Searches and Sorts

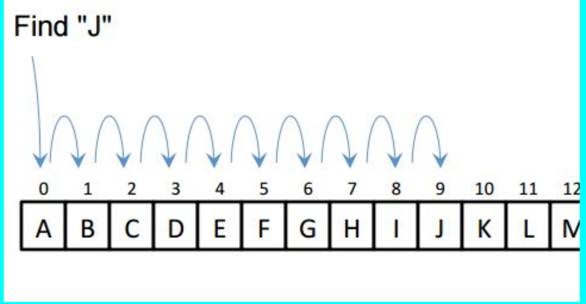
A tutorial by Joshua Rovner

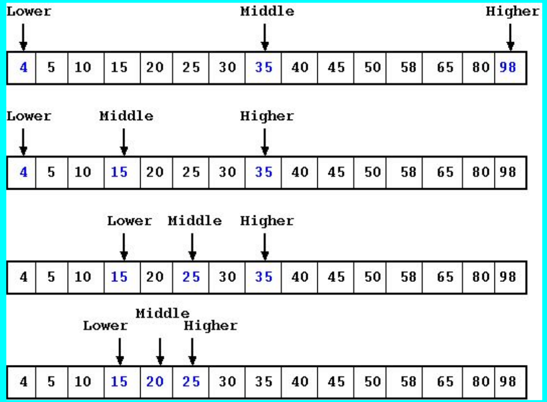
General Knowledge

- ❑ A search, in computer science terms, is an algorithm which goes through a list of data (generally an array or arraylist), and finds a target value.
- ❑ For example, a search could find a certain word in an array of sentences or the largest number in an array of integers.
- ❑ A sort, as in real life, is an algorithm which moves elements and values in a collection of data based on predefined criteria.
- ❑ For example, a sorting algorithm could be made to sort an array of numbers from least to greatest or an array of “Person” objects based on their height attributes.
- ❑ Generally, a sort is performed in order to make data easier to deal with in the future.

Searching Algorithms

- There are dozens of searching algorithms, but for simplicity purposes, only the two in the AP Java subset will be examined:
  - **Sequential Search** - this search goes through every element in the list, trying to find the target value. With large sets of data, this search is very inefficient, as on average, many checks will have to be made until the correct value is found. Average number of comparisons - *n* / 2.
  - **Binary Search** - this search ONLY works with sorted lists, but is efficient in this scenario. It works by dividing the list in half and checking the middle position with the target repeatedly, until the value is found or the whole list has been checked. Average number of comparisons - $\log_2 n$.





Non-Recursive Sorting Algorithms

- ❑ Again, there are dozens of sorting algorithms, but for simplicity purposes only 4 will be covered:
  - ❑ **Selection sort** - this algorithm functions by repeatedly finding the smallest element in a list and placing it at the beginning. An array being sorted with selection sort can be thought as split into two parts: the beginning, with the sorted elements, and the end, with the unsorted ones. Average time complexity - O (n^2).
  - ❑ **Insertion sort** - this algorithm passes through the array and immediately places each element into its correct position by sliding it towards the beginning until it is “inserted” into its correct place. Average time complexity - O (n^2).

Recursive Sorting Algorithms

- **Mergesort** - this algorithm works by splitting a list into halves until they are each of length 1 (essentially creating a list for each element). Then, each adjacent pair of lists is merged into a larger, sorted list, until one sorted list is made. Average time complexity - O (nlogn)
- **Quicksort** - this algorithm works by choosing a “pivot” element and moving all values greater than it to the right, and all the smaller elements to the left. The same thing is done to the left part as well as the right part, and so on, until all the elements have been sorted. Average time complexity - O (nlogn).

When to Use Which Sorts

- ❑ For smaller lists, either use selection or insertion sort. If the list is already partially sorted, selection sort will be more efficient, but with fairly unsorted lists, insertion sort runs in less time.
- ❑ Never use either of these sorts on large sets of data. For selection sort, an extreme number of passes through the list will have to be made finding the smallest element through each pass, and for insertion sort, a large number of comparisons and shifts will be required.
- ❑ For this, "divide-and-conquer" algorithms such as mergesort and quicksort are always the way to go, as they require much less comparisons to be made and instead subdivide the data, making it easier to manipulate.