Next Greater Element

Using indexes and traversing from L->R

```
TC: O(N)
SC: O(N)
vector<int> nextGreaterElement(vector<int> data) {
    stack<int> s;
    vector<result> res;

    for (int i = 0; i < data.size(); i++) {
        int curr = data[i];
        res.push_back(-1);

        while (!s.empty() && data[s.top()] < curr) {
            int idx = s.pop();
            res[idx] = curr;
        }

        s.push(i);
    }
    return res;
}
```

```
Data  2  3  4  2  3
index 0  1  2  3  4

i     0  1  2 3    4
s     0  1  2 2,3  2,4
res   3  4 -1 3   -1
```

In [ ]:

In [ ]:

Thinking in Recursion

- a function that calls itself is a recusive function
- recursion uses stack memory
- recursive functions have notion of a `base` or `terminating` condition

1. Factorial
2. https://leetcode.com/problems/reverse-string/ (https://leetcode.com/problems/reverse-string/)
3. https://leetcode.com/problems/power-of-two/ (https://leetcode.com/problems/power-of-two/)
4. https://leetcode.com/problems/decode-string/ (https://leetcode.com/problems/decode-string/)
5. https://leetcode.com/problems/count-good-numbers/ (https://leetcode.com/problems/count-good-numbers/)

In [ ]:

In [3]:
```
def printer(n):
    print("n =", n)

printer(10)
```

```
n = 10
```

In [5]:
```
def f1():
    print("f1")

def f2():
    print("f2")
    f1()

f2()
```

```
f2
f1
```

In [6]:
```
def f1():
    print("f1")

def f2():
    f1()
    print("f2")

f2()
```

```
f1
f2
```

In [ ]:
```

```

In [8]:
```
def printer(n):
    print("n =", n)
    printer( n-1 )

printer(5)

# printer(5)
#         printer(4)
#             printer(3)
#                 printer(2)
#                     printer(1)
#                         printer(0)
#                             printer(-1)
```

```
n = 5
n = 4
n = 3
n = 2
n = 1
n = 0
n = -1
n = -2
n = -3
n = -4
n = -5
n = -6
n = -7
n = -8
n = -9
n = -10
n = -11
n = -12
n = -13
```

In [9]:
```
def printer(n):
    if n <= 0:
        return

    print("n =", n)
    printer( n-1 )

printer(5)

# printer(5)
#         printer(4)
#             printer(3)
#                 printer(2)
#                     printer(1)
#                         printer(0) X

 ->[] -> [] -> [][][]X
```

```
n = 5
n = 4
n = 3
n = 2
n = 1
```

In [10]:
```
# TC: O(n)
# SC: O(n)
def printer(n):
    if n <= 0:
        return

    printer( n-1 )
    print("n =", n)
    return

printer(5)

# printer(5)
#         printer(4)
#             printer(3)
#                 printer(2)
#                     printer(1)
#                         printer(0) X
#

Stack
(l6, n=1)
(l6, n=2)
(l6, n=3)
(l6, n=4)
(l6, n=5)

printer(5)
printer(4)
printer(3)
..
printer(0)
```

```
n = 1
n = 2
n = 3
n = 4
n = 5
```

In [11]:
```
# TC: O(n)
# SC: O(1)
def printer(n):
    while n > 0:
        print("n =",n)
        n -= 1

printer(5)
```

```
n =  5
n =  4
n =  3
n =  2
n =  1
```

In [ ]:
```

```

Sum of first n numbers

In [12]:
```
def sum_n(n):
    if n <= 0:
        return 0

    return n + sum_n(n-1)

sum_n(4)
# sum_n(4)
#     return 4 + sum_n(3)
#                 return 3 + sum_n(2)
#                             return 2 + sum_n(1)
#                                         return 1 + sum_n(0)
```

Out[12]: 10

Factorial in recursive manner

In [ ]:
```

```

Question
https://leetcode.com/problems/reverse-string/ (https://leetcode.com/problems/reverse-string/)

In [ ]:
```
class Solution:

    def reverseStrUtil(self, s: List[str], i: int, j:int) -> None:
        if i >= j:
            return

        temp = s[i]
        s[i] = s[j]
        s[j] = temp
        self.reverseStr(s, i+1, j-1)

    def reverseString(self, s: List[str]) -> None:
        self.reverseStrUtil(s, 0 , len(s) - 1)
```

In [19]:
```
def reverseStrUtil(s, i: int) -> None:
    if i >= len(s)//2:
        return

    temp = s[i]
    s[i] = s[len(s) -i - 1]
    s[len(s) -i -1] = temp
    reverseStrUtil(s, i+1)

def reverseString(s) -> None:
    reverseStrUtil(s, 0)

s = list("abcd")
print(s)
reverseString(s)
print(s)
```

```
['a', 'b', 'c', 'd']
['d', 'c', 'b', 'a']
```

In [ ]:

In [ ]:

In [25]:
```
def my_pow(x,y):
    res = 1
    i = 0
    while(i < y):
        res *= x
        i += 1

    return res

print(pow(2, 10))
print(my_pow(2,10))
```

```
1024
1024
```

In [27]:
```
print(my_pow(2,1000))
```

```
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571964776865421676604298316526243868372056680693760
```

what is (x to the power y) % z ?

