Searching: The Binary Search

In the previous reading the linear search was examined. A careful examination of that search algorithm reveals that it requires a maximum of one iteration of its loop for each element of data to be searched. Thus, there is a linear mathematical relationship between the number of elements to be searched and the time (measured in loop iterations) to complete the search.

The next search algorithm is the BINARY SEARCH. This search algorithm is more complex, much faster, but has a built in limitation – the data must be sorted for the search algorithm to work!

The algorithm can be described:

- Pick the middle element and determine if it is the element you are trying to find
- If the element matches, return it's index
- Reduce the search area in half by narrowing the list
- Repeat these steps until the element is found or there are no elements in the search area

Searching is easier to visualize using names so we will use the following array:

```
String[] names={"Adam","Becky","Charlie","Dan","Ethan","Fred","Glen","Hilda","Ilsa"};
```

Searching an Array

The binary search algorithm's method signature is identical to that of the linear search except for the name.

Below is a summary of the information about the method to be written:

Method Attribute	Description	JAVA
Return Type	Returning an array index	`int`
Method Name	This method will search an array for a value	`binarySearch`
1st Parameter	The array to search, using a generic name	`String[] array`
2nd Parameter	The value to search for, or find	`String find`
Return value(s)	If the string to be found is in array, return the index	`0 to array.length-1`
	If the string to be found isn't in the array, return -1	`-1`

With this information the method header can be written:

```
public static int binarySearch(String[] array, String find)
```

The binary search algorithm is more complex than the linear search so let's first examine the steps from above in more detail:

Binary Search Algorithm Steps:

1. Define `find` as the value you are looking for in the array
2. Define the portion of the list remaining to be searched:
   a. The `start` index is the first element index
   b. The `end` index is the last element index
3. If `start>end` then exit, returning `-1` (`find` is not in the array)
4. Define the `middle` index as `(start + end)/2) + start`
5. Examine `names[middle]`:
   a. If equal to `find`, then exit, returning `middle`
   b. If greater than `find`, then define `end` as `middle - 1` and go to step 3
   c. If less than `find`, then define `start` as `middle + 1` and go to step 3

Next we will follow these steps to search for a value in the sample array!

Following these steps with the sample array pictured below:

1. Define `find` as the value you are looking for in the array
2. Define the portion of the list remaining to be searched,
3. If `start>end` then exit, returning `-1` (`find` is not in the array)
4. Define the `middle` index as `(start + end)/2) + start`
5. Examine `names[middle]`:
    a. If equal to `find`, then exit, returning `middle`
    b. If greater than `find`, then define `end` as `middle - 1` and go to step 3
    c. If less than `find`, then define `start` as `middle + 1` and go to step 3

`String[] names`

index	Value
0	"Adam"
1	"Becky"
2	"Charlie"
3	"Dan"
4	"Ethan"
5	"Fred"
6	"Glen"
7	"Hilda"
8	"Ilsa"

Step 1: define `find` as `"Dan"`, Step 2: define `start = 0` and `end=8`

Iteration	start	end	middle	names[middle]	equal, <, or > find
1st	0	8	(8-0)/2+0=4	"Ethanr"	> so end = 4 − 1
2nd	0	3	(3-0)/2+0=1	"Becky"	< so start = 1 + 1
3rd	2	3	(3-2)/2+2=2	"Charlie"	< so start = 2 + 1
4th	3	3	(3-3)/2+3=3	"Dan"	= so return 3

Putting it all together we can write the following method:

```
01 public static int binarySearch(String[] array, String find) {
02    int middle=0, start=0, end=array.length-1;
03    while (start<=end) {
04       middle=(end-start)/2+start;
05       int compare=array[middle].compareTo(find);
06       if (compare<0) {
07          start=middle+1;
08       } else if (compare>0) {
09          end=middle-1;
10       } else {
11          return middle;
12       }
13    }
14    return -1;
15 }
```

Line `01` is the method header from the prior page.

Line `02` creates the variable `middle` that will be used to store a new "guess" each time the loop iterates and the variables `start` and `end` to keep track of the "bottom" and "top" of the search range.

Line `03` tests to see if the loop should terminate.

Line `04` calculates a new `middle` by finding the halfway point between `start` and `end` using the equation: `middle=(end-start)/2+start`.

Line `05` compares the `String` found in the array at the index `middle` using the `.compareTo()` method resulting in a value that is either less than, equal to, or greater than 0.

Lines `06-07` set `start` to be equal to the current `middle` value `+ 1` because the `String` at index `middle` falls before `find` and thus all the values below the current `middle` need not be searched.

Lines `08-09` set `end` to be equal to the current `middle` value `- 1` because the `String` at index `middle` falls after `find` so all the values above the current `middle` need not be searched.

Lines `10-11` are executed because `compare` must have the value `0` because the `String` at index `middle` is equal to the `String find` so the current `middle` is returned.

Line 14 is only executed if the while statement on Line `04` becomes false, which means that the `String find` is not in `array`.

Searching for a value that is an `int`, `double`, or `boolean` is much simpler because they can be compared simply using the mathematical operators less than and greater than instead of the `.compareTo()` method.

Typically a method like search array would have the return value stored in a variable for later use.

```
public static void main(String[] args) {
   String[] names = 
   {"Adam","Becky","Charlie","Dan","Ethan","Fred","Glen","Hilda","Ilsa"};
   String lookFor="Adam";
   int index=binarySearch(names,lookFor);
   System.out.print(lookFor+" was ");
   if (index==-1) {
      System.out.println("not found");
   } else {
      System.out.println("found at index "+index);
   }
}
```

The code above uses the `lookFor` variable to store the value to try to find in the list.

Next, the *`binarySearch`*`()` method is called using the array of names and the variable `lookFor` with the result stored in the variable `index`.

After that the program either identifies where `lookFor` is in the array or indicates that it isn't in the array!