

Remember that binary search works on a sorted array; therefore, we must first use one of the sort algorithms to sort the array. Once sorted, we look at the middle element in the array. If that middle element matches what we want to find, then we are done. If not, then we have a choice to make. We either look in the first half or the second half, but not both. In comparing the element we want to find to the middle element, we will know if it is equal to, less, or greater than the middle element. If it is less, then we look in the first half and ignore the second half of the array. If it is more, we look in the second half. Once we have picked the half to search, we repeat the process. We keep splitting the array in half until we either find the match or run out of elements to search.

That is the binary search algorithm at work. You keep splitting in half the items you have to search, over and over again. It works much faster than sequential search.

Continuing with our example from the last lesson, we need to add a sort routine to sort our roster. To keep it simple, we just add a bubble sort. If we had implemented a comparable `<t>` interface, we could have used it for the comparison. That means if we wanted to binary search on something else, then we would need another bubble sort for that property.

```
public static void bubbleSort(assignment[] a)
{
    int out, in;
    assignment temp;

    for(out=a.length-1; out>1; out--) // outer loop (backward)
        for(in=0; in<out; in++) // inner loop (forward)
            if( a[in].getPerson().compareTo(a[in + 1].getPerson())
                > 0 )
                // out of order?
                {
                    // swap them
                    temp = a[in + 1];
                    a[in + 1] = a[in];
                    a[in] = temp;
                }
}
```

Now we have to write the binary search method. Since we are going to implement it on the person attribute, it will look like this:

```
public static int binarySearch(assignment[] r, String toFind )
{
    int high = r.length;
    int low = -1;
    int probe;

    while ( high - low > 1 )
    {
```

```

        probe = ( high + low ) / 2;
        if ( r[probe].getPerson().compareTo(toFind) > 0 )
            high = probe;
        else
            low = probe;
    }
    if ( (low >= 0) && (r[low].getPerson().compareTo(toFind)
                                                                == 0 ) )

        return low;
    else
        return -1;
}

```

Notice that this breaks the problem in half, each time through the loop, which means it will work in less time than the sequential search algorithm. Worst case for sequential search is to have to go through the entire array. Using that algorithm, every element in a 100,000 element array may have to be looked at. Binary search breaks that array in half over and over, and it is extremely fast. For small arrays, you probably will not notice the difference, but for larger ones, you will.

One drawback to the binary search method is that if you want to find multiple matches, it will not find them unless you modify the algorithm. You would have to use a binary search to find the location of one match and then linear search in both directions from that point, until you found values that did not match, which indicate that you should stop. For example, if I matched 5, then I search left until I find no fives, and then right until I find no fives. For every five I find I print it. Once I hit a non-5 value, I would stop. This would occur in both directions. And, if you want the records in the same order as they are in the list, that requires another modification. Also, since we are printing the matches, the binary search would not have a return value.

Just for fun, let's include multiple matches. Remember that it is a modification of the binary search algorithm and will not run as fast as the regular binary search, but will still be faster than a sequential search.

First, the binary search:

```

public static void binarySearch2(assignment[] r, String toFind
)
{
    int high = r.length;
    int low = -1;
    int probe;

    while ( high - low > 1 )
    {
        probe = ( high + low ) / 2;

        if ( r[probe].getLocation().compareTo(toFind) > 0 )

```

```

        high = probe;
    else
    {
        low = probe;
        if ( r[probe].getLocation().compareTo(toFind) == 0)
        {
            break;
        }
    }
}
if ( (low >= 0) && (r[low].getLocation().compareTo(toFind)
                    == 0 ))
{
    linearPrint(r, low, toFind);
}
else
    System.out.println("Not found: " + toFind);
}

```

Notice a few things. First, we add a check in the **while** loop, to break the loop once we find a match. Second, the **if** after the loop tests to see if we found a match and, if we did, it calls the **linearPrint()** method. Third, if no match is found, we print an appropriate message.

Here is the **linearPrint()** method:

```

public static void linearPrint(assignment[] r,int low,
                               String toFind)
{
    int i;
    int start= -1;
    int end = -1;

    // find starting point of matches
    i = low-1;
    while ((i >= 0) && (r[i].getLocation().compareTo(toFind)
                        == 0))
    {
        start = i;
        i--;
    }
    // find ending point of matches
    i = low+1;
    while ((i < r.length) &&
           (r[i].getLocation().compareTo(toFind) == 0))
    {
        end = i;
        i++;
    }
}

```

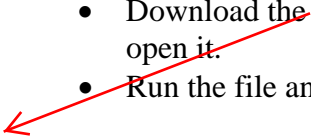
```
}  
// now print out the matches  
for(i = start; i <= end; i++)  
    System.out.println(r[i]);  
}
```

We wanted to print them out in the order they appear in the array, so first we used two **while** loops to determine the start and end points; then the **for** loop will print them up in the order that they are in the array.

To recap, binary search is great to quickly find out if something is in an array, but needs modification if you want to find all the matching values in an array. Also, remember that the array must be sorted, for binary search to work.

Now try it out:

- Download the [testAssignment2.java](#) file to your module 18 demo programs directory and open it.
- Run the file and make sure you understand it before you continue.



Click this box for the correct link to the testAssignment2.java file