DYNAMIC PROGRAMMING

Solution ways:

-> Tabulation —> Bottom up

-> Memorization -> Top - Down

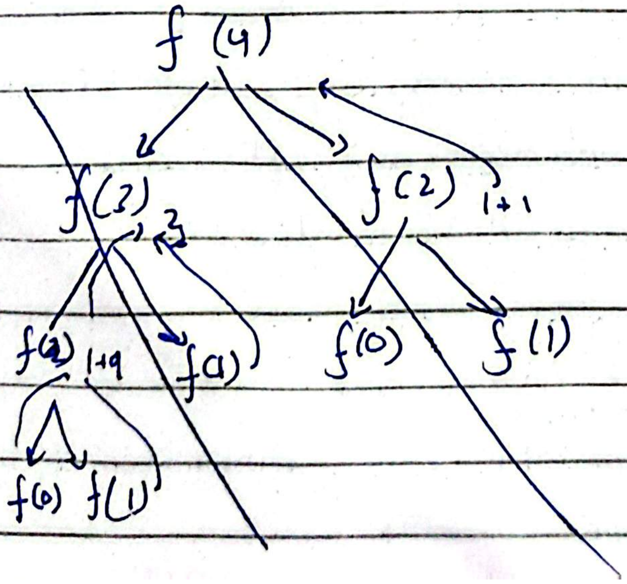
Fibonacci:

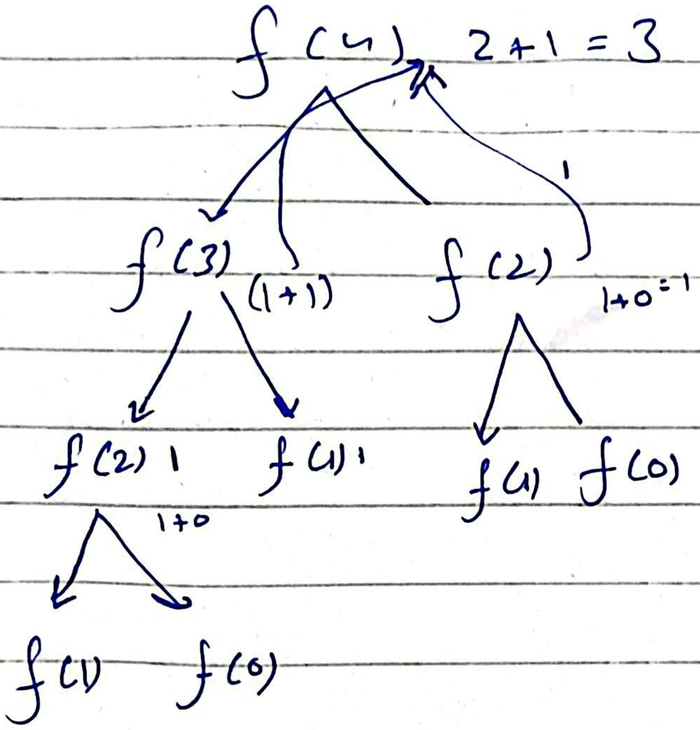
0 1 1 2 3 5 8 13

every number is sum of 2 numbers before it.

for f(4) = 3

code in file to find any number in fib sequence.





If is a waste of time in a way cuz there is no need of recomput f(2) cuz along the way it is calculated in the process in left side we should not re compute it right side again.

So thats' called an overlapping subproblem. this is where memorization jumps in.

Memorization :

tend to store the value of subproblem in some map/table.

if we create an array and.

dp = [_ _ _ _]

So when fib(2) is calculated
it will store

dp = [_ _ 1 _ _]

