Searching and Sorting

- **Linear Search**
- **Binary Search**
  -Reading p.671-679

Linear Search

- Searching is the process of determining whether or not a given value exists in a data structure or a storage media.
- We discuss two searching methods on one-dimensional arrays: linear search and binary search.
- The linear (or sequential) search algorithm on an array is:
  - Sequentially scan the array, comparing each array item with the searched value.
  - If a match is found; return the index of the matched element; otherwise return –1.
- Note: linear search can be applied to both sorted and unsorted arrays.

Linear Search

- The algorithm translates to the following Java method:

```
public static int linearSearch(Object[] array,
  Object key)
{
  for(int k = 0; k < array.length; k++)
    if(array[k].equals(key))
        return k;
  return -1;
}
```

Binary Search

- Binary search uses a recursive method to search an array to find a specified value
- The array must be a sorted array:

  `a[0]≤a[1]≤a[2]≤. . . ≤ a[finalIndex]`
- If the value is found, its index is returned
- If the value is not found, -1 is returned
- Note: Each execution of the recursive method reduces the search space by about a half

Binary Search

- An algorithm to solve this task looks at the middle of the array or array segment first
- If the value looked for is smaller than the value in the middle of the array
  - Then the second half of the array or array segment can be ignored
  - This strategy is then applied to the first half of the array or array segment

Binary Search

- If the value looked for is larger than the value in the middle of the array or array segment
  - Then the first half of the array or array segment can be ignored
  - This strategy is then applied to the second half of the array or array segment
- If the value looked for is at the middle of the array or array segment, then it has been found
- If the entire array (or array segment) has been searched in this way without finding the value, then it is not in the array

Pseudocode for Binary Search

Display 11.5 Pseudocode for Binary Search

ALGORITHM TO SEARCH a[first] THROUGH a[last]

```
/**
 Precondition:
 a[first]<= a[first + 1] <= a[first + 2] <=... <= a[last]
*/
```

TO LOCATE THE VALUE key:

```
if (first > last) //A stopping case
    return -1;
else
{
    mid = approximate midpoint between first and last;
    if (key == a[mid]) //A stopping case
        return mid;
    else if key < a[mid] //A case with recursion
        return the result of searching a[first] through a[mid - 1];
    else if key > a[mid] //A case with recursion
        return  the result of searching  a[mid + 1] through a[last];
 }
```

