Dynamic Programming

* { – Overlapping Subproblem (subtree, prefix of an array, suffix, substring..)
– Optimal Substructure (solutions of subproblems help to solve the original problem)

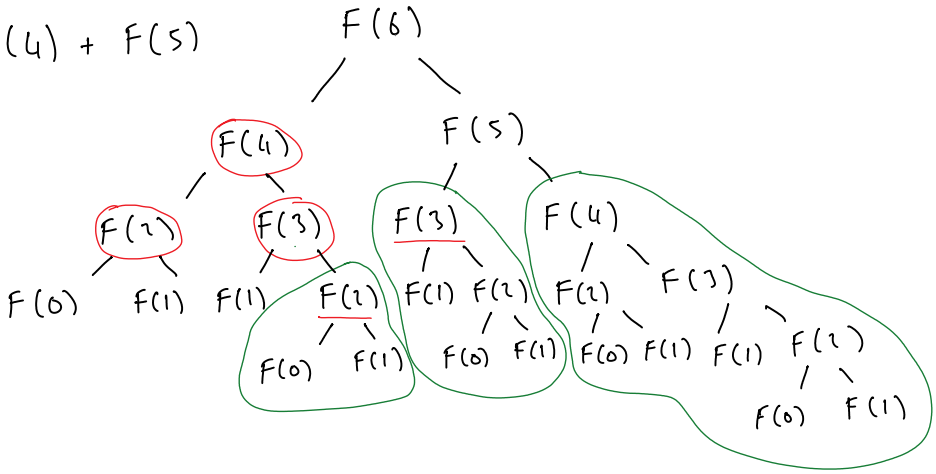
implementation { – Recursion + memoization (store partial solution in a data structure) (top to bottom design)
– Tabulation (table filling) (bottom-up design)

Example: N^{th} Fibonacci number.

0	1	2	3	4	5	6	7	8	
0	1	1	2	3	5	8	13	21	. . .

base case

$F(6) = F(4) + F(5)$




```
int fib(int n)
{
    if (n < 2)          // base case
        return n;
    return fib(n-2) + fib(n-1);
}
```

Recursive solution

$O(2^n)$ exponential

$O(1)$

```
vector<int> memo(N+1, -1);

int fib(int n)
{
    if (n < 2)          // base case
        return 0;
    if (memo[n] > -1)   // already calculated
        return memo[n];
```

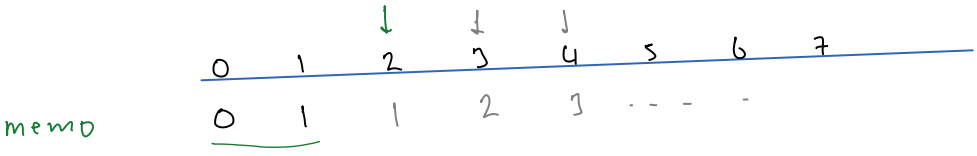
recursion + memoization

$O(N)$ time, linear

$O(N)$ space

```
    return memo[n] = fib(n-1) + fib(n-2);
}
```

Implementation with Tabulation




```
memo[0] = 0, memo[1] = 1;

for(int i=2; i<=N; i++)
    memo[i] = memo[i-1] + memo[i-2];

cout << memo[N] << endl;
```

$$Fib(n) = \begin{cases} n & \text{if } (n < 2) \quad \text{// base case} \\ Fib(n-1) + Fib(n-2) & \end{cases}$$

Dynamic Programming Solves

1) Decision problem (if it is possible ...)
2) Optimization problem (min cost, max profit ...)
3) Combinatorial problem (in how many different ways ...)

Example 1: Sum

In how many different ways you can obtain N as sum of a, b, and c? N, a, b and c are positive integers. Repetitions are allowed.

