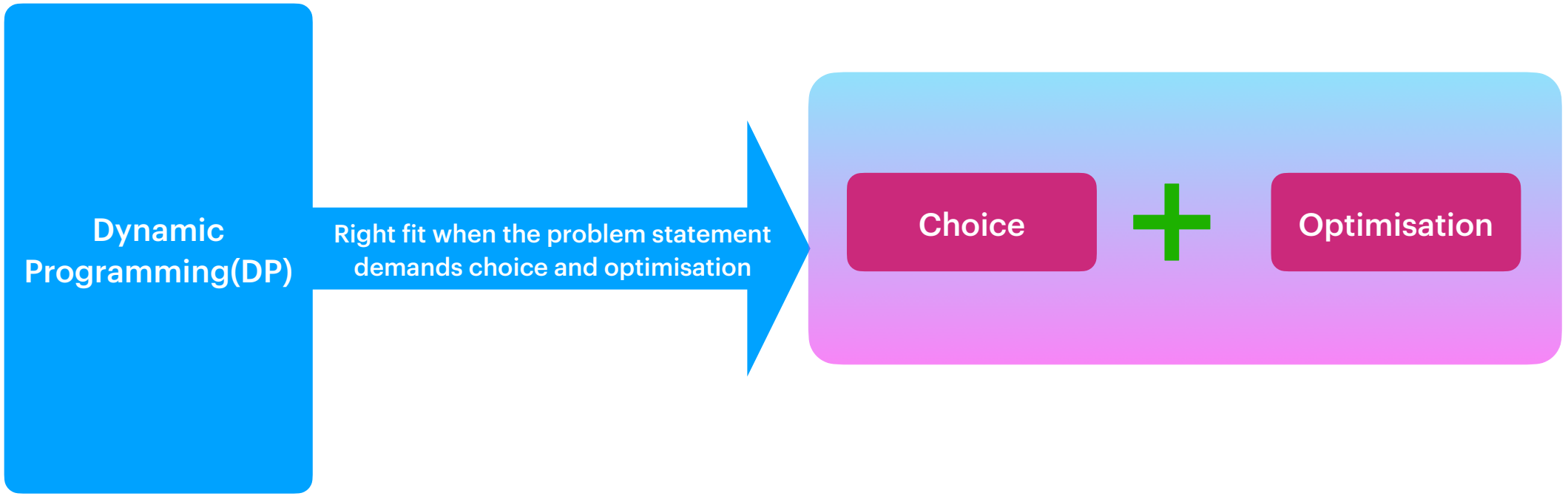
	What?	Why?	When?	How ?
Sorting	Ordering element in either Ascending or Descending	If we don’t sort, search takes linear time O(n), with sorting we can do BinarySearch which takes O(logn)	We go for sorting for searching .	InsertionSort MergeSort QuickSort
Binary Search	Element less than the root on left side, greater than root on right side. {1,2,3,4,5}	With BinarySearch at every step we can eliminate n/2 elements recursively .Helps us to find element in O(logn)	Where there are sorted elements.	Find a mid element do the math over the mid for solving the problem.

Dynamic Programming says derive a optimised solution at “sub problem” level which obviously result to a optimistic solution for the main problem.

The best way to understand Dynamic Programming is through recursion. With recursion we can reach to the depth of the problem(i.e sub problem) from there we can give the optimised solution.

Dynamic Program is right fit when the problem statement demands Choice & Optimisation.



Let’s figure out the use cases of Dynamic Programming (DP).

Withdraw use case in ATM:

ATM make sure always minimal number of notes would be given while withdraw process.
Let’s see how this fit into DP.

ATM has three choices (100, 500, 2000 notes) and then make sure minimal notes would be given to user
Ex : Withdraw of 2000 = 1 note (i.e 2k note)
Withdraw of 1700 = 5 notes (500 notes 3 + 100 notes 2)

. It’s a DP problem.
DP = Choice (Out of 3 Notes) + Optimisation (Should be minimal)

Finding shortest Path in Google Maps.

In Google Maps when we choose destination, there could be multiple possible routes from source to destination but the shortest path to be chosen by Google Maps.

It’s a DP problem.

DP = Choice (Out of multiple routes/nodes)
+
Optimisation (shortest path to be chosen)

Finding higher profit with fixed budget on stock exchange.

We have wide number of stocks available in the market but we need choose specified stocks which gives higher profit for a given budget.

