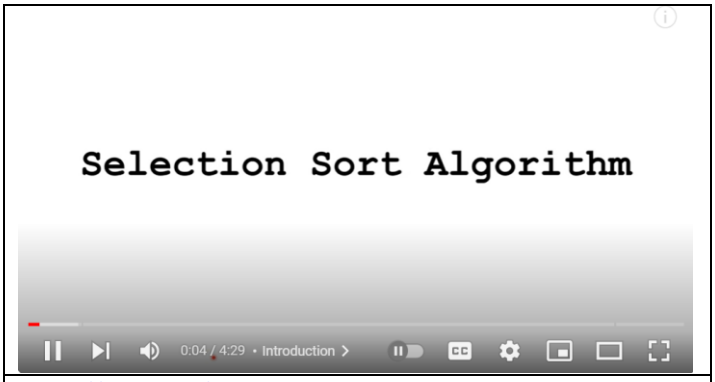
Set 35: Sorts and Searches

Skill 35.01: Implement and interpret a selection sort

The Selection Sort uses an incremental approach to sorting. During the first pass the smallest value is selected from the entire array and swapped with the first element. On the second pass the smallest value is selected from the array beginning with the 2nd element and swapped with the second element, etc.

The video below illustrates the selection sort concept,



https://youtu.be/jZBWRZZT8AI?si=bpfmHMu_MvveYhC2

Skill 35.01 Exercise 1

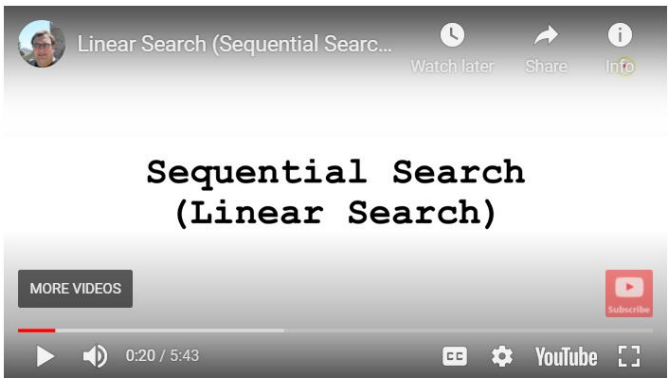
Skill 35.02: Implement a linear (sequential) search

Skill 33.02 Concepts

The sequential search, also known as the linear search, is the simplest search algorithm. The basic strategy is that every element in the data set is examined in the order presented until the value being searched for is found. If the value being searched for doesn't exist, a flag value is returned (such as -1 for an array).

If the data being searched are not sorted, then the sequential sort is a relatively efficient search. However, if the data being searched are sorted, we can do much better.

The video below illustrates the linear (sequential) search:



[Skill 35.02 Exercise 1]

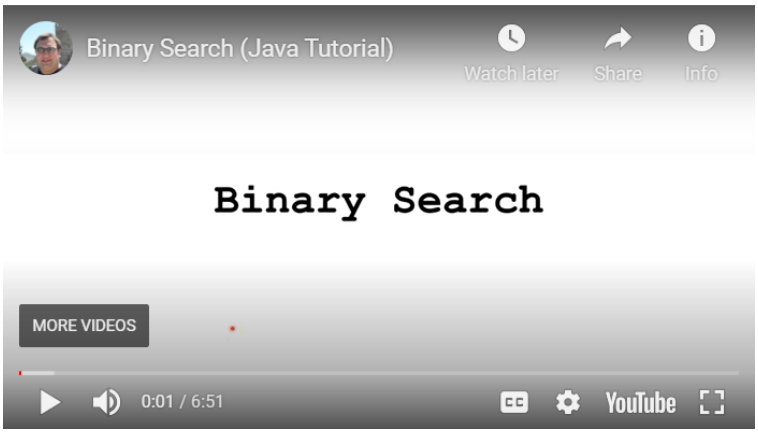
Skill 35.03: Interpret a binary search

Skill 33.03 Concepts

In a sequential search, after we compare against the first item, there are at most n−1 more items to look through if the first item is not what we are looking for. A key feature of a binary search is that the list being search **must** be sorted. A binary search takes advantage of a sorted list of elements, by first examining the middle item. If that item is the one we are searching for, we are done. If it is not the correct item, we can use the ordered nature of the list to eliminate half of the remaining items. If the item we are searching for is greater than the middle item, we know that the entire lower half of the list as well as the middle item can be eliminated from further consideration. The item, if it is in the list, must be in the upper half.

We can then repeat the process with the upper half. Start at the middle item and compare it against what we are looking for. Again, we either find it or split the list in half, therefore eliminating another large part of our possible search space.

The video below illustrates the binary search:



The following code illustrates the implementation of a binary search,

```
function binarySearch(data, target) {

    var start = 0;
    var end = data.length - 1;

    while (start <= end)   {
         var mid = Math.floor((start + end) / 2);     /* Calculate midpoint */

         if (target < data[mid])     {
              end = mid - 1;
         }     else if (target > data[mid])     {
              start = mid + 1;
         }     else     {
              return mid;
         }
    }
    return -1;
}
```

Skill 35.03 Exercises 1 & 2