Recursive Method for Binary Search

Display 11.6 Recursive Method for Binary Search

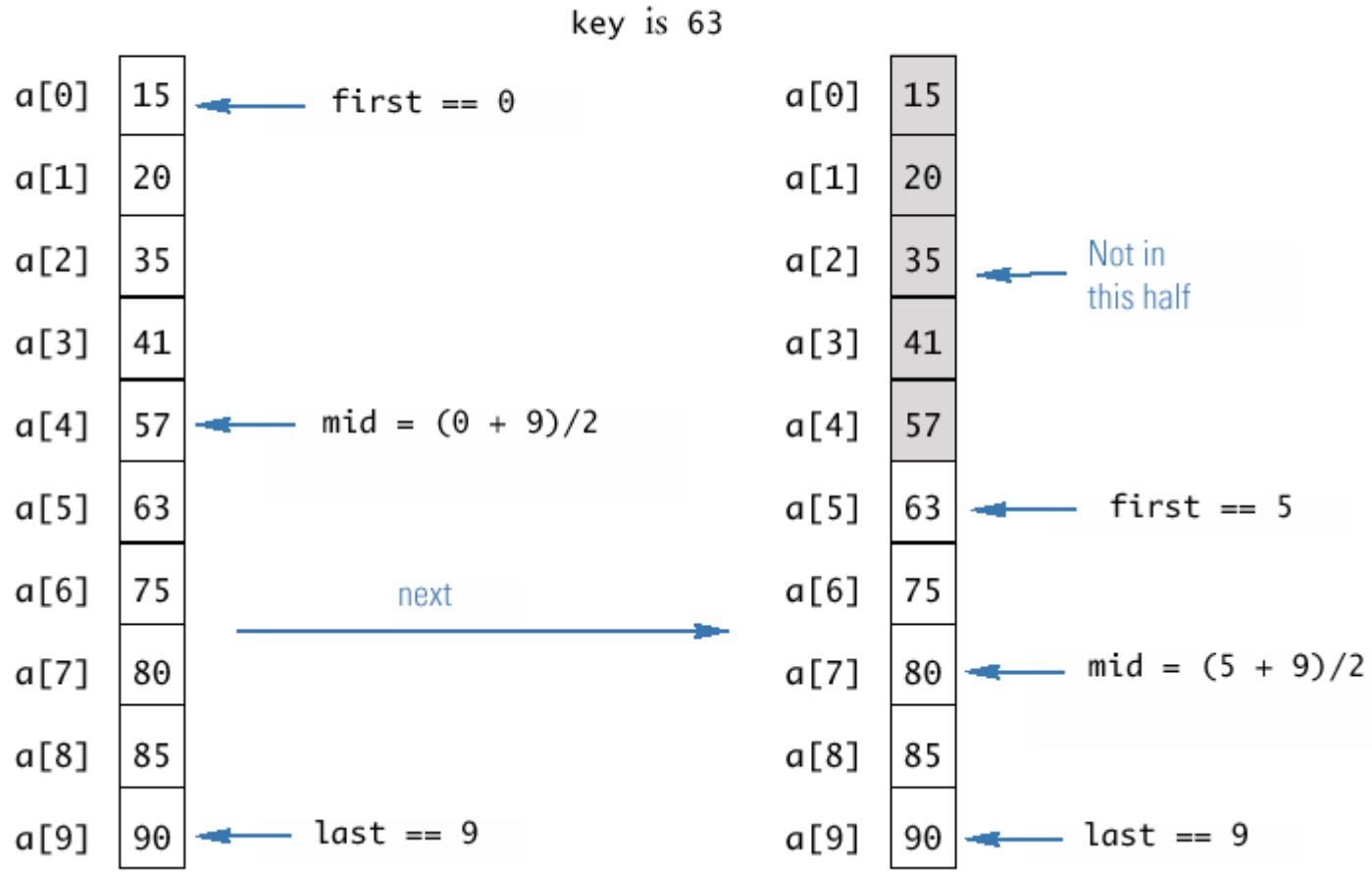
```
1   public class BinarySearch
2   {
3       /**
4        Searches the array a for key. If key is not in the array segment, then -1 is
5        returned. Otherwise returns an index in the segment such that key == a[index].
6        Precondition: a[first] <= a[first + 1]<= ... <= a[last]
7       */
8       public static int search(int[] a, int first, int last, int key)
9       {
10          int result = 0; //to keep the compiler happy.

11          if (first > last)
12              result = -1;
13          else
14          {
15              int mid = (first + last)/2;

16              if (key == a[mid])
17                  result = mid;
18              else if (key < a[mid])
19                  result = search(a, first, mid - 1, key);
20              else if (key > a[mid])
21                  result = search(a, mid + 1, last, key);
22          }
23          return result;
24      }
25  }
```

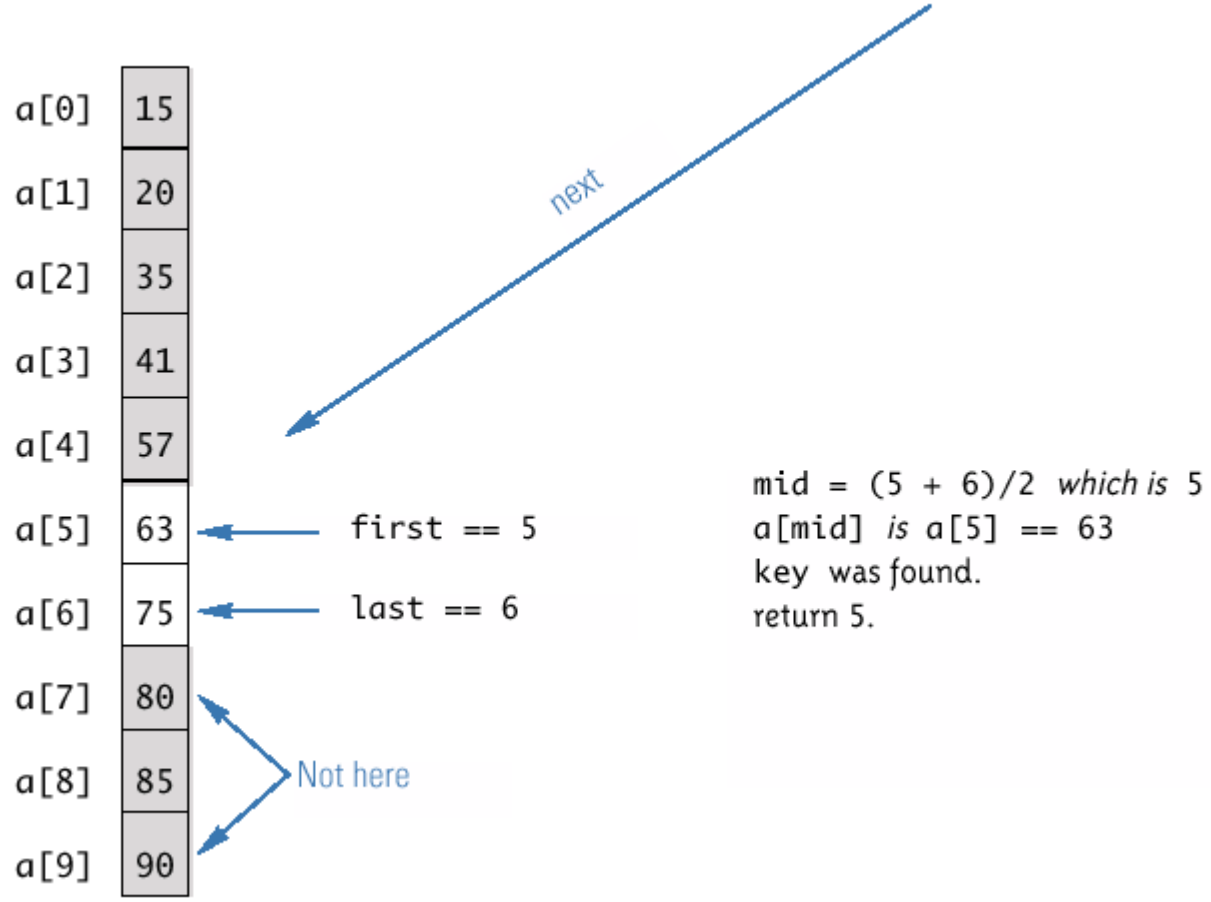
Execution of the Method `search` (Part 1 of 2)