Sample Input

```
5
1 3 4
```

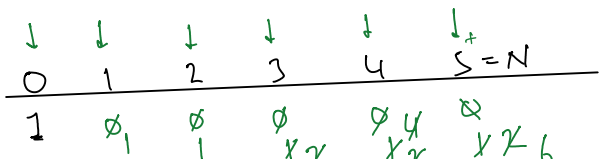
Sample Output

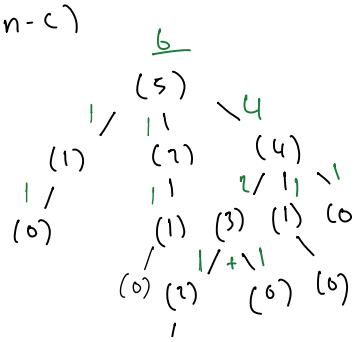
```
6
```

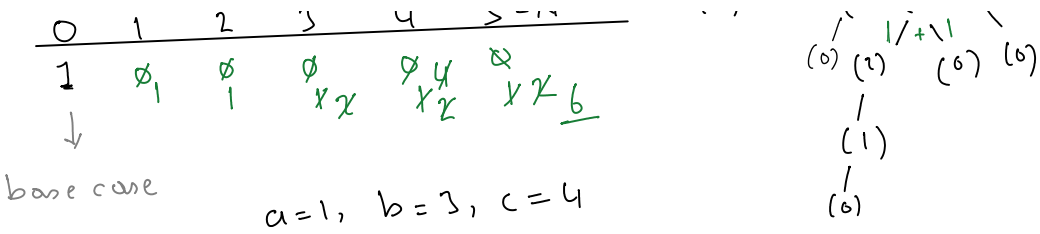
Output Details

```
1 + 1 + 1 + 1 + 1
1 + 4
4 + 1
1 + 1 + 3
1 + 3 + 1
3 + 1 + 1
```

$$F(n) = \begin{cases} 1 & \text{if } n == 0 \quad \text{// base case} \\ F(n-a) + F(n-b) + F(n-c) & \end{cases}$$








```
// Sum. Top-down desgin.
#include "pch.h"
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

int N = 55;
int a = 1, b = 3, c = 4;
vector<int> dp;
//-------------------------------
int go(int n)
{
	//base case
	if (n == 0)
		return 1;

	//already calculated
	if (dp[n] > 0)
		return dp[n];

	//recurrence
	int temp = 0;
	if (n - a >= 0)
		temp += go(n - a);
	if (n - b >= 0)
		temp += go(n - b);
	if (n-c >=0)
		temp += go(n - c);

	return dp[n] = temp;		//store solution of this subproblem
}
//-------------------------------
int main()
{
	dp.resize(N + 1, 0);
	cout << go(N) << endl;
}
```

```
// Sum. Bottom-up desgin.
#include "pch.h"
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

int N = 45;
int a = 1, b = 3, c = 4;
//-------------------------------
int go(int n)
{
	vector<int> dp(N+1, 0);

	dp[0] = 1;  //base case

	for (int i = 1; i <= N; i++)
	{
		if (i - a >= 0)
			dp[i] = dp[i - a];
```

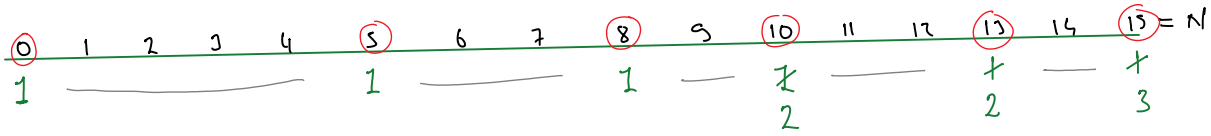
```
        if (i - b >= 0)
            dp[i] += dp[i - b];
        if (i - c >= 0)
            dp[i] += dp[i - c];
    }

    return dp[N];
}
//------------------------------
int main()
{
    cout << go(N) << endl;
}
```

Exercise 1: Bottom-Up Design For a Sparse Memo

Re-implement the iterative *Sum* problem to skip the elements of the memo array if they remain 0.
Consider the input where *N* is 10^6, *a* is 10^4, *b* is 2*10^4 and *c* is 5*10^4.

a = 5, b = 8, c = 10




```
priority_queue<int> q; ← should be min heap
q.push(0);
dp[0] = 1;
vector<bool> inqueue(N+1, false); inqueue[0] = true;
while (!q.empty())
{   int cur = q.top();
    q.pop();  inqueue[cur] = false;
    if (cur + a <= N) { dp[cur + a] += dp[cur];
                        if (!inqueue)
                            q.push(cur + a);
        cur + b . . .
        cur + c . . .
}
return dp[N];
```

BSF solution

The "Wines" Problem

Imagine you have a collection of N wines placed next to each other on a shelf. For simplicity, let's number the wines from left to right as they are standing on the shelf with integers from 1 to N, respectively. The price of the i-th wine is pi (prices of different wines can be different).

