Recursion

## Objectives of Today

1. Concepts of Recursion
2. Time Complexity of Recursion
3. Tracing of Recursion
4. Practice Problems

## WHAT IS RECURSION?

* Act of defining a value/problem in terms of itself.
* PROBLEM SOLUTION = SUBPROBLEM SOLUTIONS

## RECURRENCE RELATION (not related to coding)

* Mathematical way of defining a problem using a recursive formula.

## RECURSIVE FUNCTION (code related)

* A function in code, calling itself DIRECTLY or INDIRECTLY.

|  |  |
| --- | --- |
| void fun()  {  fun();  }  // DIRECT | void bun() {  fun();  }  void aun() {  bun();  }  void fun() {  aun();  }  // INDIRECT |

Repetition ===> Loop/Recursion

A function repeating itself. ===> Recursive Function

Function calling itself with different parameters ===> Recursive Function

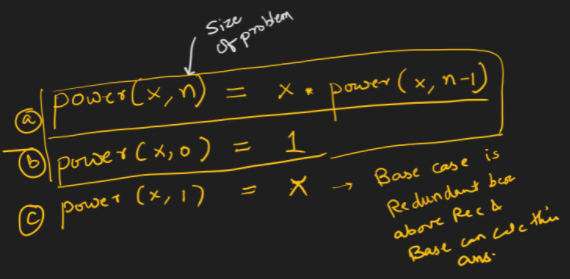
## PROBLEM OF SIZE N

What can be the size of its subproblem?

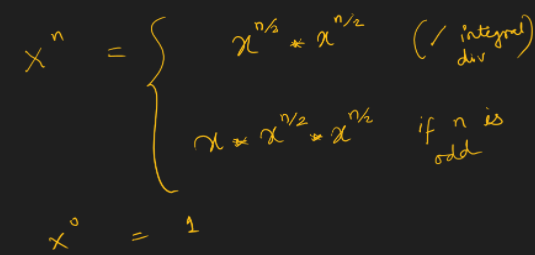
* N-1
* N-2
* N-K (K <= N)
* N/2
* N/3
* N/K (K <= N)

Recurrence Relation

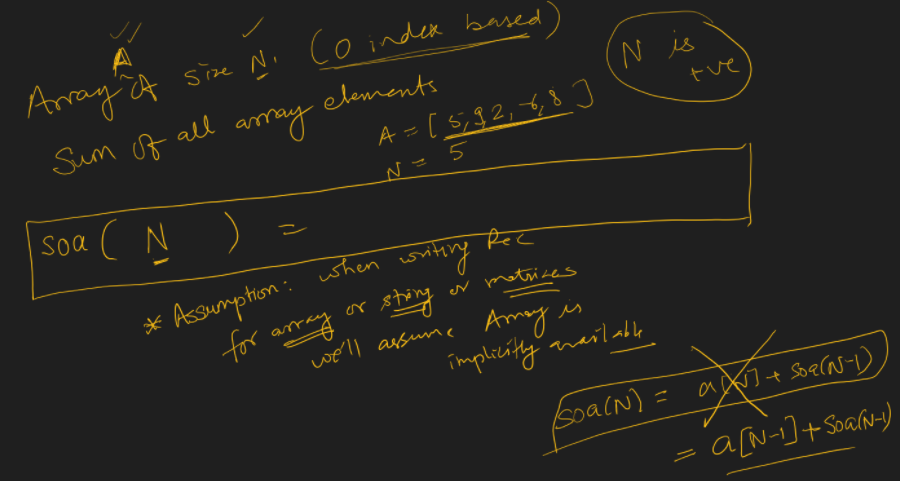
* Factorial of N
  + fact(n) = fact(n-1) \* n
  + fact(O) = 1 [BASE CASE]
* Nth Fibonacci Term
  + fib(n) = fib(n-1) + fib(n-2)
  + fib(0) = 0
  + fib(1) = 1
* X power N (Subproblem of size N-1)



