number of elements are always integers.
- Also, if there are duplicates, return the last occurrence.

What to submit:

- HW5_C.c

Submission

Files to submit:

- HW5_1.pdf
- HW5_2.txt
- HW5_C.c
- InsertionSort.java (optional Extra credit part)

Remember:

  - All relevant files must have the same file names as in “Files to submit”
  - Make sure you hit **submit** button to submit your code
  - **Do not forget to check submission.txt report for correctness**
  - If submission.txt says you will get a zero and you don’t change your files, you *will* get a zero and cannot ask for a regrade
  - When submitting, the script will check for your PID in HW5_C.c, **please make sure the PID it grabs is correct**

Extra Credit Problem

Write an **Insertion Sort** algorithm for integer key values. However, here’s the catch: The input is a **stack** (not an array), and the only variables that your algorithm may use are a fixed number of integers and a fixed number of stacks. The algorithm should return a stack containing the records in sorted order (with the smallest value being at the top of the stack). Your algorithm should be $\Theta(n^2)$ in the worst case.

You either can use your own stack implementation from HW4 or use a built-in stack.