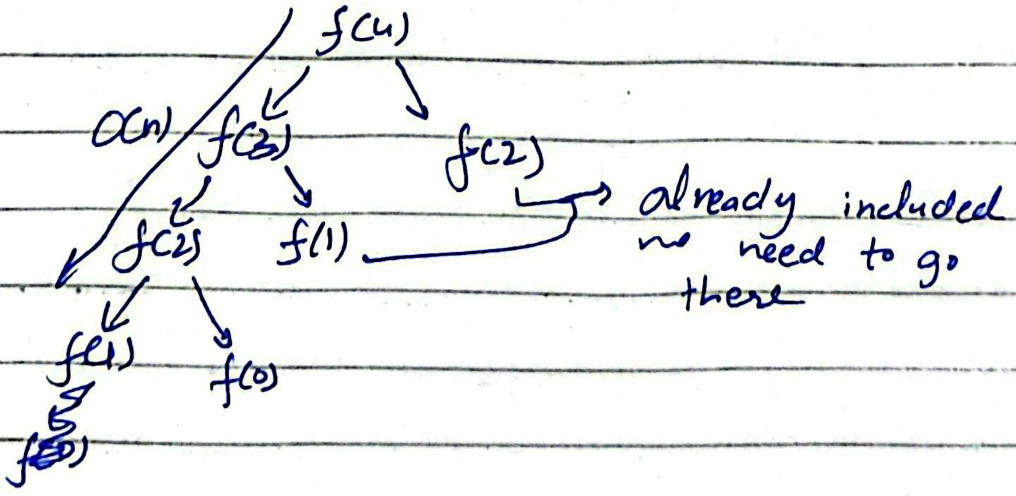
So when next time we are going on fib(2) it will see if it is already in the storage.

code in file.

Also

TC is reduced to O(n)

cuz now tree is like



Also

$$SC = \underset{\text{recursion stack.}}{O(n)} + \underset{\text{array}}{O(n)}$$

Recursion → Tabulation. (Bottom-up)

top→(Down)

answer ↘ Base case ↗ then come back

Base case (to the required answer)

in order to convert to tabulation we do this.

```
dp[0] = 0
dp[1] = 1
```

} → base case.

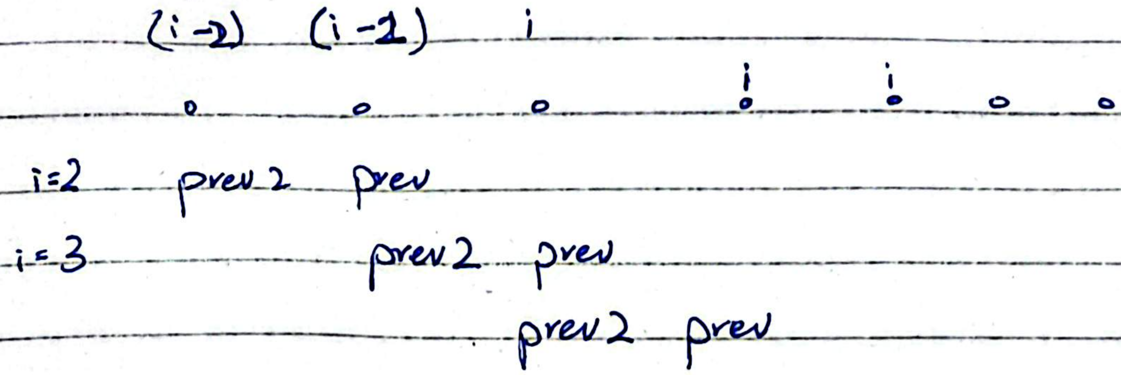
Now from 2 and onward.

```
for (i=2→n){
    dp[i] = dp[i-1] + dp[i-2]
}
```

TC → O(N)

SC → O(N) → we want to minimize this as well.

for that for any i

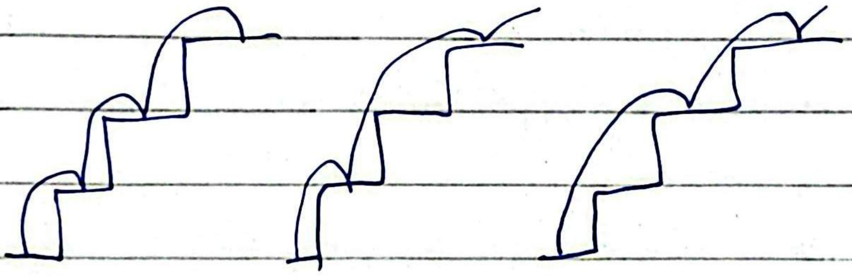


So we dont need an array as well we just need two variables to store prev and prev2.

