Recap: Time Complexity

Code → depending on input size, you get $f(n)$ ⇒ # steps code has to perform to produce o/p.

Once I have $f(n)$

$f(n) < 10^8$ for this code to finish in 1s.

$f(n)$ → n_{max}

Eg. $f(n) = n^2$ $\quad n_{max} = 10^4$

Easy for us to compare codes.

Thumb: Smaller $f(n)$, better is code. Better = Faster.

Problem: Bulbs toggle problem

Q. n bulbs ⇒ array [0, 1, 1, 0 ⋯] → 1 = ON, 0 = OFF.

q tasks ⇒ Each task (l, r) → from l to r, toggle bulbs.

Eventually o/p: Total number of lit bulbs.

Code1

1. For each task : ⟶ #q times
   get l and r
   From l to r
   toggle the bulbs. } O(n)

sum(bulbs)

∴ $O(nq)$ → TC

$f(n,q) = nq$ $\quad n\uparrow\ q\uparrow$, 10^8 first

$g(n,q) = n+q$ $\quad t$

TC → $O(n+q)$ ⟹

DEMO: g → 0.1 s
f → 100 s.

Code2

1. diff = [0] * (n+1)
   For each task :
   get l and r
   diff [l] += 1
   diff [r+1] −= 1

2. prefix [0] = diff [0]
   for i = 1, n :
   prefix [i] = prefix [i−1] + diff [i]

3. Only toggle bulb [i] if prefix [i] is odd.

4. sum(bulbs)

DATA STRUCTURES.

store alot of data.

Eg marks of entire class.

"Prog. lang"

Non Sequential datatypes

- Integers
- Floats
- Doubles
- Booleans
- characters

Sequential DT

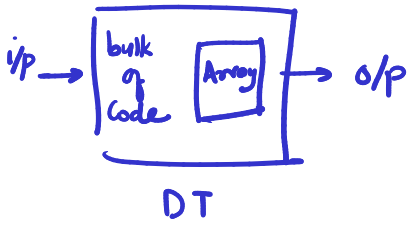
① Arrays

Derived DT

② Sets
Maps/Dicts

③ Stacks
queues

} available to you in your collections/ templates.



Advanced DS ⟹ DT which are not available in-built.

① We want to know everything ⟹ arrays!!

2 questions → arrays.

Array → sequence of non-sequential datatypes.

Array of chars ⇒ string.

What all can you do with a sequence?

1. Sort it → bring it to a certain order
2. Searching → alot many numbers, find if x exists
3. Rotations →  ⟹ rotation of 2 elements

4. Two pointer method — i,j → 2 indices
5. Sliding window technique — i,j + elements bef: index i,j.

6. Prefix sum + Diff arrays.

(A) SORTING. → many computer sci. spent lives on sorting.

Def^n: Ascending order (default sorting) Eg. 1 4 3 2 5 6 8 ⇒ n nums

⇓ sort

1 2 3 4 5 6 8 ⇒ n nums

increasing order.

Brute force / Unoptimized:

Logic: Run a loop n-1 times.
Each time I will bring the max element at the end.

Bubble sort. Just like bubbles, the largest bubble surfaces first.

Eg.2 5 6 3 1 2 → n=5

4 times:

#1 5,3,1,2 | 6

#2 3 1 2 | 5 6

#3 1 2 | 3 5 6

#4 1 2 3 5 6

```
for (int i= 0 ;  i< n ;  i++)
  for (int j= 1 ;  j< n-i ;  j++)
      if arr[j] < arr[j-1] :
          swap (arr[j], arr[j-1])
```

T.C = n-1 + n-2 + n-3 + ⋯ + 1 = $\frac{n(n-1)}{2}$

T.C ~ $O(n^2)$ This code can sort an array of max 10^4 ele.

$n^2 < 10^8$ ∴ $n < 10^4$ $n_{max} = 10^4$. in time limit of 1 s.

Agenda: $n_{max} \uparrow$ 10^4 is not enough! Optimize.

Redundant Work.: Identify / Eliminate.

This is not how sorting works naturally. ⇒ Largest element is given very high priority. | I want: Every element should be treated, not just largest.

To do so, we need Recursion.

