Agenda

Prob 1 : Fast Power

Prob 2 : Print Array

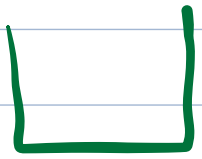
Prob 3 : Indices of an Array

Prob 4 : Check Palindrome

Revision

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

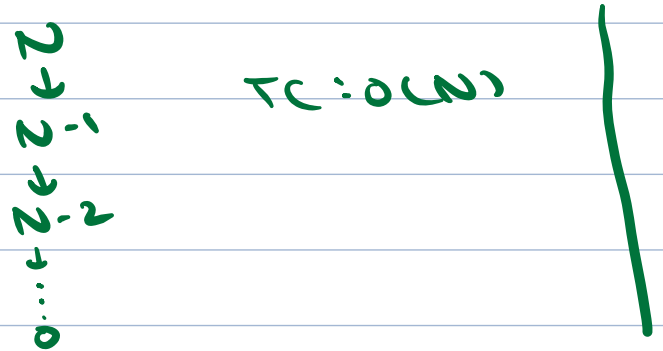
Quiz 2 : Data structure for fn call tracing : Stack



Quiz 3 : Base case for calculating factorial :

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

Quiz 4 : TC and SC for factorial (recursion)



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: $2^3 = 8$

$2 \times 2 \times 2$

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

BF : Multiply 'a' N times TC: O(N) SC: O(1)

Recursion

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

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

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

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

$a^1 = a^0 \times a$

a n

pow (5, 3) ↓↑ 125

pow (5, 2) x5 ↓↑ 25

pow (5, 1) x5 ↓↑ 5

pow (5, 0) x5

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

$a^N = a^{N/2} \times a^{N/2}$ $\quad 7^6 = 7^5 \times 7$

a^N → N is even → $a^{N/2} \times a^{N/2}$ $\quad 7^6 = 7^3 \times 7^3$

a^N → N is odd → $a^{N/2} \times a^{N/2} \times a$ $\quad 2^{10} = 2^5 \times 2^5$

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

Assumption : Given a and n, return a^n

Main Logic : If n is even

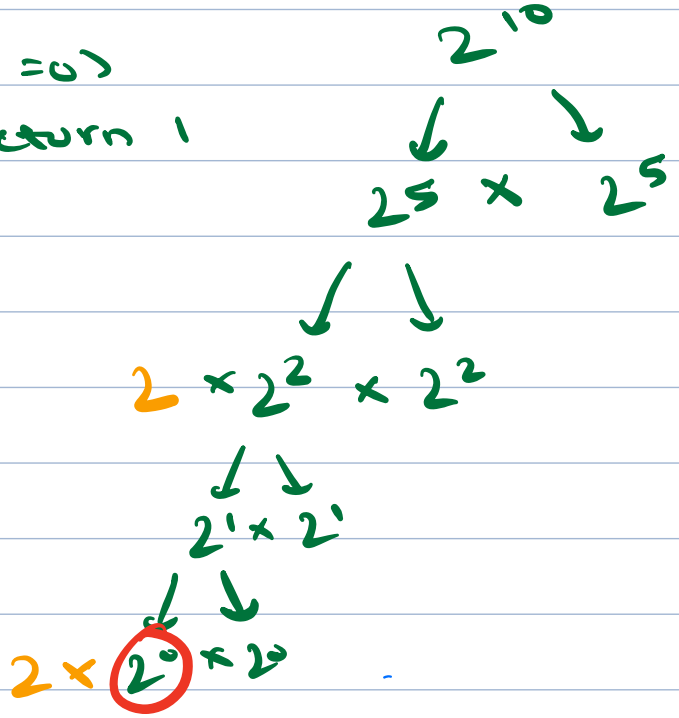
pow(a, n) = pow(a, n/2) * pow(a, n/2)

else

pow(a, n) = pow(a, n/2) * pow(a, n/2) * a

Base Case : If (n == 0)

