CECS 277 – Lecture 7 – Searching

Searching Algorithms – There are many different types of algorithms that will search an array or collection of items for a particular value. Some algorithms are faster, some are more efficient, and some are just easier to program. They are usually compared by the amount of time it would take to process the collection given the worst case.

Equating Objects – When searching for a particular value in a list, it is important that the values can be compared for equality. It is easy to test for equality with numerical values since the relational operator for equality (==) can be used, but when testing if two objects are the same, the equality operator will only test if their memory addresses are the same. Instead, the equals() method from the Object class should be overridden and used.

```
public class Student {
     private int idNum;
     public Student( int i ) {
          idNum = i;
     }
     @Override
     public boolean equals( Object o ) {
          if( o instanceof Student ) {
               Student s = (Student) o;
               return idNum == s.idNum;
          }
          return false;
     }
}
```

Linear Search – A brute-force searching algorithm that is easy to implement. Worst case, it will take O(n) time.

<u>Algorithm</u>:

1. Examine each item in the list, one at a time.
2. When the item is found, stop.

Given a list: `list => [ 4, 2, 6, 5, 1, 3, 9, 0, 7, 8 ]`
And a search value: `value => 3`

A loop is used to examine each element in the list. If the element does not match the search value, then it continues to the next element. If it does match, then it returns the index of the matching element. If the loop manages to exhaust the entire list and not find the value, then the function should return -1, to signify that the value was not found.

```
public static int linear(ArrayList<Student>list,Student f){
     for( int i = 0; i < list.size(); i++ ) {
          if( list.get( i ).equals( f )) {
               return i;
          }
     }
     return -1;
}
```

Binary Search – A recursive search algorithm that only works on sorted lists. But since the list is sorted, the algorithm knows where the value should be, as such, it takes less time to find the value. This algorithm has a worst case of O(log(n)). However, since it takes at least O(n log(n)) to sort the list in the first place, it may still be more efficient to just do a linear search. Perform binary search when the list already needs to be sorted, or if the program you are creating will be doing many searches.

Algorithm –

1. Find the midpoint of the list and determine the value there.
2. Test that value with the value you are searching for, if it is the same, your value has been found. If not, then compare the values, if the search value is greater, then search the right half of the list, otherwise search the left.
3. Repeat from step 1 using the sublist of step 2.
4. If the sublist only has one item and it is not a match, then your search value was not found.

Given a sorted list: `list => [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]`
And a search value: `value => 3`

The indexes of the list range from 0-9, the midpoint would be located at index 4. The value at index 4 is the value 5. The search value 3 is less than 5 so the left side of the list will be searched on the next iteration.

```
sublist => [1, 2, 3, 4]
```

The indexes of the list range from 0-3, the midpoint would be located at index 1. The value at index 1 is the value 2. The search value 3 is greater than 2 so the right side of the list will be searched on the next iteration.

```
sublist => [3, 4]
```

The indexes of the list range from 2-3, the midpoint would be located at index 2. The value at index 2 is the value 3. The search value 3 is equal to 3 so the value has been found at index 2.

```
public static int binary( ArrayList<Student> list, Student find,
                                        int low, int high ) {
     if( low <= high ) {
          int mid = ( low + high ) / 2;
          if( list.get(mid).equals( find ) ) {
               return mid;
          } else if ( list.get(mid).compareTo( find ) < 0 ) {
               return binary(list, find, mid+1, high);
          } else {
               return binary(list, find, low, mid-1);
          }
     } else {
          return -1;
     }
}
```