Every recursion has 3 parts.

Recursion:

```
f(n):
  ||
  ||
 f(n')
  ||
  ||
```

→

```
func(n):
// Base cond^n → smallest value of n for which f(n) is known
// Problem decomposition
   ↳ func(n')
```

Mandatory ⇒ $n' < n$

Eg. (1) Factorial

$n! = n \times n-1 \times n-2 \times n-3 \dots \times 2 \times 1$

$n! = n \times (n-1)!$ ⇒ Recurrence Relation // Recomposition

Utilize the o/p from func(n') to produce o/p for func(n)

$0! = 1$ ← Why?

$1! = 1$

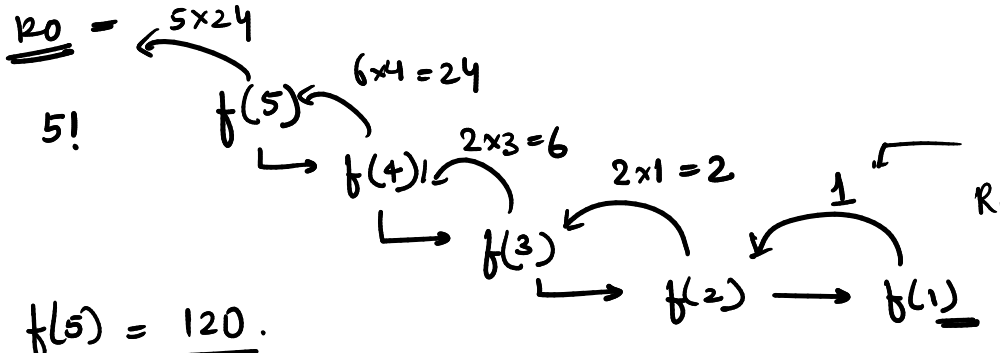
```
int fact(n):
    if n==0 || n==1 :
        return 1          ← Base

    int temp = fact(n-1)  ← Decomposition

    return n* temp        ← Recomposition
```

return n* fact(n-1)

↙ → Initially go explicit.



Base cond^n hits & Recomposition starts.

Till this point decomposition happens.

$f(5) = 120$.

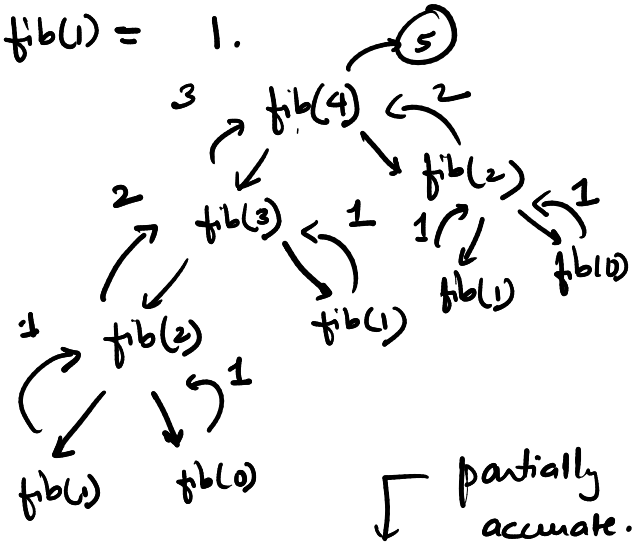
T.C. ~ O(n) ∵ the call chain is linear 5 → 4 → 3 → 2 → 1

TC : 2n ~ O(n)

eg.② Fibonacci numbers. * $fib(n) = fib(n-1) + fib(n-2)$ | Def^n:

$fib(0) = fib(1) = 1.$

```
int fib(n):
    if n==0 || n==1 : return 1
    int a = fib(n-1)
    int b = fib(n-2)
    return a+b
```



partially accurate.

Things starts getting messier. Tree recursion. $O(2^n)$

$n < 40$ for sure !! At $n=51$, computer hangs !!

Better idea to visualize recursion ? Yes !! $n = 100$ | Actual recursion ↓ Not visualized.

Mathematical Induction.

Eg Φ. $1+2+3+\ldots.+n = \frac{n(n+1)}{2}$ Prove that it works !!

