Divide and Conquer:

Ex: BinarySearch

Sort the array

↳ n==1
   ↳ Small problem
      ↳ return arr(i)

n>1
   ↳ big problem
      ↳ Divide & conquer approach

Steps:

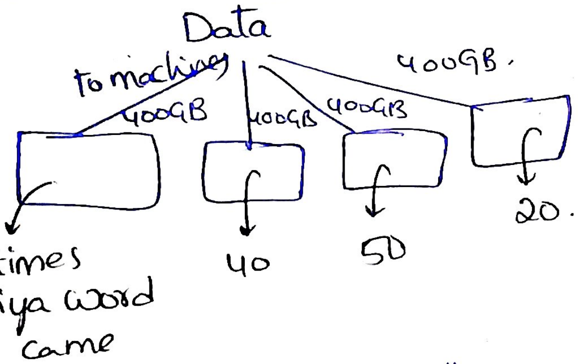
1. Divide the problem into various subproblems.
2. conquer each Subproblem
3. combine all the Solution. (optional)

Example in Real World:

① Data Engineering | Big data:

let us take an Example:

find count of 'priya'

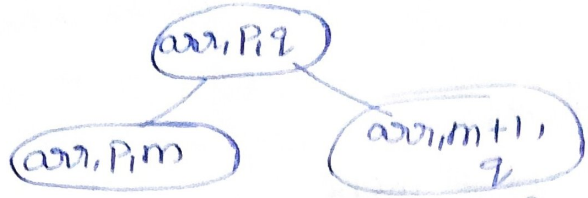


Total Count = 30 + 40 + 50 + 20 = 140

This process is called 'Hadoop' (map reduce) in Big data.

Pseudo Code:

```
divideAndConquer (arr, P, q):      # P -> start Index, q -> end Index
    if (Small (arr, P, q)):
        return Solution
    else:
Divide      m = divide (arr, P, q)
Conquer     b = divideAndConquer (arr, P, m)
            c = divideAndConquer (arr, m+1, q)
Combine     return combine (b, c)
```



Ex: Finding maximum and min. in an array.

arr = [75, 45, 95, 50, 60, 67, 29, 32]
0 1 2 3 4 5 6 7

Approach-1: Taking two pointers. → O(n)

max = ~~75~~ 95
min = ~~75~~ ~~45~~ 29

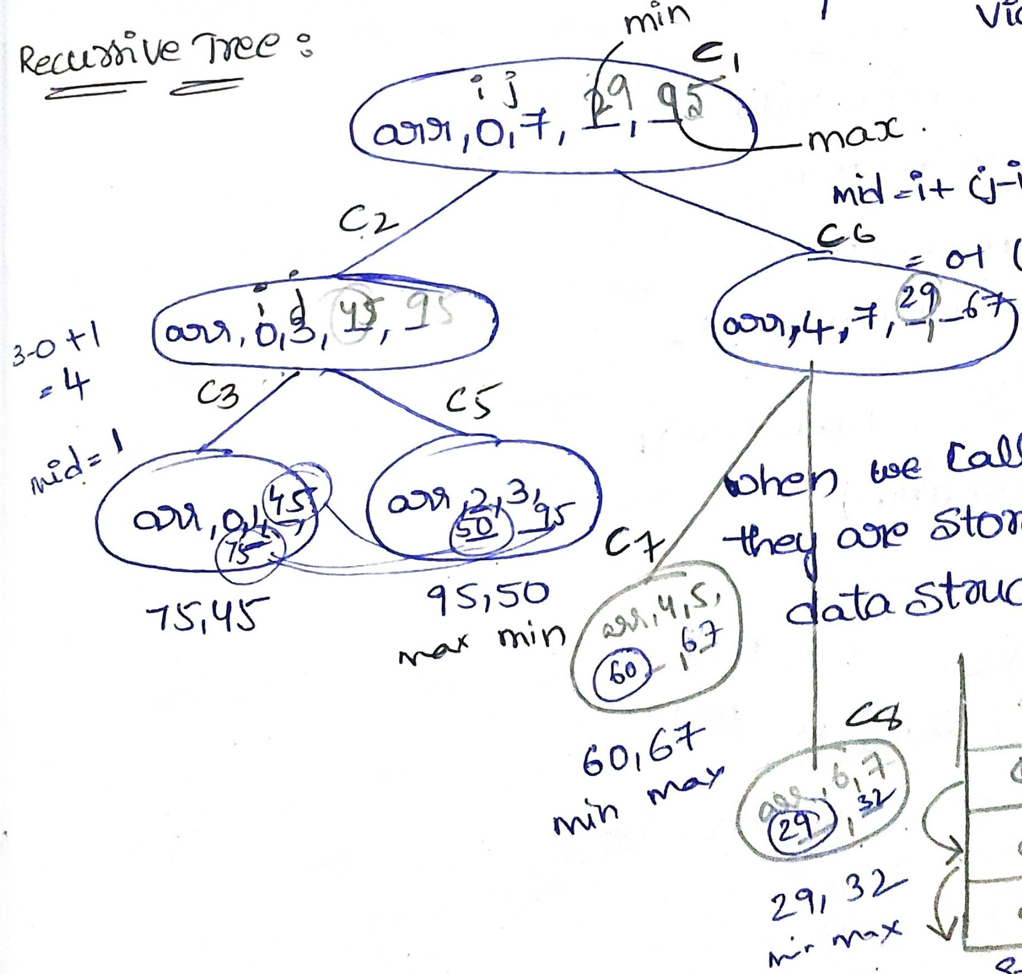
Approach-2: Divide and Conquer.