return 1

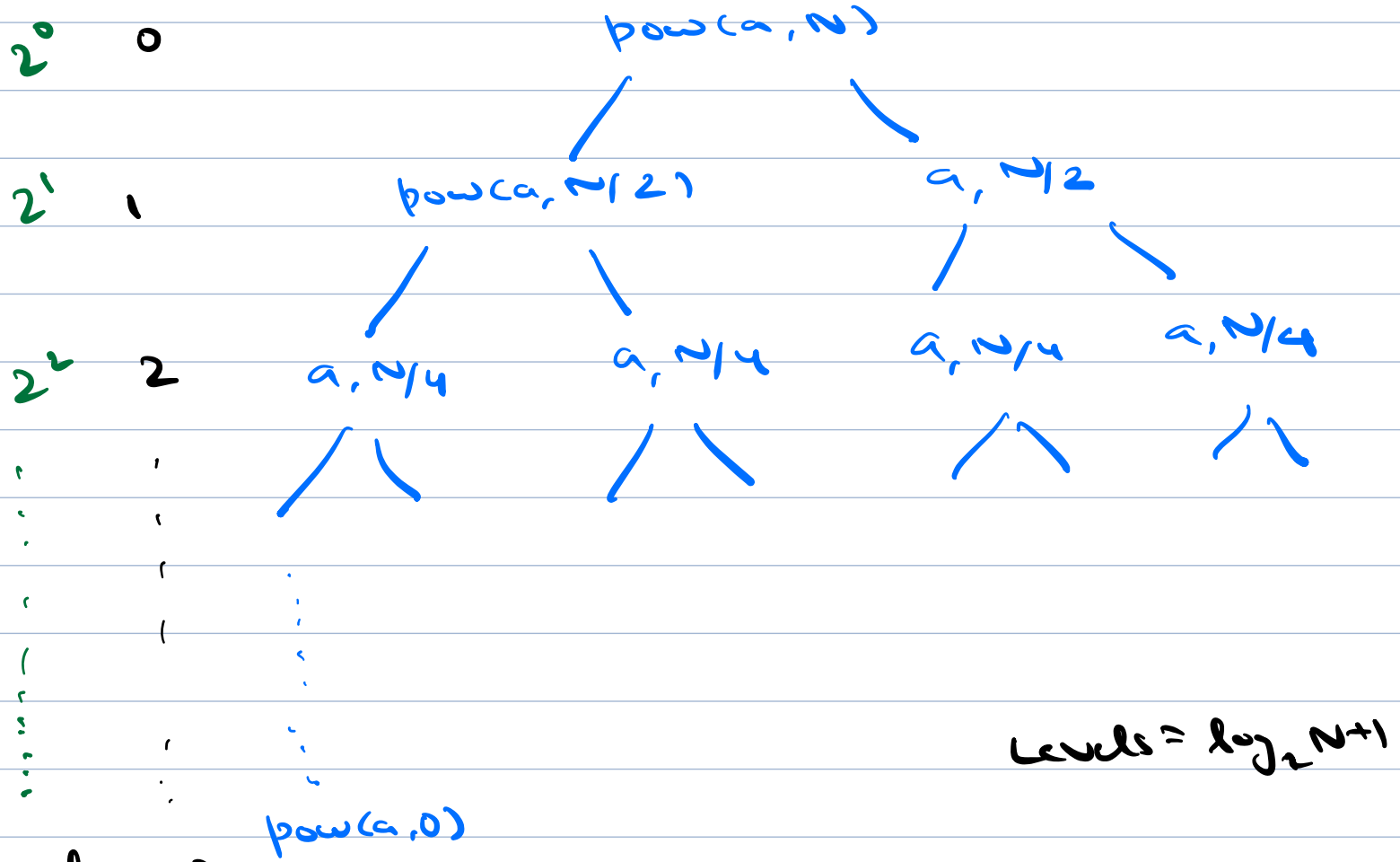


$2^{10} = 2^5 \times 2^5$

32

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

```
// 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
}
```



