Recursion

Recursive thinking

- Consider searching for a target value in an array
  - Assume the array elements are sorted in increasing order
  - We compare the target to the middle element and, if the middle element does not match the target, search either the elements before the middle element or the elements after the middle element
  - Instead of searching n elements, we search $n/2$ elements

Recursive algorithm

Recursive Algorithm to Search an Array

`if` the array is empty

    return -1 as the search result

`else if` the middle element matches the target

    return the subscript of the middle element as the result

`else if` the target is less than the middle element

    recursively search the array elements before the middle element and return the result

`else`

    recursively search the array elements after the middle element and return the result

Recursive algorithm design

- There must be at least one case (the base case), for a small value of *n*, that can be solved directly
- A problem of a given size *n* can be reduced to one or more smaller versions of the same problem (recursive case(s))
- Identify the base case and provide a solution to it
- Devise a strategy to reduce the problem to smaller versions of itself while making progress toward the base case
- Combine the solutions to the smaller problems to solve the larger problem

Recursive length() function

```
public static int length(String str) {
    // base case
    if (str == null || str.equals("")) {
        return 0;
    }
    // recursive case
    return 1 + length(str.substring(1));
}

System.out.println(length("ace"));
```

Tracing recursive method

- The process of returning from recursive calls and computing the partial results is called *unwinding the recursion*