We use divide and conquer technique to solve large problems by simplifying them into small problems.

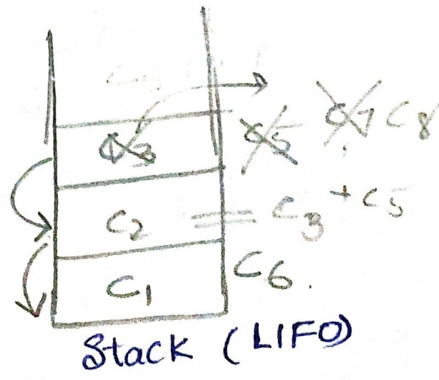
Small-problems: ① n==1
max = arr[i]
min = arr[i]

n==2
if arr(i) < arr(j)
max = arr(j)
min = arr(i)
else
vice-versa

Recursive Tree:



When we call a function, they are stored in a stack data structure.



Pseudocode :

```
                          T(n)
find max and Min (arr, i, j):
    if i==J: # single element
        min = arr(i)
        max = arr(i)
    elif i==J-1; # two element
        if arr(i) < arr(j):
            max = arr(j)
            min = arr(i)
        else:
            max = arr(i)
            min = arr(j)
    else: # more than two element
        mid = i+(j-i)//2  -> Divide (O(1))
        min1, max1 = find Max and min (arr, i, mid)   T(n/2)    } -> Conque
        min2, max2 = find max and min (arr, mid+1, J)  T(n/2)   }   2T(n/2)
        if min1 < min2:             }
            min = min1              }
        else min = min2             }  Combine
        if max1 < max2              }    -> C
            max = max2              }
        else:  max = max1
    return (max, min)
```

Recurrence Relation :

$$T(n) = \begin{cases} c & n \leq 1 \\ 2T(\frac{n}{2})+c & n>1 \end{cases}$$

By using master's Theorem, we Solve.

$a=2 \quad k=0$

$b=2 \quad p=0.$

$\log_b^a = \log_2^2 = 1$

$\log_b^a > k \Rightarrow O(n^{\log_b^a}) = O(n)$

Analysis:

Brute-force : T(C) = O(n)

Divide & Conquer. T(C) = O(n)

Recursion uses Stack.

★ Stack Space = # levels in recursive tree = O(log n).