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

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

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

$= 2N - 1$

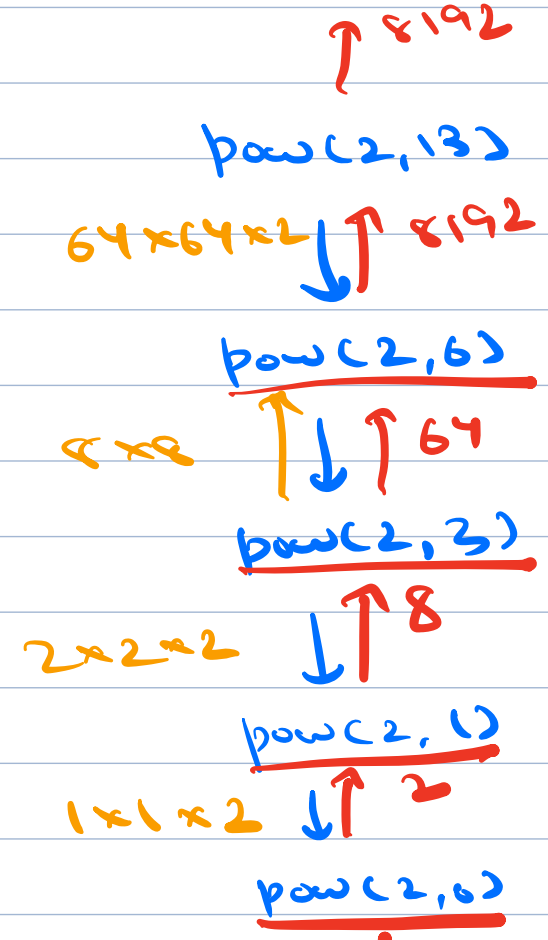
$a^{\log_a b} = b$

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 (a, n)

↓

pow (a, n/2)

↓

pow (a, n/4)

↓

⋮

pow (a, 0)

TC : $O(\log_2 N)$

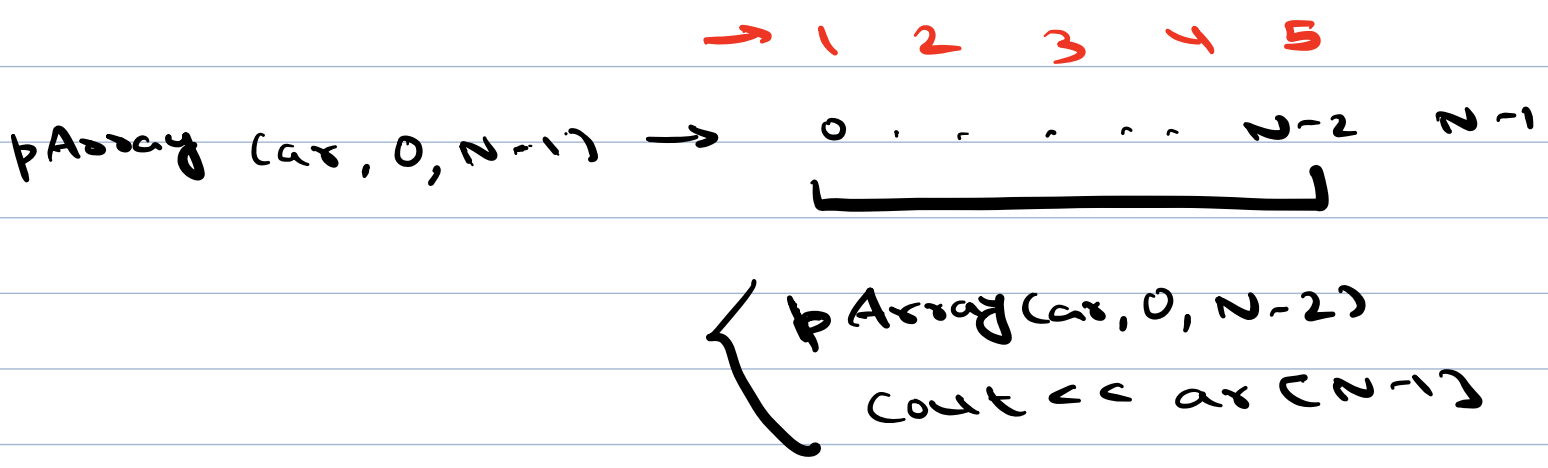
SC : $O(\log_2 N)$

---

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

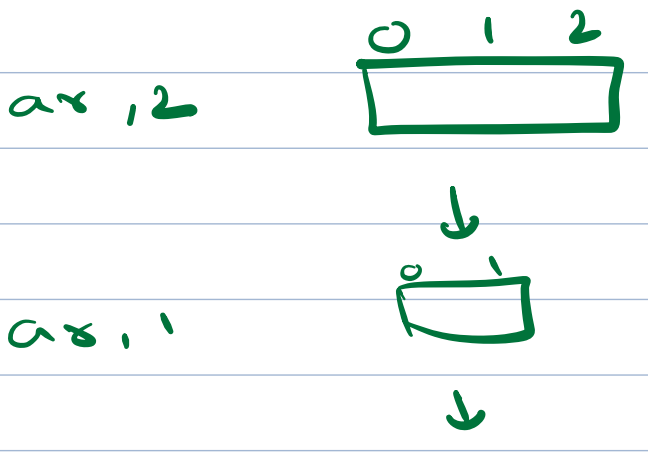
A = [1, 2, 3, 4, 5]

O/P : 1 2 3 4 5



// 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])
}
```



pArray(ar, e=N-1)

↓

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 s → N-1 idx

```
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

10:42

To find sum of array using recursion, what recursive state is to be used?