Display 11.7 Execution of the Method search



Execution of the Method `search` (Part 1 of 2)

Display 11.7 Execution of the Method search (continued)



Checking the `search` Method

1. There is no infinite recursion
   - On each recursive call, the value of `first` is increased, or the value of `last` is decreased
   - If the chain of recursive calls does not end in some other way, then eventually the method will be called with `first` larger than `last`

Checking the `search` Method

2. Each stopping case performs the correct action for that case
   - If `first > last`, there are no array elements between `a[first]` and `a[last]`, so `key` is not in this segment of the array, and `result` is correctly set to `-1`
   - If `key == a[mid]`, `result` is correctly set to `mid`

Checking the `search` Method

3. For each of the cases that involve recursion, *if* all recursive calls perform their actions correctly, *then* the entire case performs correctly
   - If `key < a[mid]`, then `key` must be one of the elements `a[first]` through `a[mid-1]`, or it is not in the array
   - The method should then search only those elements, which it does
   - The recursive call is correct, therefore the entire action is correct

Checking the `search` Method

- If `key > a[mid]`, then `key` must be one of the elements `a[mid+1]` through `a[last]`, or it is not in the array
- The method should then search only those elements, which it does
- The recursive call is correct, therefore the entire action is correct

The method `search` passes all three tests:

Therefore, it is a good recursive method definition

Efficiency of Binary Search

- The binary search algorithm is extremely fast compared to an algorithm that tries all array elements in order
  - About half the array is eliminated from consideration right at the start
  - Then a quarter of the array, then an eighth of the array, and so forth

Efficiency of Binary Search

- Given an array with 1,000 elements, the binary search will only need to compare about 10 array elements to the key value, as compared to an average of 500 for a serial search algorithm
- The binary search algorithm has a worst-case running time that is logarithmic: O(log *n*)
  - A serial search algorithm is linear: O(*n*)
- If desired, the recursive version of the method **`search`** can be converted to an iterative version that will run more efficiently

Iterative Version of Binary Search (Part 1 of 2)

Display 11.9 Iterative Version of Binary Search

```
1   /**
2    Searches the array a for key. If key is not in the array segment, then -1 is
3    returned. Otherwise returns an index in the segment such that key == a[index].
4    Precondition: a[lowEnd] <= a[lowEnd + 1]<= ... <= a[highEnd]
5   */
6   public static int search(int[] a, int lowEnd, int highEnd, int key)
7   {
8       int first = lowEnd;
9       int last = highEnd;
10      int mid;

11      boolean found = false; //so far
12      int result = 0; //to keep compiler happy

13      while ( (first <= last) && !(found) )
14      {
15          mid = (first + last)/2;
```

Iterative Version of Binary Search (Part 2 of 2)

Display 11.9 Iterative Version of Binary Search (continued)

```
16            if (key == a[mid])
17            {
18                found = true;
19                result = mid;
20            }
21            else if (key < a[mid])
22            {
23                last = mid - 1;
24            }
25            else if (key > a[mid])
26            {
27                first = mid + 1;
28            }
29        }

30        if (first > last)
31            result = -1;

32        return result;
33    }
```