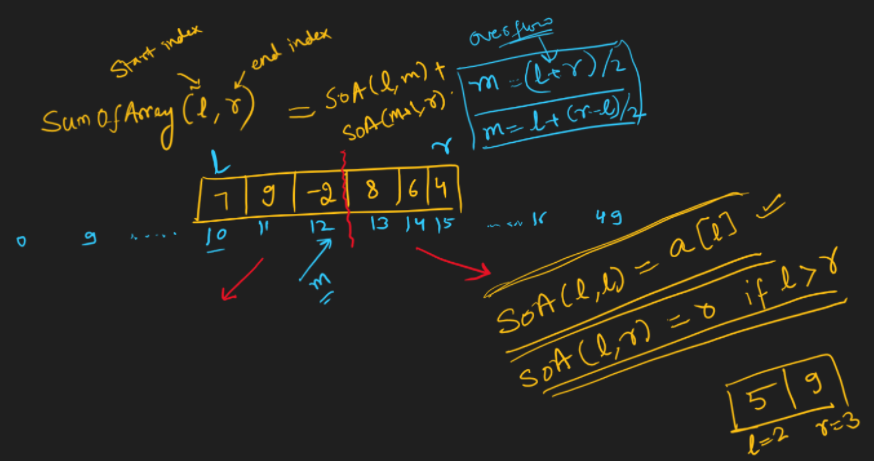
* X Power N (Subproblem of size N/2)
  + power(x, n) = power(x, n/2) \* power(x, n/2) \* x [if n is odd]
  + power(x, n) = power(x, n/2) \* power(x, n/2) [if n is even]
  + power(x, 0) = 1 [Base Case]



* Sum of Array Elements (For all positive values of N)
  + sumOfArray(N) => Means sum of array elements considering array size as N
  + sumOfArray(N) = a[N-1] + sumOfArray(N-1)
  + sumOfArray(0) = 0



* Divide and Conquer
  + sumOfArrayElements(l, r) = sumOfArrayElements(l, m) + sumOfArrayElements(m+1, r)
  + sumOfArrayElements(l, l) = a[l]
  + sumOfArravElements(l. r) = 0 if l > r



* When we convert recurrence relation to CODE
  1. Assume Array is declared globally.
  2. Just translate relation to code.

**[PROBLEM]** Given X and N, Find X power N using a function.

long long power(int x, int n) {

}

1. Built-In Function
2. Iterative Loop Way (ans = 1, in loop that runs n times make ans = ans \* X) [TC: n]
3. Recursive Code (using recurrence power(x, n) = power(x, n-1) \* X)
4. Recursive Code (using recurrence power(x, n) = x \* power(x, n/2) \* power(x, n/2) for n odd and for even removing x.
5. 4th approach, by solving only one subproblem.

|  |
| --- |
| [APPROACH 4]  long long power(int x, int n) {  if (n == 0) return 1;  if (n % 2 == 1)  return power(x, n / 2) \* power(x, n / 2) \* x;  return power(x, n / 2) \* power(x, n / 2);  } |
| [APPROACH 5]  long long power(int x, int n) {  if (n == 0) return 1;  long long hf = power(x, n / 2);  long long sq = hf \* hf;  if (n % 2 == 1) return x \* sq;  return sq;  } |
| [APPROACH 3]  long long power(int x, int n) {  if (n == 0) return 1;  return x \* power(x, n-1);  } |

