Computer Science AP
Sorting Algorithms

Introduction
This section is about sorting data! Sorting numbers, names, scores, etc. can be a useful task in your programs. You can sort just about any data type. Many classes already have sorting methods built into them that make this task very easy for you in your programs – for example, the Collections class has a method call *sort* that you can use to quickly sort items stored in Lists (like ArrayList).

We are going to take a look at learning how to code a few sorting algorithms. The algorithms we will look at are called bubble sort, selection sort, insert sort. Later in the course we will look at merge sort. We will learn the algorithms, analyze their run times on various size lists, and get an appreciation for how the same problem (sorting!) can be solved in various ways. Ultimately it is a good beginner programmers introduction to reading/writing code and getting you thinking about algorithms.

We will stick to sorting arrays of numbers while learning and analyzing the algorithms since arrays of integers are easy to work with. The algorithms can, however, be applied to almost any object type that is sortable.

Videos
Many of the videos for this section of the course are made by other educators and hosted on YouTube. I have selected videos that are well done and I saw no point 're-doing' them. If you don't like the videos I've selected, feel free to search YouTube as this is a very popular topic and there are thousands of videos explaining these algorithms.

Sorting Applets
An applet is a program that is written in Java that can be placed on a web page. Here are links to a few applets that will demonstrate the various sorting algorithms. You'll be asked to run the applets during the lessons.

***Some browsers will prevent applets from running depending on your security settings. Rather than fiddle with security settings, it is often easier to try running the applet in a different browser. If you want to use Chrome, you'll probably have to google how to permit applets to run if they are not running for you!

X Sort Lab : http://math.hws.edu/TMCM/java/xSortLab/

SortingAlgorithms.com: http://www.sorting-algorithms.com/

UBC Sorting Algorithms: http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html

Section 01 [Bubble Sort]

This sorting method is almost every students first sorting algorithm.

Watch
Java: BubbleSort by Joe James @ https://www.youtube.com/watch?v=F13_wsHDIG4

Run
BubbleSort in the X Sort Lab, or in one of the other applets. Keep running it until you understand how the algorithm works.

Read the Code

```
public void bubbleSort(int[] nums){
      //run bubble sort with nums, reprint after every swap
      for (int i=0; i<nums.length; i++){                          //line A
         for (int j=0; j<nums.length - i - 1; j++){               //line B
            if (nums[j] > nums[j+1]){                             //line C
               int temp = nums[j];                                //line D
               nums[j] = nums[j+1];
               nums[j+1] = temp;              }
         }//end j loop
      }//end i loop
   }
```

Questions

consider the array {10, 8, 6, 4, 2}

1. Be able to describe in words how this sort works.
2. Which two numbers are the first to change positions in the array?
3. Which numbers reach their final position in the array early on in the sort?
4. Which lines of code represent the 'swapping' portion of the sort?
5. Using the bubble sort shown above, how many times would line A, B, C, and D be executed?
6. If an array had 100 numbers that were backward, approximately how many swaps would be required to sort the list?
7. If the array {10,1,2,3,4,5,6,} was used, how many swaps would be required to sort the list?
8. What is the general mathematical relationship (Big O notation) between the time required to sort the list and the number of items in the list?

Section 02 [Selection Sort]

This sorting method involves looking for the smallest value in the array, moving it to the beginning of the list, and then repeating until the list is sorted.

Watch
Java: SelectionSort animated demo with code
by Joe James @ https://www.youtube.com/watch?v=cqh8nQwuKNE

Run
Selection Sort in the X Sort Lab, or in one of the other applets. Keep running it until you understand how the algorithm works.

Read the Code

```
public void selectionSort(int[] nums){
      for (int i=0; i<nums.length-1; i++){
         int posOfLowest = i;                          //line A
         for (int j=i+1; j<nums.length; j++){
            if (nums[j] < nums[posOfLowest])           //line B
               posOfLowest = j;
         }
         int temp = nums[i];                           //line C
         nums[i] = nums[posOfLowest];
         nums[posOfLowest] = temp;
      }
}
```

Questions

consider the array {10, 8, 6, 4, 2}

1. Be able to describe in words how this sort works.
2. Which two numbers are the first to change positions in the array?
3. Which numbers reach their final position in the array early on in the sort?
4. Which lines of code represent the 'swapping' portion of the sort?
5. Using the selection sort shown above, how many times would line A, B, C be executed?
6. If an array had 100 numbers that were backward, approximately how many swaps would be required to sort the list?
7. If an array had 100 numbers with only two numbers out of order, how many swaps would be required to sort the list?
8. Does the order of the values in the original array have a big impact on the running time of the sort? Explain.
9. What is the general mathematical relationship (Big O notation) between the time required to sort the list and the number of items in the list?

Section 03 [Insertion Sort]

This sorting method involves looking for the smallest value in the array, moving it to the beginning of the list, and then repeating until the list is sorted.

Watch
Java: Insertion Sort sorting algorithm
by Joe James @ https://www.youtube.com/watch?v=lCDZ0IprFw4

Run
Selection Sort in the X Sort Lab, or in one of the other applets. Keep running it until you understand how the algorithm works.

Read the Code

```
public void insertionSort(int[] nums){
   for (int i = 1; i < nums.length; i++){
       int j = i;                                          //line A
       int B = nums[i];
       while ( (j > 0) && (nums[j-1] > B) ){
              nums[j] = nums[j-1];                         //line B
              j--;
       }
       nums[j] = B;                                        //line C
   }
}
```

Questions

consider the array {10, 8, 6, 4, 2}

1. Be able to describe in words how this sort works.
2. Which two numbers are the first to change positions in the array?
3. Which end of the array becomes 'sorted' as the sort progresses?
4. This sort does not 'swap' numbers, it 'slides' them. Which is the first number to 'slide'?
5. Which line in the code represents 'sliding' a number?
6. Using the insert sort shown above, how many times would line A, B, C be executed?
7. If an array had 100 numbers that were backward, approximately how many slides would be required to sort the list?
8. If the array {10,1,2,3,4,5,6,} was used, how many slides would be required to sort the list?
9. *Tricky question* Assume you have a completely randomly ordered list of 100 numbers. Predict the number of 'slides' required on average to sort the list.
10. What is the general mathematical relationship (Big O notation) between the time required to sort the list and the number of items in the list?

Section 04 [Timing Lab]

Watch 'Timing Lab'.
Use Xsort and complete the following chart.

Use XSort Lab website!

		Time to Sort			
		n=20000	**n=40000**	**n=60000**	**n=80000**
sort	**bubble**				
	selection				
	insertion				
	***merge**				

For bubble, selection, and insertion sort

1. Which was slowest?
2. Which was fastest?
3. When you double the size of the list, how does the running time change?
4. Do the results support the Big O Notation?

For merge sort

1. How did merge sort compare to the other sorts in terms of time?
2. When the size of the list was doubled, did merge sort behave the same as the other