In [46]:
```
# TC: O(n)
# SC: O(1)
def my_pow(x, y, z):
    res = 1
    i = 0
    while(i < y):
        res *= x
        i += 1

    return res%z

print(my_pow(2,1000000, 20))
```

```
16
```

In [ ]:

Modulo Arithmetic

(x*y)%z = (x%z * y%z)%z // why ?? -> int overflow

In [ ]:

```
2^5 = 2^2 * 2^2 * 2^1
2^10 = 2^5 * 2^5
2^21 = 2^10 * 2^10 * 2^1

(x^y *x^z) = (x ^ (y+z))
pow(2, 21)
```

In [39]:
```
# TC: O(Log n)
# SC: O(Log n)

def my_pow(x,y):
    if y ==0:
        return 1

    if y == 1:
        return x

    if y%2 == 0:
        r = my_pow(x, y//2)
        return r * r
    else:
        r = my_pow(x, y//2)
        return r * r * x

print(my_pow(2,10))
print(my_pow(2,11))
print(my_pow(2,1000000))
```

```
1024
2048
99006562292958982506979236163019032507336242417875673328663961145317094833094861030546145512346483914824315070345837238835106
58989416314927422565031572905372314869377232871775494713664238970125842914489614716338412188631103792398056007740136270960553
07053866717981233606159217927983273223643032286260657430925691627858204283477200179449319005699514097510312526917394308961549
31403784291767137807931479533574241316141949252646322704610310518726715434463264155347328283288444762629663913610121194240251
00613431716524949459232757177220781095411635556324285623401399844018872126340393003478382714500015263923202290122589106172290
37066549822325526657074236467887628194407017819696756431326847303952947178641997730586792287821993064882303345065034596791221
52391755510927579220303167443634444817606652173925000462615745612852501910786888559853823959109102819047524378271693221421142
12392700345639503812353502503119133229035702263721456724737028852183729673865439926655978577772441147113534229818712960636023
42124838258072006674958033769065568092701436494004477009798690062991574175490539021387229485951252745725843568693057429433187
61488042125870705081210989810673096016733118551190531977968948827750233385474039969174297188347576501021741278616607099138092
02613818145296878884244352274572274581016535608594827480046503854138131186451279310440923966416764730530096072681651965998558
83163924960253450803846226233415060476114953790220372111150760791409757457811339576926357615128826298343155141561825405071450
90496908356938824373503748136802379014732234896458593630309175937966148659079272356756206251296994270163564782160224336729043
44807606952184673828762029275058248015293274931396581899857266679461433306593743364113813692627810553555141318609189614864029
74778748508880160549702100345096024678742308819273864162687251086992052277838707721304863610277948260856068796303905192240404
64652661321890595123230887255644796047463416268626842402761431122536389331425858460985638366320374387759550542009788137760423
26316628619102324598113841300603753775993390952276982455476518137898587713078425308571330225253178917068683209419560529835520
```

In [ ]:
```
my_pow(3, 21)
            my_pow(3, 10)
                my_pow(3, 5)
                    my_pow(3, 2)
                        my_pow(3, 1)

my_pow(3, 32)
            my_pow(3, 16)
                my_pow(3, 8)
                    my_pow(3, 4)
                        my_pow(3, 2)
                            my_pow(3, 1)
```

In [ ]:
```
```

In [44]:
```
# TC: O(log n)
# SC: O(log n)

def my_pow(x,y,z):
    if y ==0:
        return 1

    if y == 1:
        return x%z

    if y%2 == 0:
        r = my_pow(x, y//2, z)
        return (r%z * r%z)%z
    else:
        r = my_pow(x, y//2, z)
        return (r%z * r%z * x%z)%z

# print(my_pow(2,10))
# print(my_pow(2,11))
print(my_pow(2,1000000, 20))
```

```
16
```

In [ ]:
```
```

In [ ]:
```
```

Binary Number

```
symbols -> 0, 1 = 2

size 1 -> 2
size 2 -> 4
size 3 -> 8

(2^size)
```

Octal

```
symbols -> 0..7 = 8
size 1-> 8
size 2-> 64

(8^size)
```

Question

https://leetcode.com/problems/count-good-numbers/description/ (https://leetcode.com/problems/count-good-numbers/description/)

```
Size=2

_ _
0 1


Size=3

_ _ _
0 1 2

Size=4

_ _ _ _
0 1 2 3

0-> 0 2 4 6 8 = 5
1-> 2 3 5 7 = 4


number is of size n
n  = 1 even indexes = 1 odd indexes = 0, 5
n  = 2 even indexes = 1 odd indexes = 1, 5*4
n  = 3 even indexes = 2 odd indexes = 1, 5*4*5
n  = 4 even indexes = 2 odd indexes = 2, 5*4*5*4

count of even positions = (n+1)/2
count of odd positions = n/2


combinations for even position =  (5 ^ (n+1)/2 )
combinations for odd position =  (4 ^ (n)/2 )

total = even Comb * odd Comb
      = (5 ^ (n+1)/2 ) * (4 ^ (n)/2)
```

In [45]:
```
(2**31 > (10**9 + 7))
```

Out[45]: True

In [ ]:

DIY

https://leetcode.com/problems/power-of-two/ (https://leetcode.com/problems/power-of-two/)
https://leetcode.com/problems/decode-string/ (https://leetcode.com/problems/decode-string/)

In [ ]: