

Problem Set 4, Part I

Problem 1: Rewriting a method

1-1)

```
public static boolean search(Object item, Object [] arr) {
    if(arr==null || item==null){
        throw new IllegalArgumentException();
    }
    for (int i = 0; i< arr.length; i++) {
        if (arr[i].equals(item)) {
            return true;
        }
    }

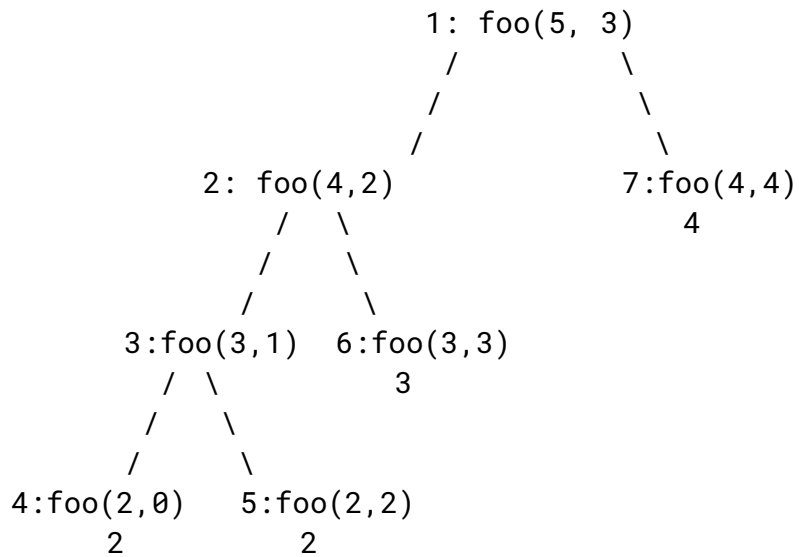
    return false;
}
```

1-2)

```
public static boolean search(Object item, Object[] arr, int start) {
    boolean found=false;
    if(arr==null || item==null){
        throw new IllegalArgumentException();
    }
    else if(arr.length==start){
        return false;
    }
    else{
        boolean rest=search(item,arr,++start);
        if(arr[start-1].equals(item)){
            return (true || rest);
        }
        else{
            return (false || rest);
        }
    }
}
```

Problem 2: A method that makes multiple recursive calls

2-1)



2-2)

call 4 foo(2,0) returns 2
Call 5 foo (2,2) returns 2
Call 3 foo(3,1) returns 4
Call 6 foo(3,3) returns 3
Call 2 foo(4,2) returns 7
Call 7 foo(4,4) returns 4
Call 1 foo(5,3) returns 11

Problem 3: Sum generator

3-1)

$$f(n) = ((n^2)/2) + (n/2)$$

3-2)

This was found by the arithmetic sum:

$$1 + 2 + \dots + (n - 2) + (n - 1)$$

$O(n^2)$

- n^2 is the highest order in the expression above
- Run time grows proportionally to n^2

3-3)

```
public static void generateSums(int n) {  
    int sum=0;  
    for (int i = 1; i <= n; i++) {  
        sum +=i;  
        System.out.println(sum);  
    }  
}
```

3-4)

$O(n)$

- Run time grows proportionally to n
 - The statement `sum+=i` is executed n times
- This is linear growth and it grows slower than $O(n^2)$ and therefore it is more efficient as n gets larger