Step 1: 1 element. $n=1$ LHS = 1 RHS = $\frac{1(1+1)}{2} = 1$ ∴ LHS = RHS

Step 2: Assume for $n=k$ it is true.

$$1+2+3+\ldots.+k = \frac{k(k+1)}{2} \quad \text{——} \quad ①$$

Step 3: $n = k+1$ LHS = $1+2+3+\ldots+k+k+1$

$$= \frac{k(k+1)}{2} + \frac{(k+1)2}{2}$$

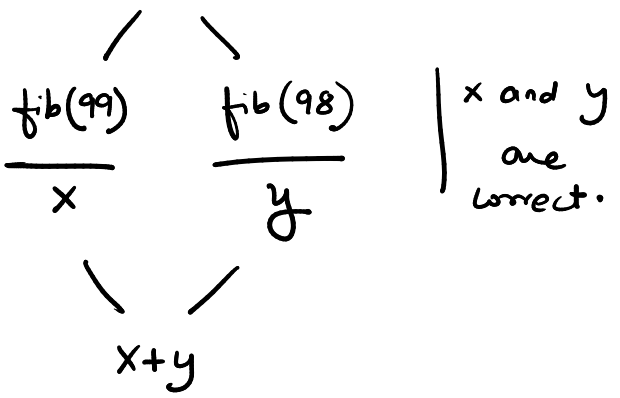
$$= \frac{(k+1)(k+2)}{2}$$

$$RHS = \frac{(k+1)(k+1+1)}{2} = \frac{(k+1)(k+2)}{2} \qquad LHS = RHS.$$

$\therefore \quad 1+2+3+\dots+n = \frac{n(n+1)}{2}$ Hence Proved.

Some Idea in Recursion.

1. fib(100)



x and y are correct.

Step1: ⇒ fib(1) / fib(0) = 1

Step2: Assume you know the ans for f(99) and f(98).

Step3: If you can recompose properly using the assump. your recursion will work.

Induction. ⟶ TRUST THE PROCESS.

Eg. ③ fact(5)

Step 1: fact(0) = fact(1) = 1

Step 2: fact(4) = 24 | assume you get this.

fact(4) returns 24.

↳ assume this! Don't ask how!

a = 24.

Step3: Ans = 5 * 24 = 120

```
int fact(n):
    if n==0 || n==1 :
        return 1
    int a = fact(n-1)
    return n*a;
```

WILL WORK!!

return n*a;

n = 100 × 6 fact(6) ⟵ 720 ans = n* a

Where are we going to use this idea?

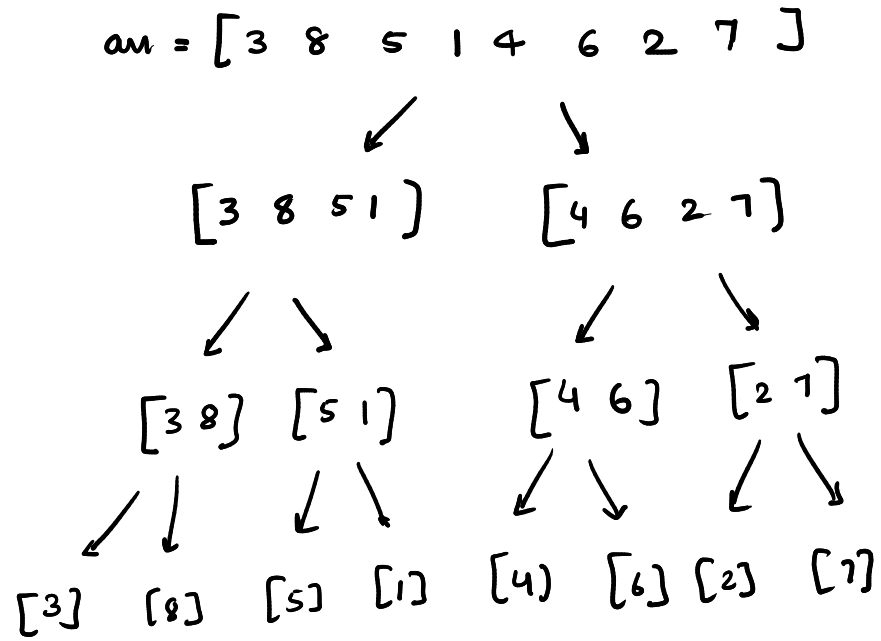
Power of this idea: True Rec.

sort(arr): → Expect: give me numbers in ascending order.

