COMP 3761 – Summer 2019 --- Assignment 2 --- Time Allocation: 50 minutes.

Answer the following questions and submit to D2L by deadline.

1. **Fair attraction.** In olden days, one could encounter the following attraction at a fair. A light bulb was connected to several switches in such a way that it lighted up only when all the switches were closed. Each switch was controlled by a push button; pressing the button toggled the switch, but there was no way to know the state of the switch. The object was to turn the light bulb on. Design an algorithm to turn on the light bulb with the minimum number of button pushes needed in the worst case for 'n' switches.

Answer:

Since we don't know what the switches position is before starting the game (ex. if any are on or any are off), and since there is no feedback or knowledge on what positions the switches currently are, the worst case and the number of presses needed for every button equals (2^n)-1

Explanation:

Each switch has 2 positions on and off. We can cover all permutations at (2^n)-1 using grey code which is an alternative to binary counting

```
def isOdd(integer):
    return integer % 2 == 1


def isEven(integer):
    return integer % 2 == 0


def _list_to_string(li):
    return ''.join(map(str, li))


class GrayCode(object):
    def __init__(self, nbits):
        self._nbits = nbits
        self._grayCode = []
        self.__generate()


    def __getitem__(self, i):
        return self._grayCode[i]
    def __str__(self):
        return str(self._grayCode)

    __repr__ = __str__

    def __iter__(self):
        return self._grayCode.__iter__()

    def __generate(self):
        li = [0 for i in range(self._nbits)]
        self._grayCode.append(_list_to_string(li))

        for term in range(2, (1<<self._nbits)+1):
            if isOdd(term):
                for i in range(-1,-(self._nbits),-1):
                    if li[i]==1:
                        li[i-1]=li[i-1]^1
                        break

            if isEven(term):
                li[-1]=li[-1]^1

            self._grayCode.append(_list_to_string(li))

class GrayCodeIterator(object):
    def __init__(self, nbits):
        self._nbits = nbits

    def __iter__(self):
                    li = [0 for i in range(self._nbits)]
                    yield _list_to_string(li)
        for term in range(2, (1<<self._nbits)+1):
            if isOdd(term):
                for i in range(-1,-(self._nbits),-1):
                    if li[i]==1:
                        li[i-1]=li[i-1]^1
                        break

            if isEven(term):
                li[-1]=li[-1]^1

            yield _list_to_string(li)


if __name__=='__main__':
    d = 4
    codes=GrayCode(d)
    print ('%d digits gray codes:' % d)
    print (codes)
    print ('Using Iterator:')
    for c in GrayCodeIterator(4):
        print (c)
```

Josephus Problem. To answer the following question, you'd need to read pages 154 and 155 of your textbook.

2. For the Josephus problem,

   a. Compute J(n) for n=1,2,3,…,15

   The last survivor for a group of 15 is 15

   [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1]

   [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1, 3]

   [7, 8, 9, 10, 11, 12, 13, 14, 15, 1, 3, 5]

   [9, 10, 11, 12, 13, 14, 15, 1, 3, 5, 7]

   [11, 12, 13, 14, 15, 1, 3, 5, 7, 9]

   [13, 14, 15, 1, 3, 5, 7, 9, 11]

   [15, 1, 3, 5, 7, 9, 11, 13]

   [3, 5, 7, 9, 11, 13, 15]

   [7, 9, 11, 13, 15, 3]

   [11, 13, 15, 3, 7]

   [15, 3, 7, 11]

   [7, 11, 15]

   [15, 7]

   [15]

   b. Discern a pattern in the solutions for the first fifteen values of n and prove its general validity.

```
def josephus(arr):

while len(arr)>1:
       arr.pop(1)
       temp = arr[0]
       arr.pop(0)
       arr.append(temp)
       print(arr)

josephus([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
```

```
Examples from the book validated
```
[3, 4, 5, 6, 1]

[5, 6, 1, 3]

[1, 3, 5]

[5, 1]

[5]

[3, 4, 5, 6, 7, 1]

[5, 6, 7, 1, 3]

[7, 1, 3, 5]

[3, 5, 7]

[7, 3]

[7]

c. Prove the validity of getting J(n) by a one-bit cyclic shift left of the binary representation of n.

Binary shift works as long as we add 1 to the result

1) 1111 = 15

1110 = 14 + 1 = 15 = f(15)

2) 111 = 7

110 = 6 + 1 = 7 = f(7)

3) fn(16) = b10000 << 1 = b0000 = 0+1 = 1