CP2530

Lab #1

Name: Ryan Horwood

Question #1: (20 points) Define a recurrence T(n) for the following code excerpt:

```
public int f(int n)
{
    if(n == 0 || n == 1)
        return n;
    else
        return 7 * f(n - 1) - 12 * f(n - 2);
}
```

Answer:

This recursive function has two base cases where n = 0 or 1 it returns itself, otherwise it returns 7 * f(n - 1) - 12 * f(n - 2). If we assume the solution is in the form of T(n) = a*r^n we will end up with ar^n = ar^(n-1) + a*r^(n-2) + c. This will simplify down to the quadratic equation r^2 = r + 1. Knowing r1 and r2 are greater than 1 or less than -1 it tells us that T(n) = Ar1^n + Br2^n. This will let us find the values of A and B using the formula T(n) = A*((1+sqrt(5))/2)^n + B*((1-sqrt(5))/2)^n. We can find the time complexity with this equation as O(r1^n) = O(((1+sqrt(5))/2)^n) and then giving us the O-notation as O(ϕ^n). This tells us the time complexity grows very fast.

Question #2: (20 Points) What are worst case and best case (O and Ω) run-time for the following piece of code? Be sure to explain your reasoning clearly.

```
public static boolean find(int[] n, int target) {

    boolean found = false;

    for(int i = 0; i < n.length && !found; i++) {

        if(n[i] == target)

            found = true;

    }

    return found;

}
```

Answer:

The worst-case scenario for the runtime will be when O(n) which is when the value is not found in the array making the time complexity proportional to the array. The best-case being O(1) with the value needed in the first position of the array which makes the time complexity non dependant on the array size. This will be the same for the Ω notation.