Problem Set 5, Part I

Problem 1: Using recursion to print an array

1-1)

```
public static void print(int[] arr, int start){


        if (arr == null || start > arr.length || start < 0) { //(1)
            throw new IllegalArgumentException();
        }
        if(arr.length-1 == start){
            System.out.println(arr[arr.length-start-1]);
        } else{

            print(arr, start+1);
            System.out.println(arr[arr.length-1-start]);
        }
}
```

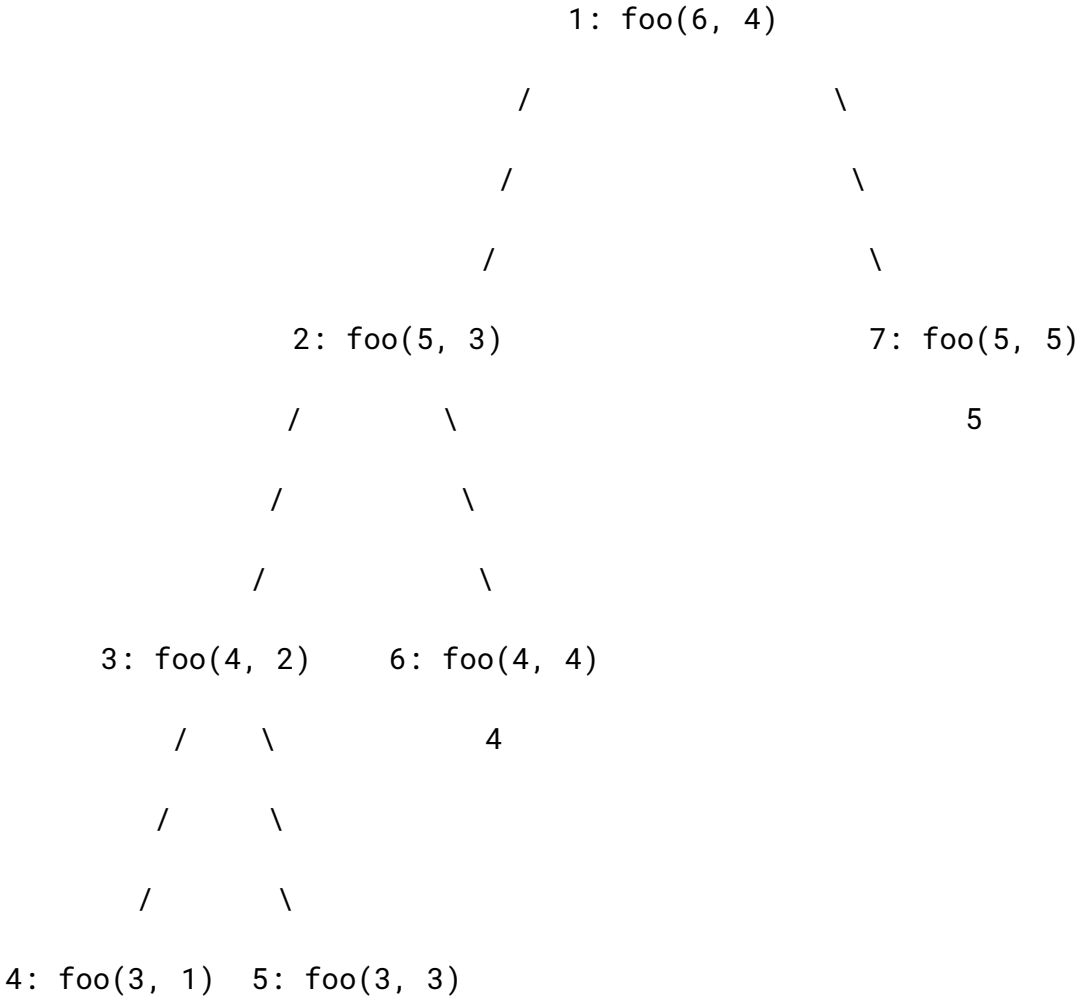
1-2)

```
public static void printMirrored(int[] arr, int start){ //recursively
        if (arr == null || start < 0) { //(1)
            throw new IllegalArgumentException();
        }
        if((arr.length * 2) <= start){

            return;
        } else if(start >= arr.length){
            //System.out.println(arr[start]);
            System.out.println(arr[2*arr.length- start -1]);
            start++;
            printMirror(arr, start);
            //System.out.println(arr[start]);
            return;
        } else{
            System.out.println(arr[start]);
            start++;
            printMirror(arr, start);
            return;
        }

    }
```

1-3) ***initial call:*** `printMirrored(arr,0 )`

Problem 2: A method that makes multiple recursive calls

2-1)



2-2)

Call 2 (foo(5,5)) return 5
Call 3 (foo(4,4)) return 4
Call 6 (foo(3,1)) return 3
Call 7 (foo(3,3)) return 3
Call 5 (foo(4,2)) return 6
Call 4 (foo(5,3)) return 10
Call 1 (foo(6,4)) return 15

Problem 3: Sorting practice

3-1) {3,4,18,24,33,40,8,10,12}

3-2) {4,10,18,24,33,40,8,3,12}

3-3) {4,10,18,8,3,12,24,33,40}

3-4) {10,18,4,24,12,3,8,40,33}

3-5) {10,18,4,8,12,3,24,40,33}

3-6) {4,10,18,24,33,40,8,3,12}

Problem 4: Practice with big-O

4-1)

function	**big-O expression**
a(n) = 5n + 1	a(n) = O(n)
b(n) = 2n^3 + 3n^4	b(n) = O(n^4)
c(n) = 10+5logn + 6n	c(n) = O(n)
d(n) = 4nlogn + 7n	d(n) = O(nlogn)
e(n) = 8n + 4n^2 + 3	e(n) = O(n^2)

4-2) O(n) because each case will be multiplied by 5, but the 5 can be dropped, hence making it O(n)

4-3)O(nlogn), will only iterate about half as many times, it will essentially iterate not in an n^x manner nor a n manner, so it's nlogn

Problem 5: Comparing two algorithms

worst-case time efficiency of algorithm A:O(nlogn)
Explanation: Both inner variables use the size of the array as the limit, and so n times n is n^2

worst-case time efficiency of algorithm B: O(n^2)
Explanation: since the worst case scenario is nlogn for merge sort and there is a lone n term for the for-loop which is smaller than, the worst case is O(nlogn)