Hint: max(ar[e], maxArray(ar, e-1))

✓ a ar[e] + findsum(ar, e-1)

b max(ar[e], findsum(ar, e-1))

c ar[e] + max(ar[e], findsum(ar, e-1))

d ar[e] - findsum(ar, e-1)

3. Given an array A of N integers and target B, find all indices at which B occurs in array.

A = [4, 5, 3, 1, 5, 4, 5] (indices 0 1 2 3 4 5 6) B = 5

ans: [1, 4, 6]

A = [1, 2, 3, 1, 1] (indices 0 1 2 3 4) B = 1

ans: [0, 3, 4]

① ArrayList → GLOBAL var

② ArrayList → Input in function

getIndices(ar, 0, N-1, B):

getIndicy(ar, 0, N-2, B)

if (ar[N-1] == B)

ans.add(N-1)

// Given ar, B, and c idx, add all indices from 0 → c idx where B is present in ans

```
void getIndices(int ar[], int c, int B, list<int> ans){
    if (c == -1)
        return

    getIndicy(ar, c-1, B, ans)
    if (ar[c] == B){
        ans.add(c)
    }
}
```

N = 10


```
int main(){
    list<int> ans = new ArrayList<>
```

getIndices(ar, ar.size()-1, B, ans)

TC : O(N)

SC : O(N)
↓
stack

A = [1, 2, 3, 1, 1] B = 1
0 1 2 3 4
N = 5

e ↑ B

getIndices(ar, 4, 1, ans) ans [ ]
[0,3,4]

↓ ↑

getIndices(ar, 3, 1, ans) [0,3]

↓ ↑

getIndices(ar, 2, 1, ans) [0]

↓ ↑

getIndices(ar, 1, 1, ans) [0]

↓ ↑

getIndices(ar, 0, 1, ans) ans [0]

↑

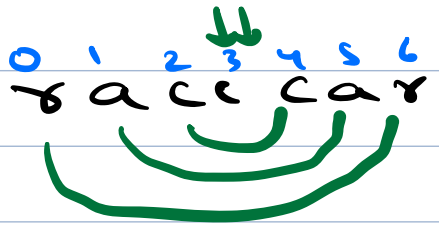
getIndices(ar, -1, 1, ans)

---

4. Given a string, write a recursive function to check if it is palindrome.

str = "radar" ans : true

str = "random" ans : false



0 → 6 s[0] == s[6]

1 → 5

// Given a str, check if its palindrome from s → e

```
bool isPalindrome(str, s, e)
{
    if (s > e) {
        return true
    }
    if (str[s] != str[e])
        return false
    return isPalindrome(str, s+1, e-1)
}
```

isPalindrome(str, 0, N-1)

N/2 fn calls

TC : O(N)

SC : O(N)

isPal (0, N-1)

↓

isPal (1, N-2)

↓

2, N-3

↓

⋮

N/2, N/2

Quiz Output for N = +3

```
void solve (int N) {
    if (N == 0)
        return
    print (N)
    solve (N-1)
    print (N)
}
```

3 2 1 1 2 3

Quiz Output for N = -3

```
void solve (int N) {
    if (N == 0)
        return
    print (N)
    solve (N-1)
}
```

solve(-3)

↓

solve (-4)

↓

solve (-5)

↓

-6

O/P

-3 -4 -5

Stack overflow

↓

MLE

① MCQs Recordings

② Visualizer Link

③ Optional class

dec N → 1

```
solve(A) {
    printNos(A)
    println()
}
```

// Given A, print all nos from 1 to A

```
void printNos(A) {
    if (A == 0) return
    printNos(A-1)
    cout << A << " ";
}
```

A = 4 1 – 2 – 3 – 4

1 – 2 – 3 – 4 –

```
printNos (num, A) {
    if (num == A) {
        cout << A; return
    }
    cout << num << " "
    printNos (num+1, A)
}
```

A –  A –

3 – 2 – 1 – 1 – 2 – 3

// Print (A) A-1 . . . 1 1 2 . . . A

```
values (A) {
    if (A == 0)
        return

    print(A)
    values (A-1)
    print (A)
}
```

(A) A-1 . . . . 1 1 . . . . A-1 A

print (A)

print (A)