Recursion

Using recursion to implement maximum item in array

It's quite intuitive to get the maximum item of an array using standard methods - namely, iterative functions:

```
public static int maxInArr(int[] arr, int start, int end){
    int maxSoFar = arr[start];
        for(int i=start+1; i<=end; i++)
        maxSoFar = Math.max(maxSoFar, arr[i]);
    return maxSoFar;
}
```

However, there are times when it's useful, and often more intuitive, to use recursion in algorithms. I'll do the same thing as before but this time using recursion:

```
public static int maxInArrRec(int[] arr, int start, int end){
    //base case
    if(start == end) return arr[end];
    int middle = (start+end)/2;
    //splitting in half recursively
    int max1 = maxInArrRec(arr, start, middle);
    int max2 = maxInArrRec(arr, middle+1, end);
    //combining together
    return Math.max(max1, max2);
}
```

If you use the array [1,3,10,8,4], you get the expected result: 55.

Now, we can implement the Merge Sort algorithm to sort an array using recursion as well:

```
public static void mergeSort(int[] arr, int start, int end){
    if(start == end) return; //base case
    int middle = (start+end)/2;
    mergeSort(arr, start, middle);//splitting into two arrays
    mergeSort(arr, middle+1, end);
    merge(arr, start, middle, end);//combining the arrays
}
```

With the merge function being:

```
public static void merge(int[] arr, int start, int middle, int end){
    int[] auxiliar = new int[end-start+1];
    int p1 = start;
    int p2 = middle+1;
    int curr = 0;
```

```
    while(p1<=middle && p2<=end){
        if (arr[p1]<=arr[p2]) auxiliar[curr++] = arr[p1++]; //choose smallest number
        else auxiliar[curr++] = arr[p2++];
    }
    while(p1 <= middle) auxiliar[curr++] = arr[p1++];
    while(p2 <= end) auxiliar[curr++] = arr[p2++];
    //copy array auxiliar[] to arr[]
    for(int i=0; i<curr; i++) arr[start+i] = auxiliar[i];
}
```

If the array is [1, 3, 12, 28, 85, 10, 8, 4], you not surprisingly get [1, 3, 4, 8, 10, 12, 28, 85]!

Reversing an array

To reverse an array, you have to use the same process as before:

1. Start with a base case.
2. Do the operations you need.
3. Make recursive call to repeat the operations made previously.

Here's the code which does exactaly that:

```
static void reverse(int[] arr,int start, int end){
    //base case
    if(start >= end) return;
    //swaping first with last
    int tmp = arr[start];
    arr[start] = arr[end];
    arr[end] = tmp;
    //recursive call
    reverse(arr, start+1, end-1);
}
```

Flood fill

In this case, the recursive algorithm goes into every position of a 2D array and gets the number of '#' chars that are near each other (near being right next to each other horizontally and vertically). The recursive method which does that is:

```
static int floodfill(int y, int x){
    if(y<0 || y>=rows || x<0 || x>=cols) return 0;
    if(visited[y][x]) return 0;
    if(m[y][x]=='.') return 0;
    int count=1;
```

```
    visited[y][x] = true;
    count += floodfill(y,x+1);
    count += floodfill(y,x-1);
    count += floodfill(y+1,x);
    count += floodfill(y-1,x);
    return count;
}
```

It also uses the following variables:

```
static char[][] m;
static int cols;
static int rows;
static boolean[][] visited;
```

The visited 2D array keeps track of the positions which were already visited, preventing infinite loops from one position to the other and subsequently stack overflow.

Generating all subsets

One way you could do this (and the way it's done on the DS class) is to create a boolean array and use it to generate all the combination of trues and falses. For instance, if you have the set {1,2,3}, the set of all subsets is {1,2,3}, {2,3}, {1,2}, {1,3}, {1}, {2}, {3}, {}. If you then create a boolean array and assume the ith boolean value represent the whether the ith position on the set is present or not, then:

```
[True,True,True] = {1,2,3}
[True,True,False] = {1,2}
[True,False,True] = {1,3}
[True,False,False] = {1}
[False,True,True] = {2,3}
[False,True,False] = {2}
[False,False,True] = {3}
[False,False,False] = {}
```

Notice the pattern: the first item in the boolean array is True for the first 4 sets and False for the last 4. And it's also clear the the total number of subsets in an n size set is 2^n. For each first boolean value (True or False), you will then have the iterations of True and False. Here is the code which does that:

```
class TestSets{
    static void sets(int arr[]){
        boolean used[] = new boolean[arr.length];
        goSets(0, arr, used);
    }
```

```
    static void goSets(int curr, int[]arr, boolean[] used){
        //base case: when all the array is traversed
        if(curr == arr.length){
            System.out.print("Set: ");
            for(int i=0;i<arr.length;i++)
            if(used[i]) System.out.print(" "+arr[i]);
            System.out.println();
        }
        else{
            //generating sets which start with true
            used[curr] = true;
            goSets(curr+1, arr, used);
            //generating sets which start with false
            used[curr] = false;
            goSets(curr+1, arr, used);
        }
    }

    public static void main(String[] args){
        int[] arr = {1,2,3};
        sets(arr);
    }
}
```

Generating permutations

```
class TestPermutations{
    static void permutations(int arr[]){
    boolean used[] = new boolean[arr.length];
    int perm[] = new int[arr.length];
    goPermutation(0, arr, used, perm);
    }

    static void goPermutation(int curr, int arr[], boolean[] used, int[] perm){
    if(curr == arr.length){ //the entire array was traversed
        for(int i=0;i<arr.length;i++)
        System.out.print(arr[perm[i]]+" ");
        System.out.println();
    }
    else{
        for(int i=0;i<arr.length;i++)
        if(!used[i]){
            used[i] = true;
            perm[curr] = i;
            goPermutation(curr+1, arr, used, perm);
```

```
                used[i] = false;
            }
        }
    }

    public static void main(String[] args){
    int[] arr = {1,2,3};
    permutations(arr);
    }
}
```