Agenda

Prob 1 : Fast Power

Prob 2 : Print Array

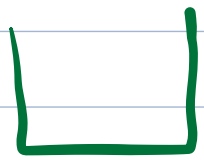
Prob 3 : Indices of an Array

Prob 4 : Check Palindrome

Revision

Quiz 1 : Recursion : A function that solves a problem by breaking it into smaller subproblems by calling itself

Quiz 2 : Data structure for fn call tracing : Stack



Quiz 3 : Base case for calculating factorial :

```
if (N==0) return 1
```

Quiz 4 : TC and SC for factorial (recursion)

N → N-1 → N-2 → ... → 0

TC : O(N)

SC : O(N)

Quiz 5: Fib(N) : Fib(N-1) + Fib(N-2)

1. Given two integers a and n, find a^n using recursion.

$a = 2$, $n = 3$ ans = $2^3 = 8$

$\downarrow$

$2 \times 2 \times 2$

$$a^N = \underbrace{a \times a \times a \ldots\ldots \times a}_{N \text{ times}}$$

BF : Multiply 'a' N times TC: O(N)

SC: O(1)

Recursion

$$2^9 = \underbrace{2 \times 2 \times \ldots\ldots}_{} \times 2 \;]^{9 \text{ times}}$$

$$2^9 = 2^8 \times 2$$

$$a^N = a^{N-1} \times a$$

Base Case : If (n==0)

return 1

```
// Given a and n, return a^N
int pow(int a, int n) {
    If (n==0)
        return 1
    return pow(a, n-1) * a
}
```

125

$\boxed{a^1 = a^0 \times a}$

a n

pow (5, 3)

↓ ↑ 125

pow (5, 2) × 5

↓ ↑ 25

pow (5, 1) × 5

↓ ↑ 5

pow (5, 0) × 5

1

// Assumption

fn ( ) {

// Base case

// main Logic

}

pow(a, n)

↓

a, n-1

↓

a, n-2

↓

⋮

a, 0

N+1

TC : O(N)

SC : O(N)

↓

Fn call stack

Optimized Approach

$a^N = a^{N/2} \times a^{N/2}$

$a^N \xrightarrow{N \text{ is even}} a^{N/2} \times a^{N/2}$

$a^N \xrightarrow{N \text{ is odd}} a^{N/2} \times a^{N/2} \times a$

$7^6 = 7^5 \times 7$

$7^6 = 7^3 \times 7^3$

$2^{10} = 2^5 \times 2^5$

$2^{11} = 2^5 \times 2^5 \times 2$

Assumption : Given a and n, return a^n

Main Logic : If n is even

$pow(a,n) = pow(a, n/2) \times pow(a, n/2)$

else

$pow(a,n) = pow(a, n/2) \times pow(a, n/2) \times a$

Base Case : If (n==0)

