Assignment #2

Zhili Mai
Student Number: 215234842
Instructor: Andy Mirzaian

February 15th, 2019

1. Imagine a row of n lights , numbered 1 to n, that can be turned on or off only under certain conditions as follows. The first light can be turned on or off anytime. Each of the other lights can be turned on or off only when the preceding light is on and all other lights before it are off. If all the lights are on initially, how can you turn them all off? For three lights numbered 1 to 3, you can take the following steps, where 1 indicates a light that is ON and 0 indicates OFF:

111	3 lights ON initially
011	Turn OFF light 1
010	Turn OFF light 3
110	Turn ON light 1
100	Turn OFF light 2
000	Turn OFF light 1

We can solve this problem by indirect recursion using the two methods turnOff() and turnOn() that mutually call each other. The algorithm in pseudo-code for turnOff(n) is shown below:

(a) Write a similar algorithm in pseudo-code for turnOn(n).

Algorithm 1: turnOn(n)

```
/* Pre-Condition:  Light 1...n are all currently off     */
/* Post-Condition:  Light 1...n are all turned on        */
if $n = 1$ then
    Turn ON light 1
else
    turnOn($n - 1$)
    if $n > 2$ then
        turnOff($n - 2$)
    end
    Turn Off light n
    if $n > 2$ then
        turnOn($n - 2$)
    end
end
```

(b) Implement these algorithms in Java. Use the results in a program to display the sequence of steps to turn off n lights that are initially on. Show the output for a few values of n. The output format should be similar to the 3-lights example shown above.

n = 4

```
1 1 1 1   4 lights ON initially
1 0 1 1   Turn OFF Light 1
0 0 1 1   Turn OFF Light 0
0 0 1 0   Turn OFF Light 3
1 0 1 0   Turn ON Light 0
1 1 1 0   Turn ON Light 1
0 1 1 0   Turn OFF Light 0
0 1 0 0   Turn OFF Light 2
1 1 0 0   Turn ON Light 0
1 0 0 0   Turn OFF Light 1
0 0 0 0   Turn OFF Light 0
```

n = 5

```
1 1 1 1 1   5 lights ON initially
0 1 1 1 1   Turn OFF Light 0
0 1 0 1 1   Turn OFF Light 2
1 1 0 1 1   Turn ON Light 0
1 0 0 1 1   Turn OFF Light 1
0 0 0 1 1   Turn OFF Light 0
0 0 0 1 0   Turn OFF Light 4
1 0 0 1 0   Turn ON Light 0
1 1 0 1 0   Turn ON Light 1
0 1 0 1 0   Turn OFF Light 0
0 1 1 1 0   Turn ON Light 2
1 1 1 1 0   Turn ON Light 0
1 0 1 1 0   Turn OFF Light 1
0 0 1 1 0   Turn OFF Light 0
0 0 1 0 0   Turn OFF Light 3
1 0 1 0 0   Turn ON Light 0
1 1 1 0 0   Turn ON Light 1
0 1 1 0 0   Turn OFF Light 0
0 1 0 0 0   Turn OFF Light 2
1 1 0 0 0   Turn ON Light 0
1 0 0 0 0   Turn OFF Light 1
0 0 0 0 0   Turn OFF Light 0
```

n = 6

1 1 1 1 1 1	6 lights ON initially
1 0 1 1 1 1	Turn OFF Light 1
0 0 1 1 1 1	Turn OFF Light 0
0 0 1 0 1 1	Turn OFF Light 3
1 0 1 0 1 1	Turn ON Light 0
1 1 1 0 1 1	Turn ON Light 1
0 1 1 0 1 1	Turn OFF Light 0
0 1 0 0 1 1	Turn OFF Light 2
1 1 0 0 1 1	Turn ON Light 0
1 0 0 0 1 1	Turn OFF Light 1
0 0 0 0 1 1	Turn OFF Light 0
0 0 0 0 1 0	Turn OFF Light 5
1 0 0 0 1 0	Turn ON Light 0
1 1 0 0 1 0	Turn ON Light 1
0 1 0 0 1 0	Turn OFF Light 0
0 1 1 0 1 0	Turn ON Light 2
1 1 1 0 1 0	Turn ON Light 0
1 0 1 0 1 0	Turn OFF Light 1
0 0 1 0 1 0	Turn OFF Light 0
0 0 1 1 1 0	Turn ON Light 3
1 0 1 1 1 0	Turn ON Light 0
1 1 1 1 1 0	Turn ON Light 1
0 1 1 1 1 0	Turn OFF Light 0
0 1 0 1 1 0	Turn OFF Light 2
1 1 0 1 1 0	Turn ON Light 0
1 0 0 1 1 0	Turn OFF Light 1
0 0 0 1 1 0	Turn OFF Light 0
1 0 0 1 1 0	Turn ON Light 0
0 0 0 1 1 0	Turn OFF Light 0
0 0 0 1 0 0	Turn OFF Light 4
1 0 0 1 0 0	Turn ON Light 0
1 1 0 1 0 0	Turn ON Light 1
0 1 0 1 0 0	Turn OFF Light 0
0 1 1 1 0 0	Turn ON Light 2
1 1 1 1 0 0	Turn ON Light 0
1 0 1 1 0 0	Turn OFF Light 1
0 0 1 1 0 0	Turn OFF Light 0
0 0 1 0 0 0	Turn OFF Light 3
1 0 1 0 0 0	Turn ON Light 0
1 1 1 0 0 0	Turn ON Light 1
0 1 1 0 0 0	Turn OFF Light 0
0 1 0 0 0 0	Turn OFF Light 2
1 1 0 0 0 0	Turn ON Light 0
1 0 0 0 0 0	Turn OFF Light 1
0 0 0 0 0 0	Turn OFF Light 0

(c) Obtain, with sufficient explanation, an accurate estimate of the number of times lights are turned off or on during the entire process. Express the answer as a function of n.

Let turnOn(n) and turnOff(n) be $t(n)$,if we calculate the $t(n)$

$$\begin{aligned} t(1) &= 1 \\ t(2) &= 1 + t(1) = 2 \\ t(3) &= 1 + t(1) + t(1) + t(2) = 5 \\ t(4) &= 1 + t(2) + t(2) + t(3) = 10 \\ t(5) &= 1 + t(3) + t(3) + t(4) = 21 \\ t(6) &= 1 + t(4) + t(4) + t(5) = 42 \end{aligned}$$

So we can find the recurrence relation in the equations as $t(n) = t(n-1) + 2t(n-2) + 1$. The base case are $t(0) = 0, t(1) = 1$. Base on this, we can solve the recurrence relation.

It is same as $a_n = a_{n-1} + a_{n-2} + 1$.

$$\text{then we rewrite it as } x^2 - x - 2 = 0 \tag{1}$$

$$\text{solve the function } (x-2)(x+1) = 0 \tag{2}$$

$$\text{we have two roots } x = 2, x = -1 \tag{3}$$

so the $a_n^{(h)} = \alpha(2^n) + \beta(-1)$.

we have a constant coefficients $F(n) = 1$, assume it is c then

$$a = 2a + a + 1 \tag{4}$$

$$a - 2a - a = 1 \tag{5}$$

$$-2a = 1 \tag{6}$$

$$a = -\frac{1}{2} \tag{7}$$

so $a_n^{(p)} = -\frac{1}{2}$.

then $a_n = a_n^{(p)} + a_n^{(h)} = \alpha(2^n) + \beta(-1) - \frac{1}{2}$. solve for α and β.

we have a system of equation

$$\begin{cases} \alpha + \beta - \frac{1}{2} = 0 \\ 2\alpha - \beta - \frac{1}{2} = 1 \end{cases}$$

solve for the system we have $\alpha = \frac{2}{3}$ and $\beta = -\frac{1}{6}$. so we have the function $a_n = \frac{2^{n+1}}{3} - \frac{(-1)^n}{6} - \frac{1}{2}$

(d) Combine the two algorithms turnOff(n) and turnOn(n) and replace them with a single recursive two parameter algorithm flipSwitches(n, s) written in pseudo-code. The boolean parameter s indicates which version is intended: switching n lights ON or OFF.

Algorithm 2: flipSwitches(n, s)

```
/* Pre-Condition:  $1...n$ lights are $\neg s$ state          */
/* Post-Condition:  $1...n$ lights are $s$ state             */
if $n^{th}$ light is $s$ state then
    skip
end
if $n = 1$ then
    Turn ON light 1
else
    flipSwitches($n - 1$, ON)
    if $n > 2$ then
        flipSwitches($n - 2$, OFF)
    end
    Turn Off light n
    if $n > 2$ then
        flipSwitches($n - 2$, ON)
    end
    flipSwitches($n - 1$, $s$)
end
```

2. Give a Java implementation of the deque ADT (see [GTG §6.3] for definition) using two stacks as the only instance variables. What are the running times of the methods?
   From the new Deque Java source code below. We have 10 functions. The standard deque function from books are: addFirst(e), addLast(e), removeFirst(), removeLast(), first(), last(), size(), and isEmpty().

   Other then the constructor, there is one helper method call $moveStack(Stack < E > a, Stack < E > b)$ that helps to pop reverse the stack.

   Obviously, the constructor declares two stack from line 5-6. It is an $O(1)$ for the constructor. Including size(), isEmpty(), addFirst(e), addLast(e) are $O(1)$ because they only do one operation, when method size(), push(), and pop() for Stack runs on $O(1)$.

   For moveStack(a, b), if the a stack is empty, it move the stack from b to a with a while loop , which base on how many object in b. This means that this will runs on approximately $n + 5$ which is $O(n)$ when there are n element in b.

   Then we have first() and last() that they call to moveStack and then do basic operation on line 26-27 and 31-32. Which means they are also running on $O(n)$.

   The last two, removeFirst() and removeLast() are basically call to moveStack which will have the same running as moveStack(a, b), which is $O(n)$

```java
public class NewDeque<E> {
    private Stack<E> f;
    private Stack<E> r;
    public NewDeque() {
        f = new Stack<>();
        r = new Stack<>();
    }

    public int size() { return f.size() + r.size(); }
    public boolean isEmpty() { return size() == 0; }

    private E moveStack(Stack<E> a, Stack<E> b) {
        if (!a.isEmpty()) {
            return a.pop();
        } else if (!b.isEmpty()) {
            while (b.size() > 1) {
                a.push(b.pop());
            }
            return b.pop();
        }
        return null;
    }

    public E first() {
        E e = moveStack(f, r);
        f.add(e);
        return e;
    }
    public E last() {
        E e = moveStack(r, f);
        r.add(e);
        return e;
    }

    public void addFirst(E e) { f.push(e); }
    public void addLast(E e) { r.push(e); }

    public E removeFirst() {
        return moveStack(f, r);
    }
    public E removeLast() {
        return moveStack(r, f);
    }
}
```