DP

1) Fibonacci number:

$f(n) = f(n-1) + f(n-2)$

normal code: (recursive)

$\boxed{time = O(2^n) \; ; \; space = O(n)}$

```
int fibo (int n) {
if (n=1 || n=2) ret n;
ret fibo (n-1) + fibo (n-2); }
```

improvise code (dp): time = O(n) ; space = O(n)

(Top-down)

```
int memo [n+5] = {-1} ;
int fibo (int n) {
    if (n==1 || n==0) ret n;
    if (memo[n] != -1) ret memo[n];
    ret memo[n] = memo(n-1) + memo (n-2);
}
```

space optimized method: time = O(n) , space = O(1)

```
int fibo (int n) {
    int a=0, b=1, i, res;
    for (i = 0 -> n) {
        res = a+b;
        a = b;
        b = res; }
    ret b; [or ret res;]  }
```

improvised code (dp) :   bottom - up , with loop

```
int memo[n+5]       int fibo (int n) {
  = {-1};
                        memo[0] = 0;   memo[1] = 1;
                        for (i = 2 -> n+1) {
                            memo[i] = memo[i-1] + memo[i-2]
                        }
                        ret   memo[n];
                    }
```

2) $\boxed{0-1}$ knapsack problem:

take weight or not

n items → weight and profit value

Goal: max profit without crossing the limit weight

limitation: - either take or reject
- can't take half amt.

Ex: max wt = 5 ($\underline{w}$), total item, $\underline{n}$ = 4

100	20	60	40	value	→ val
3	2	4	1	weight	→ wt

Value table: (size - $n+1$ * $w+1$)

w (max) →

	w = 0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	100	100	100
2	0	0	20	100	100	120
3	0	0	20	100	100	120
4	0	40	40	100	140	140

i upto n ↑

bottom → up

code:

```
int wt [n+5];
int val [n+5];
int memo [n] [w];  → for table

int ks (int n, int w) { if(n==0|| w==0)
                          ret 0;
    if (wt[i] > w)
              memo[i][w] = memo [i-1] [w];

    ret memo[i][w] = max (memo[i-1][w],
             val[i] + memo[i-1][w-wt[i]]);
}
```

Explaination step:

1) $i = 1$ $wt[1] = 3$

$w = 1$ $3 > 1$ $dp(1,1) = d(0,1) = 0$

$3 > 2$ $dp(1,2) = d(0,2) = 0$

$3 \not> 3$ $dp(1,3) = mx(0,\ 100 + dp(0,\ 3-3))$

$3 \not> 4$ $dp(1,4) = mx(0,\ 100 + dp(0,4-3))$

$3 \not> 5$ $dp(1,5) = mx(0,\ 100 + dp(0,5-3))$

2) $i = 2$ $\quad wt[2) = 2$

$2 > 1 \quad dp(2,1) = dp(1,1) = 0$

$2 \not> 2 \quad dp(2,2) = mx(0, 20 + dp(1, 2-2))$
$dp(1,0)$

$2 \not> 3 \quad dp(2,3) = mx(100, 20 + dp(1,1))$

$2 \not> 4 \quad dp(2,4) = mx(100, 20 + dp(1,2))$

$2 \not> 5 \quad dp(2,5) = mx(100, 20 + dp(1,3))$

3) $i = 3$ $\quad wt[3] = 4$

$4 > 1 \quad dp(3,1) = dp(2,1) = 0$

$4 > 2 \quad dp(3,2) = dp(2,2) = 20$

$4 > 3 \quad dp(3,3) = dp(2,3) = 100$

$4 \not> 4 \quad dp(3,4) = mx(dp(2,4), 60 + dp(2,0)) = 100$

$4 \not> 5 \quad dp(3,5) = mx(dp(2,5), 60 + dp(2,1)) = 120$

4) $i = 4$ $\quad wt[4] = 1$

$dp(4,1) = mx(dp(3,1), 40 + dp(3, 1-1)) = 40$

$1 \not> 1 \quad dp(4,1) = mx(dp(3,2), 40 + dp(3,1)) = 40$

$1 \not> 2 \quad dp(4,2) = mx(dp(3,2), 40 + dp(3,1)) = 40$

$1 \not> 3 \quad dp(4,3) = mx(dp(3,3), 40 + dp(3,2)) = 100$

$1 \not> 4 \quad dp(4,4) = mx(dp(3,4), 40 + dp(3,3)) = 140$

$1 \not> 5 \quad dp(4,5) = mx(dp(3,5), 40 + dp(3,4))$

$\boxed{= 140}$

(Longest Common Subsequence)

A = {1, 2, 3, 4} , B = {1, 2, 3}

↳ subsequences : {1}, {2}, {3}, {4}

{1, 2}, {1, 3}, {1, 4}, {2, 3}

. . . .