It’s a DP problem.

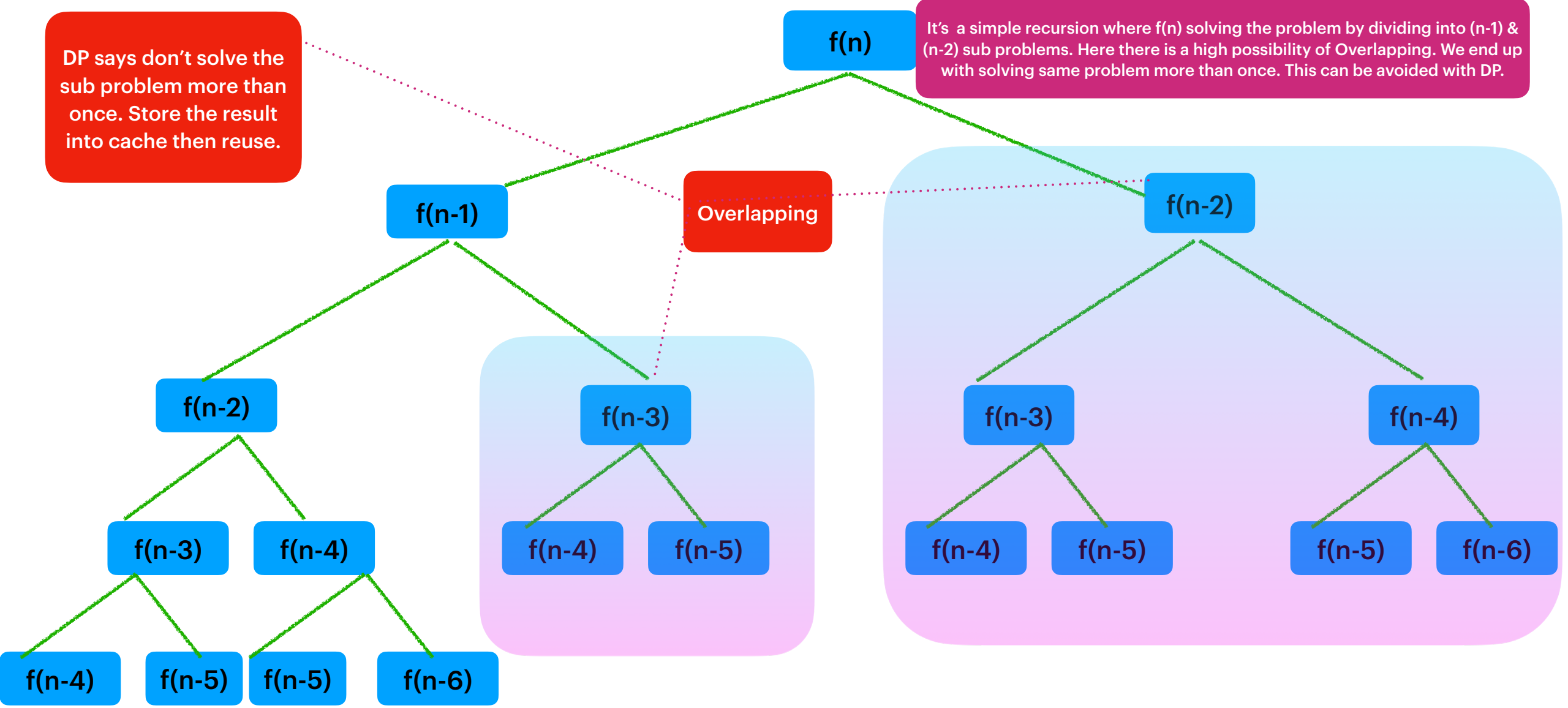
DP = Choice + Optimisation

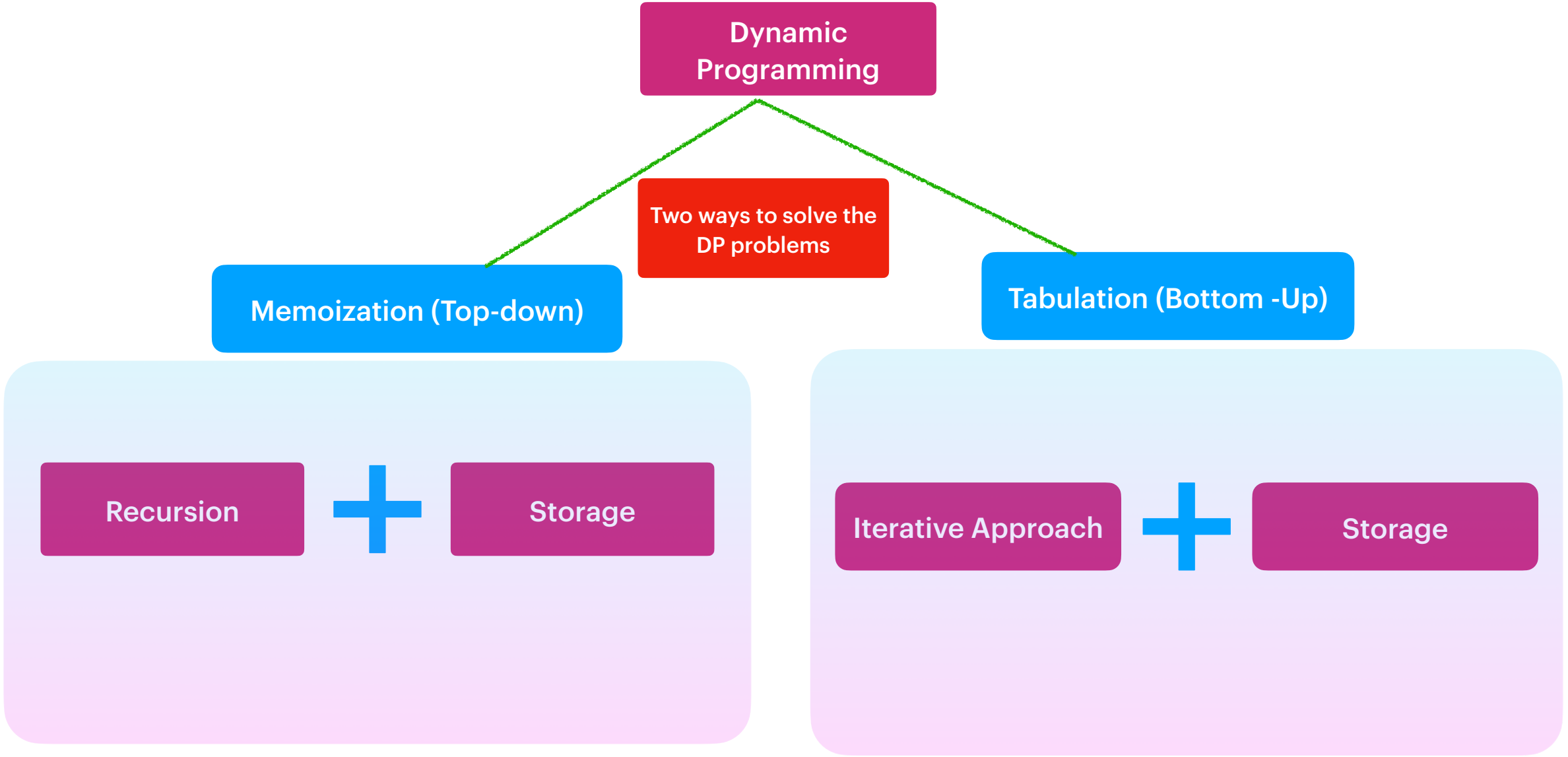
Travelling Salesman (Amazon Delivery Agent)

When Amazon Delivery Agent delivering products to specific locations, he should not revisit the same location twice.

It’s a DP problem.

DP = Choice (locations) + Optimisation (should not be revisited)


DP says don't solve the sub problem more than once. Store the result into cache then reuse.
f(n)
It's a simple recursion where f(n) solving the problem by dividing into (n-1) & (n-2) sub problems. Here there is a high possibility of Overlapping. We end up with solving same problem more than once. This can be avoided with DP.
f(n-1)
Overlapping
f(n-2)
f(n-2)
f(n-3)
f(n-3)
f(n-4)
f(n-4)
f(n-5)
f(n-3)
f(n-4)
f(n-4)
f(n-5)
f(n-5)
f(n-6)
f(n-4)
f(n-5)
f(n-5)
f(n-6)


Dynamic Programming
Two ways to solve the DP problems
Memoization (Top-down)
Tabulation (Bottom -Up)
Recursion
Storage
Iterative Approach
Storage