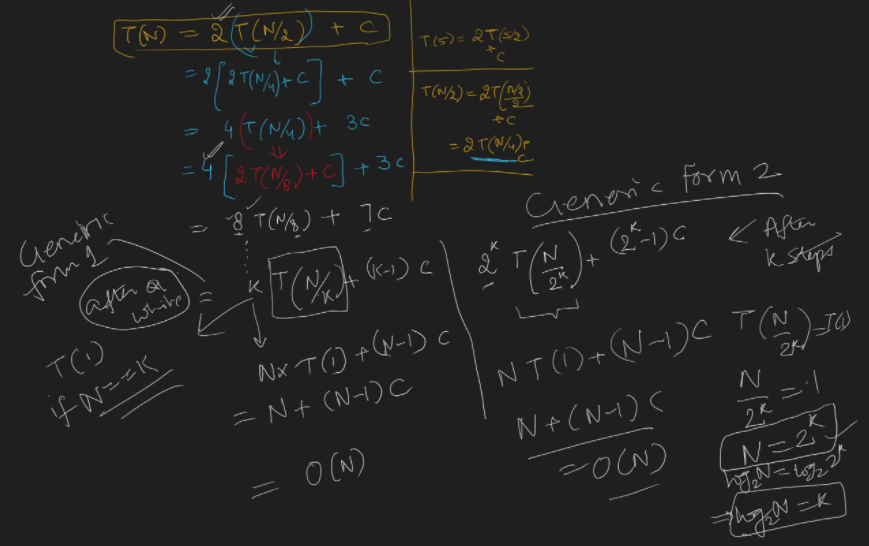
* Time Complexity of Recursive Code is Found by **Solving Recurrence Relations**.
* We need to define the time complexity of our function in **terms of problem size**.

## PROCESS

1. Expand the Recursion on Right Side.
2. Generalize the Right Side.
3. Get Rid of recursive term on right side by considering T(0) = 1, T(1) = 1.

