**Problem Set 4, Part I**

**Problem 1: Rewriting a method**

**1-1)**

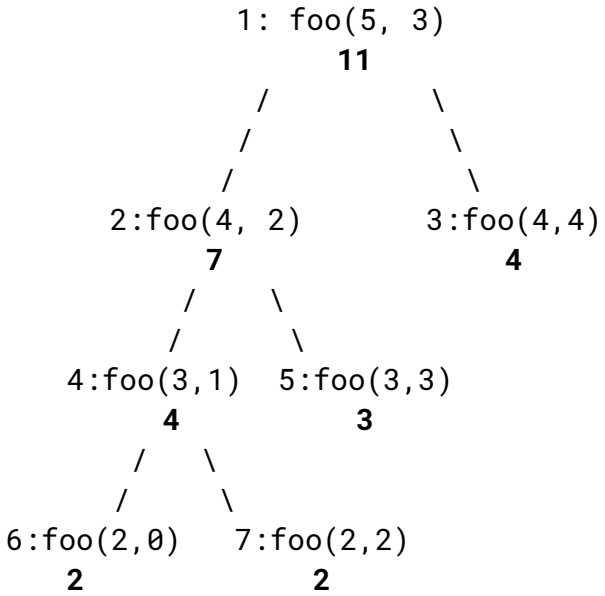
```
public static boolean search(Object item, Object[] arr) {
if (arr == null){
      throw new IllegalArgumentexception();
}

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) {
if (arr == null || arr.length == 0 || start < 0){
      throw new IllegalArgumentexception();
}

if(arr[start].equals(item)){
      return true;
}

else if (start == arr.length - 1){
      return false;
}
return search(item, arr, start +1);


}
```

**Problem 2: A method that makes multiple recursive calls**

**2-1)**




**2-2)**

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

**Problem 3: Sum generator**

3-1)

1+2+...+n = (n(n+1))/2

3-2)

O(n^2)

This is because there is a nested for loop within another for loop, and since the order of one for loop is n, then the order for two for loops is n * n. A.k.a n^2.

3-3)

```
public static void GenerateSums(int n){

int res = 0;
for(int j = 1; j <= n; j++){
res = res + j;
System.out.println(res);
}
}
```

3-4)

The time efficiency of my implementation is only O(n) because there is only one for loop. Hence, the function iterates over the sum only once where the sum is n integers long.