Q). Which is better? Why?

for medium- small size array → Brute-force

for large size array → D&C

Q). Finding power of an element? - Amazon interview Question

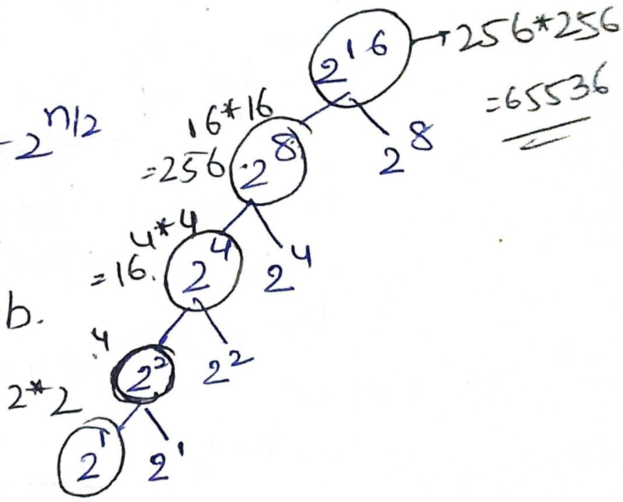
$\begin{cases} n=16 \\ a=2 \end{cases} \Rightarrow$ find $2^{16} = 2^{8} * 2^{8}$

* $n \rightarrow$ even

Divide & Conquer $2^n = 2^{n/2} * 2^{n/2}$

$2^{n/2} \Rightarrow b$

$\therefore 2^n = b * b.$



Small problem

↳ $n = 0 \Rightarrow 1$

↳ $n = 1 \Rightarrow a^1 = a$

* $n \rightarrow$ odd

$2^{17} = 2^{16} * 2 = 131072$

Divide & Conquer.

$2^{47} = 2^{46} * 2$

$2^{14} = 2^{7} * 2^{7}$

$2^{6} * 2$

$2^{3} * 2^{3}$

$2^{2} * 2$

$2^1 * 2^1 * 2^1$

if $n = -2$, $a = 2$ $\therefore 2^{-2} \Rightarrow \frac{1}{2^2}$

use divide & Conquer

Pseudocode:

```
findpowofele(a,n):
    if n==1:
        return a
    elif n==0:
        return 1
    else:
        mid=n//2          # Divide
        b=findpowofele(a,mid)  # conquer
        result=b*b        # combine
        if n%2==0:
            return result
        else:
            return result*a
```

a=2
n=16

findpowofele(2,16)
256*256

b= findpowele(2,8)
16*16=256

b= findpowele(2,4)
4*4=16

b= findpowele(2,2)
2*2=4

b= findpow(2,1) {small problem
↳ 2

Skewed Recursive Tree

Stack Space = O(n)

CBT Recursive Tree

Stack Space = O(logn)
↳ Best & Average

★ In Recursion → n is very high → Recursion in depth exceeded
↳ we overcome this by "Dynamic programming"

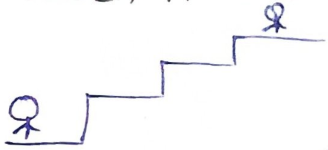
$T(n) = T(\frac{n}{2}) + c$

$= O(\log_2^n)$

Q). Count no. of ways to reach nth stair

one step at a time

two steps at a time



n = 1

output = 1

n = 2:

output = 2

(1,1) and 2

we

n = 3,

(1,1,1)

(1,2)

(2,1)

n = 4

o/p = 5

(1,1,1,1) (2,1,1)

(1,1,2), (2,2)

(1,2,1)

0	1	2	3	4	5
	1	2	3	5	8

} Fibonacci series.

∴ ways(n) = ways(n-1) + ways(n-2)

Divide & Conquer :

```
if n<=2:
    return n.
else: return
    self.fun(n-1) +
        self.fun(n-2)
```

DP version

```
if n<=2:
    return n
else.
  first, second = 1,2
  for i in range(3,n+1):
      third = first + second
      first, second = second, third
  return second
```

* if 'n' value increases.
  → we encounter overlapping subproblem.

* To avoid overalapping subproblem, we use dynamic programming.

	2	3	4	
1	2	3	5	8

first second second

first

n=4 i=3

third = 1+2 = 3

first, second -

2 3

i=4

third = 2+3 = 5

first, second

3 5

return 3