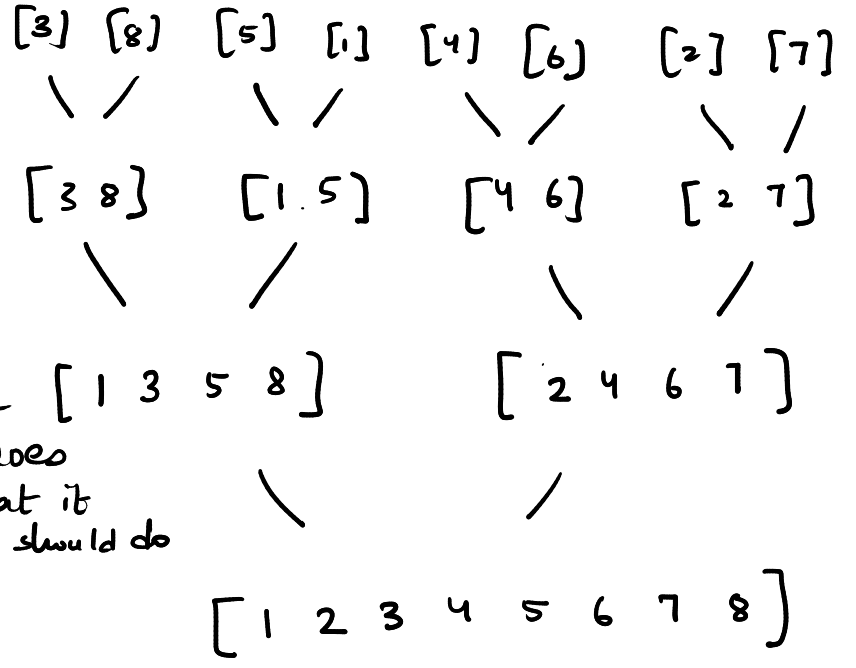
Can I code this idea? No!!

Unless I use induction.

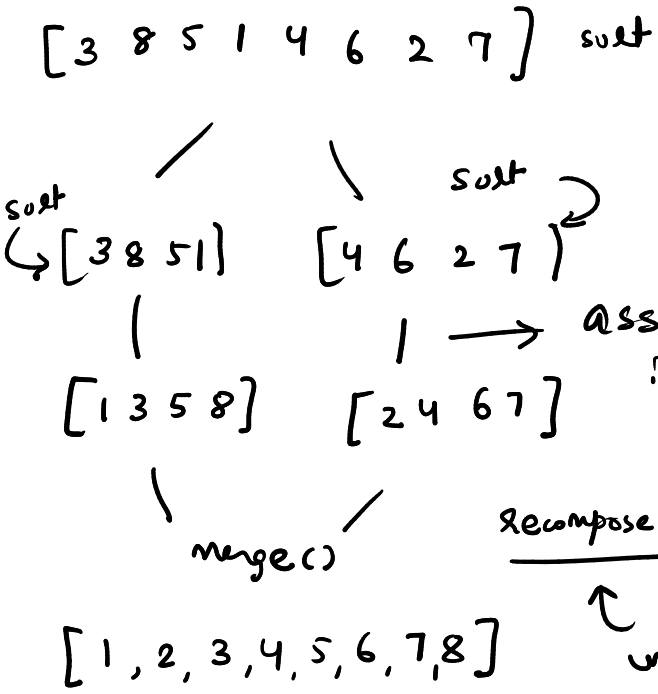
```
sort (arr):
    left_half  = arr[0: mid]
    right_half = arr[mid+1 : n]
    sort (left_half)
    sort (right_half)
    merge (left_half,
           right_half)
```



Base cond?: Single element arrays are always sorted.



Inductive Rec:



assume it does what it should do → Recompose ? ↑ where your genius lies.

Sorted

merge: It takes 2 sorted arrays. You want to give 1 sorted array (merged) back.