**[APPROACH 4]**

T(N) = 2 \* T(N/2) + c

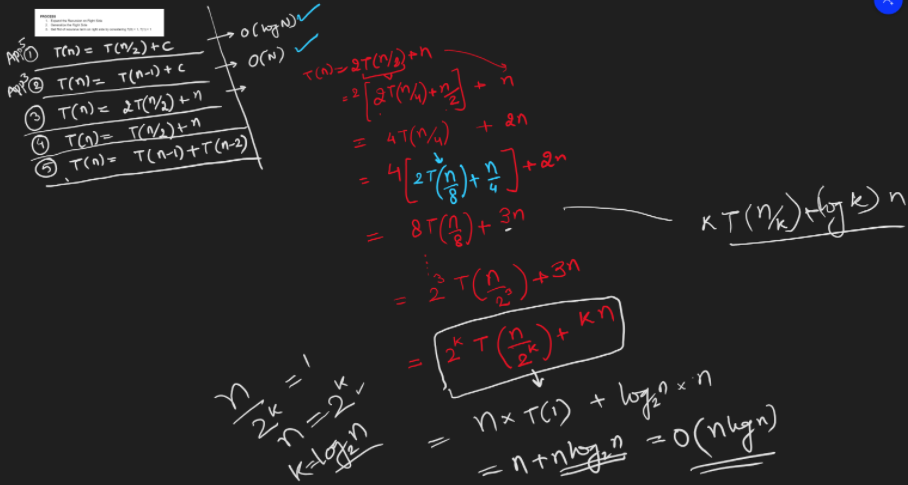


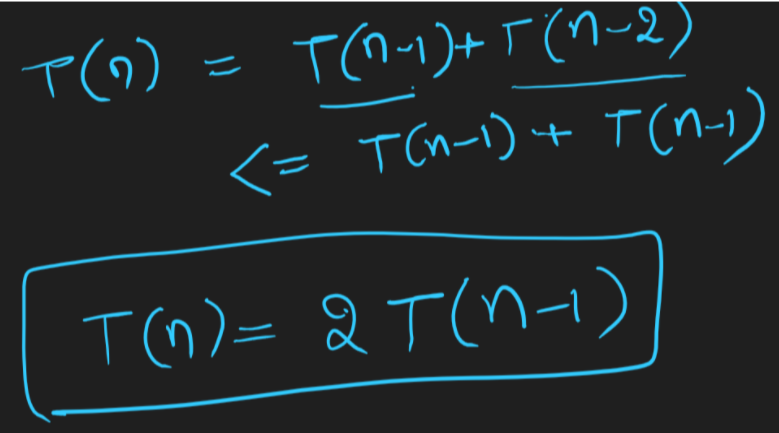
**[APPROACH 5]**

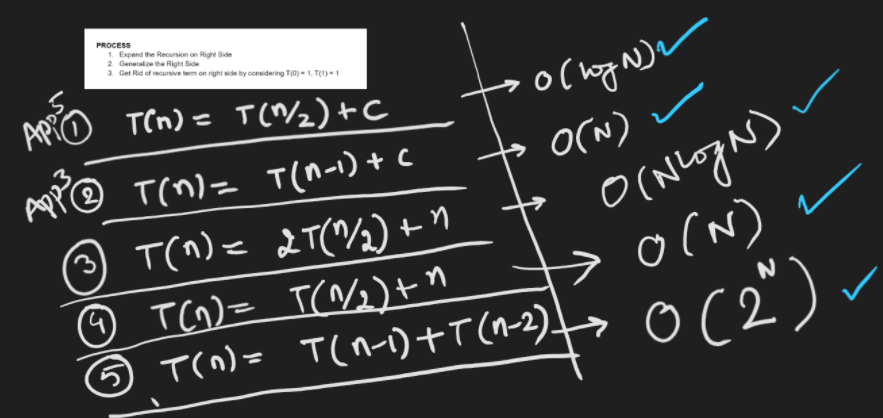
T(N) = T(N/2) + c

**[APPROACH 3]**

T(N) = T(N-1) + c







* Why is recurrence for time complexity of below code?

**T(N) = 2 \*T(N/2) + c**

X = 3

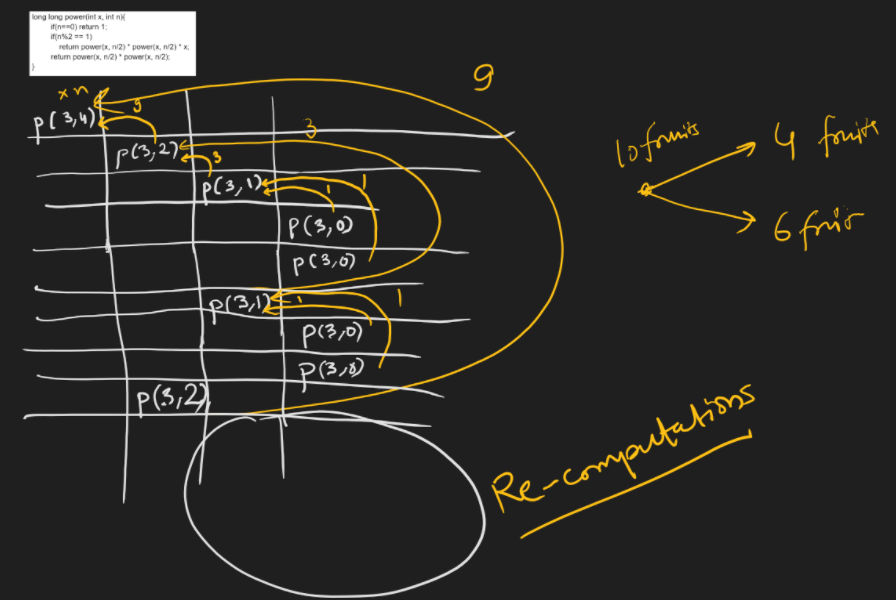
N = 4

|  |
| --- |
| [APPROACH 4]  long long power(int x, int n) {  if (n == 0) return 1;  if (n % 2 == 1)  return power(x, n / 2) \* power(x, n / 2) \* x;  return power(x, n / 2) \* power(x, n / 2);  } |
| [APPROACH 5]  long long power(int x, int n) {  if (n == 0) return 1;  long long hf = power(x, n / 2);  long long sq = hf \* hf;  if (n % 2 == 1) return x \* sq;  return sq;  } |

## Visualization

x=3 n=4

1. TABLE METHOD



1. TREE METHOD