DP 2 Climbing Problems

Distinct ways to reach n^{th} stair
like

n=3



output=3
initially we are at zero we have to reach n^{th} stair

1D dp problems:

How do we understand How is this dp problem?

Problem statements will be like :-
-> Count the total no of ways.
-> Minimum output.
-> Max output.
-> try all possibleway comes in
-> count, bestway

Thats when we apply recursions.

Short cut trick:

1) try to represent problem in terms of index
2) do all possible stuff on that index according to problem statement
3) Sum of all stuffs -> count all ways.
   min (of all stuffs) -> find min

f(n) -> no of ways (0 -> n)

```
f ( idx) {
    if (idx == 0) return 1;
    if (idx ==1 ) return 0;
either jump1 or jump 2

  l = f (idx-1)
  r = f (idx-2)

    return l+r;
  }
```

Frog jump

10	20	30	10	0->3
0	1	2	3	index

frog wants to go from 0->3

i+2 or i+1

if $10 \xrightarrow{10} 20 \xrightarrow{10} 30 \xrightarrow{20} 10$ = 40

if $10 \xrightarrow{10} 20 \xrightarrow[30]{10} 10$ = 20 -> this is min.

if $10 \xrightarrow[20]{20} 30 \xrightarrow{20} 10$ = 40

~~if~~

10 — 20 — 30 — 10

As we know if we are trying all ways then its a recursion problem.

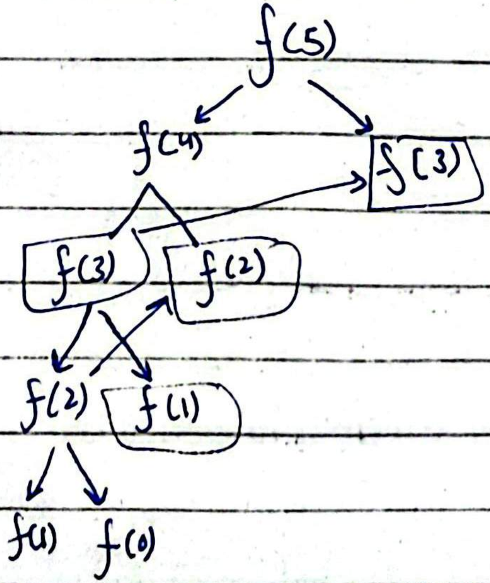
So in recursions steps are

① express the problem in terms of index
② do all possible stuffs on that index
③ take minimal of all stuffs.

```
f(idx)
  if(idx==0) return 0;

  l= f(idx-1) +abs[arr[i] - arr[i-1]]
if( r= f(idx-2) + abs(arr[i]- arr[i-2]))
  (idx >1)
  return min(l,r)
```

Now for Memorization just include the min for that index like



if for idx 3 or 2 or 1 we have already stored min then no need to go there.

Recursions to DP steps

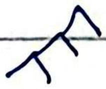
Memorization

① look at what parameter is changing. like idx in this problem.

② Before returning add it up

③ whenever call a recursion first check if it is already computed.

As in recursion we were going top to down.



But in tabulation we do Bottom up

```
first    dp = [] * n+1
  Base case  idx== 0
        so
            dp[0] = 0
then
      for (i=1 → n)
        { first step = dp[i-1] + abs(arr[i-1] - arr[i])
              if i>1:
              ss = dp[i-2] + abs(arr[i-3] - arr[i-2])
              dp[i] = min (first, second) }
```

More optimization

dp1=0

dp2=0

i=1

10 $\overset{i}{20}$ 30 10

curr=10 dp1=10

dp2=0

10, 20 $\overset{i}{30}$ 10

dp1=10

dp2=10

curr=10

10 20 30 $\overset{i}{10}$

dp2=10

dp1=20

curr=20