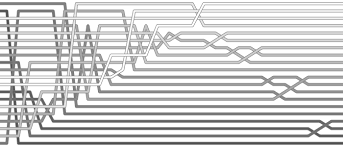
**Searching, Sorting**

We've used searching and sorting algorithms already, most likely a linear search (sequentially search each element in an array until some value is found) and an API method like Arrays.sort(). However, we need to more thoroughly understand the different searches and sorts, not only because they will definitely be on the AP CS exam, but because it will improve us as programmers and prepare you for next-level algorithms.

*Note – this should go without saying, but you should WRITE YOUR OWN CODE for the sorting algorithms. All algorithms are available online, but copying them won't help you on the tests. The algorithms are not easy - they will take some time and effort on your part to understand.*

****The following resources will help:

* [Sorting algorithm animations](https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html)
* Our website (the Searching / Sorting ppts)
* [Youtube](http://www.youtube.com) (there are tons of nice sorting demos)

**Coding the algorithms**

1. Create a new class called SearchSort. SearchSort (initially) has the following:
2. private int[] nums – instance variable, the array of integers we'll be searching / sorting.
3. public SearchSort(int[] nums) – constructor that initializes nums to the value of the parameter. Used for testing with a specific array (e.g. for testing the searches).
4. public SearchSort(int size) – constructor that initializes nums to the size specified. Should invoke (call) the initArray() method. Used for testing efficiency with large arrays and for testing the sorts.
5. public void initArray() – fills the nums array with random values, from 1 – 1000.
6. Write a method private void swap(int i, int j). This method will be useful for many of the sorting routines – it should swap the elements in nums at indexes i and j, so that you don't have to code this common routine for every sorting method.

/\* Making this a method isn't optimal from a performance standpoint, but useful for this lab \*/

1. Write a method public int linearSearch(int key). Your method should return the **index** of key (or -1 if it isn't found).
2. Write two methods that will implement a binary search.  **Arrays must begin sorted to use a binary search – for now, use a call to the static method** Arrays.sort()**.**
   1. Write a method public int binarySearch(int key). Your method should return the **index** of key (or -1 if it isn't found), using a binary search algorithm.
   2. Write a method public int binarySearchRecursive(int key). Your method should return the **index** of key, or -1 if it isn't found, using a **recursive** binary search algorithm. Note – you'll need a private helper method to 'set everything up.'
3. Write a method public void bubbleSort(boolean print). Your method should sort nums using the bubble sort algorithm. The parameter print will be true if your method should print the current state of the array while it's being sorted (to watch the sort in motion).
4. (Riddle) 6 = Z in a M
5. Write a method public void selectionSort(boolean print). Your method should sort nums using the selection sort algorithm. The parameter print will be true if your method should print the current state of the array while it's being sorted.
6. Write a method public void insertionSort(boolean print). Your method should sort nums using the insertion sort algorithm.
7. Write a method public void mergeSort(boolean print). Your method should sort nums using the merge sort algorithm. A private helper method will also be required.

**Note:** You will NOT need to be able to code a merge sort on the AP exam, but you should be able to recognize a method that is performing a merge sort (and know when it will be faster than an iterative sort like selection or insertion). Do your best to code it on your own - if you can't, don't fret; it's a complicated algorithm that requires a solid understanding of recursion.

1. (Riddle) 7 = D in a TN
2. **(Optional)** Write a method public void binaryInsertionSort(boolean print). Your method should sort nums using a slight optimization of the standard insertion sort algorithm called a *binary search* insertion sort. See [here](https://en.wikipedia.org/wiki/Insertion_sort#Variants).
3. **(Advanced)** Write a method public void quickSort(boolean print). Your method should sort nums using the quick sort algorithm. A private helper method will also be required.
4. **(Over 9000)** Write a method public void heapSort(boolean print). Your method should sort nums using the heap sort algorithm. This requires knowledge of binary trees.
5. **(Actual Nonsense)** Sorting algorithms are fundamental to computer science, and have been studied intensely for as long as computer science has been around. However, computer scientists do occasionally have fun with their algorithms. Check out the following novelty sorts:
   1. Write a method public void randomSort(boolean print). Your method should sort nums using the random (a.k.a. bogo) sort algorithm. See [here](https://en.wikipedia.org/wiki/Bogosort).
   2. Write a method public void miracleSort(). Your method should sort nums using the miracle sort algorithm. See [here](http://www.thecshandbook.com/Miracle_Sort). This shouldn't take you long.
   3. Write a method public void assumptionSort(). Your method should sort nums using the assumption sort algorithm. This should take *even less* time to implement than miracle sort! Simply assume the array is sorted and return.