{1, 2, 3}, {1, 2, 4} . . .

* maintain order / sequence :

{1, 4, 3} → not AC

Step :

lcs ({1, 2, 3, 4}, {1, 2, 3})

= 1 + lcs({2, 3, 4}, {2, 3})

= 2 + lcs ({3, 4}, {3})

= 3 + lcs({4}, { })

= 3 + 0 = 3

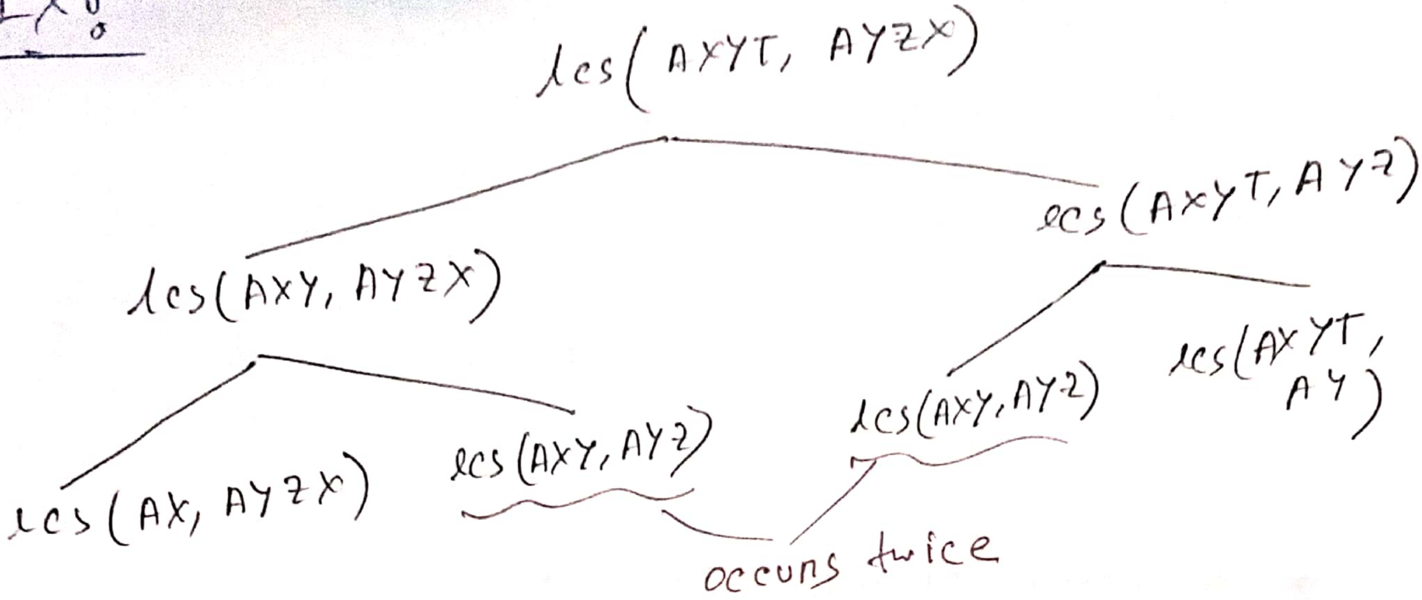
∴ length of lcs = 3

Healthcare

viset

CamScanner

Ex:



normal code : time = O(2^n)

```
                                x size    y size
int lcs (char x[], char y[], int m, int n) {
    if (m==0 || n==0) ret 0;
    if (x[m] == y[n]) ret 1+ lcs(x, y, m-1, n-1);
    ret max (lcs(x, y, m-1, n), lcs(x, y, m, n-1));
}
```

improvised : (dp) O(m×n) top-down

```
int dp[m][n]      int lcs ( char x[], char y[], int m, int n)
  = {-1};         {  if (m==0 || n==0) ret 0;
                     if (dp[m][n] != -1) ret dp[m][n];
```

~~if (x[m] = y[n] ret dp[m][n] = dp[m-1][n-1]~~

if (x[m-1] == y[n-1]) ret dp[m][n] = 1 + lcs(x, y, m-1, n-1);

ret dp[m][n] = max(lcs(x, y, m-1, n), lcs(x, y, m, n-1));

}

* iterative (bottom → up)

x : ACADB   y : CBDA

		C	B	D	A
	0	0	0	0	0
A	0	0	0	0	1
C	0	1	1	1	1
A	0	1	1	1	2
D	0	1	1	2	2
B	0	1	2	2	2

(match) → diagonal + 1

if char of current row = column (cur),
val [r][c] = val[r-1][c-1] + 1;

else
max (prev row, prev col)