return 1

$$2^{10}$$

$$2^5 \times 2^5$$

$$2 \times 2^2 \times 2^2$$

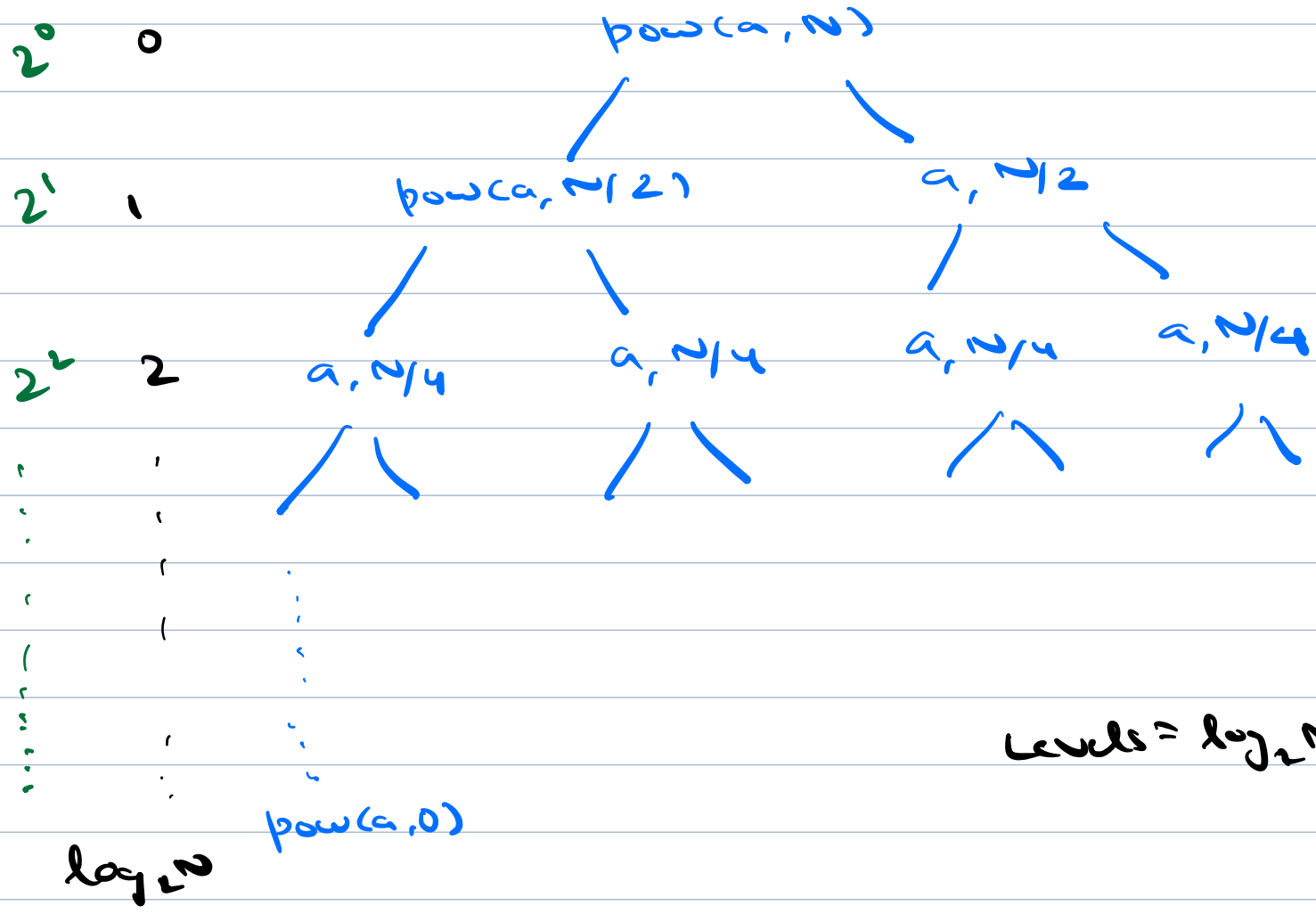
$$2^1 \times 2^1$$

$$2 \times 2^0 \times 2^0$$

$$2^{10} = \underbrace{2^5}_{32} \times 2^5$$

$$2^{10} = 32 \times 32$$

```
// Given a and n, return a^N
int pow(int a, int n) {
    If (n==0)
        return 1
    if (n%2==0)
        return pow(a,N/2) * pow(a, N/2)
    else
        return pow(a,N/2) * pow(a, N/2) *a
}
```



Levels = $\log_2 N+1$

$\log_2 N$

$2^{\log_2 N}$

Total fn calls $= 2^0 + 2^1 + \dots + 2^{\log_2 N}$

$$GP = \frac{a(r^N - 1)}{r - 1} = \frac{1(2^{\log_2 N + 1} - 1)}{2 - 1}$$

$$= 2(2^{\log_2 N}) - 1$$

$$= 2N - 1$$

$$\boxed{a^{\log_a b} = b}$$

TC : $O(N)$

SC : $O(\log_2 N)$

Fast Power or Fast Exponentiation

```
// Given a and n, return a^N
int pow(int a, int n) {
    If (n==0)
        return 1
    int p = pow(a, N/2)
    if (n%2==0)
        return p*p
    else
        return p*p*a
}
```

pow(2,13) ↑ 8192

64×64×2 ↓ ↑ 8192

pow(2,6)

8×8 ↑ ↓ ↑ 64

pow(2,3)

2×2×2 ↓ ↑ 8

pow(2,1)

1×1×2 ↓ ↑ 2

pow(2,0)

pow (a, n)

↓

pow (a, n/2)

↓

pow (a, N/4)

↓

⋮

pow (a, 0)

TC : $O(\log_2 N)$

SC : $O(\log_2 N)$

---

2. Given an array of integers, write a recursive function to print all array elements.

A = [1, 2, 3, 4, 5]

O/P : 1 2 3 4 5

→ 1 2 3 4 5

pArray (ar, 0, N-1) → 0 . . . . . N-2 N-1

pArray (ar, 0, N-2)

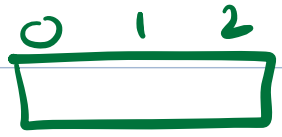
cout << ar [N-1]

// Given an ar and end idx, print ar from 0 → e idx

```
void pArray(int ar[], int e) {
    if (e == -1)        // or e<0
        return

    pArray(ar, e-1)
    print(ar[e])
}
```

ar, 2



↓

ar, 1

↓

ar, 0

↓

ar, -1

pArray(ar, N-1) (e)

↓

ar, N-2

↓

ar, N-3

↓

⋮

ar, -1

TC: O(N)

SC: O(N)

pArray(ar, 0, N-1) = print(ar[0])
pArray(ar, 1, N-1)

// Given an ar and start idx, print ar from idx
s → N-1

```
void pArray(int ar[], int s) {
    if ( s == ar.size())
        return

    print(ar[s])
    pArray(ar, s+1)
}
```

→ pArray(ar, 0)

2. Given an array of integers, write a recursive function to print all array elements.

A = [1, 2, 3, 4, 5]

Fn to print max of array

maxArray(ar, 0, N-1) = max(maxArray(ar, 0, N-2), ar[N-1])

// Given an ar and end idx, find max from 0 → e idx.

```
int maxArray(int ar[], int e) {
    if (e == -1)      // or e<0
        return INT_MIN

    return max( maxArray(ar, e-1), ar[e])
}
```

max (0 → 0)

maxArray(ar, 0) = max(maxArray(ar, -1), ar[0])

Base Case : if e == 0
return ar[0]

Handle the case of array size = 0 properly

10:40