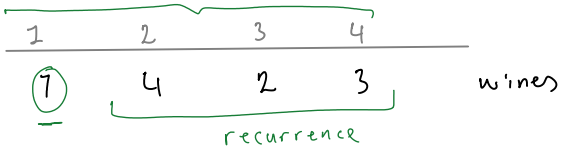
Because the wines get better every year, supposing today is the year 1, on year y the price of the i-th wine will be y*pi, i.e. y-times the value that current year.

You want to sell all the wines you have, but you want to sell exactly one wine per year, starting on this year. One more constraint - on each year you are allowed to sell only either the leftmost or the rightmost wine on the shelf and you are not allowed to reorder the wines on the shelf (i.e. they must stay in the same order as they are in the beginning).

You want to find out, what is the maximum profit you can get, if you sell the wines in optimal order.

So for example, if the prices of the wines are (in the order as they are

Optimization Problem.



F(1, N) = ?

year = 4 - (4 - 1)
= 1

N = 4
left = 1

You want to find out, what is the maximum profit you can get, if you sell the wines in optimal order.

So for example, if the prices of the wines are (in the order as they are placed on the shelf, from left to right): p1=1, p2=4, p3=2, p4=3

The optimal solution would be to sell the wines in the order p1, p4, p3, p2 for a total profit 1*1 + 3*2 + 2*3 + 4*4 = 29

N = 4
left = 1
right = 4

$$F(left, right, year) = \begin{cases} 0 & \text{if } left > right \\ \max \begin{cases} year \times wines[left] + F(left+1, right, year+1) \\ year \times wines[right] + F(left, right-1, year+1) \end{cases} \end{cases}$$

$$year = N - (right - left)$$

$$F(left, right) = \begin{cases} 0 & \text{if } left > right \\ \max \begin{cases} year * wines[left] + F(left+1, right) \\ year * wines[right] + F(left, right-1) \end{cases} \end{cases}$$

$$* \; year = N - (right - left)$$

dp[N+1][N+1] dp[i][j] => best profit in the range i...j

```
// Wines. Top-down desgin
#include "pch.h"
#include <fstream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;

ifstream cin("1.in");
ofstream cout("1.out");

int N;
vector<int> wines;
vector<vector<int>> dp;
//--------------------------------
int go(int left, int right)
{
    if (left > right)
        return 0;
    if (dp[left][right] >= 0)
        return dp[left][right];

    int year = N - (right - left);
    int sol1 = year * wines[left] + go(left + 1, right);
    int sol2 = year * wines[right] + go(left, right - 1);

    return dp[left][right] = max(sol1, sol2);
}
//--------------------------------
int main()
{
```

Solution with Tabulation

left →	1	2	3	right → 4
wines	1	4	2	3

6 + 9
7 + 10
17 + 8
9 + 10

	1	2	3	4
1	1	9	15	
2		4	10	20
3			2	8
4				3

dp

result
4 x 1 + (20)
4 x 3 + (15)

dp[i][j] ← solution from the pos. i to pos. j.

Base case: only one wine. (main diagonal)

```
}
//------------------------------
int main()
{
    cin >> N;
    wines.resize(N + 1);
    for (int i = 1; i <= N; i++)
        cin >> wines[i];

    dp.resize(N + 1, vector<int>(N + 1, -1));
    cout << go(1, N) << endl;
}
```

Base case: only one wine. (main diagonal)

dp[1][2] => year = (2-1) + 1 = 2

↳ (1)[2-2] => 2*1 + 4 = 6

[1-1](2) => 1 + 2*4 = 9

$$dp(1,j) = \max \begin{cases} 3 \times wines[1] + dp(2,j) \\ 3 \times wines[j] + dp[1,j-1] \end{cases}$$

$$dp[i][j] = \max \begin{cases} year \times wines[i] + dp[i+1][j] \\ year \times wines[j] + dp[i][j-1] \end{cases}$$

$O(N^2)$ time

$O(N^2)$ space.

Question: wines with tabulation.

dp[i][j] ← max profit starting from the position i and having j wines. (j is the length of the range)