Dynamic programming

1) It is used to solve optimisation problem.
2) It is similar to divide & conquer such that there are going to be subproblem & the given problem is solved by combining the answers of sub problems.
3) unlike divide & conquer the subproblems are not independent
4) It is based on assumption that smaller subproblem results are needed frequently.

Divide & conquer	Dynamic programming
1) partition the problem into independent sub problem	1) partition the problem into overlapping subproblems
2) doesn't store the solution of the subproblem (identical subproblems are recomputed again)	2) Avoids recompution by storing the result into a table (space complexity is more)
3) It is a top down approach	3) It is a bottom up approach
4) It is not used for optimisation problem	4) It is used for optimasation problem
5) for eg. n^{th} term of fibonacci series if computed the complexity as $O(2^n)$ ie exponetial	5) for eg. fibonacci is achieved through array (table of smaller subproblem result) complexity is $O(n)$

Rules of dynamic programming

1) optimal substructure - An optimal Sol^n to a proble given by optimal solution to sub problem.

2) overlapping sub problem - A recursive solution contains a small no of distinct sub problems repeated many times.

3) Bottom up fashion - Compute the solution in bottom up fashion

There are 3 Components in DP

1) Recurrence relation
2) Tabular Design
3) Solution Constructed by backtracking the result table.

knapsack problem using dynamic programming (0/1 knapsack)

A bag of Capacity m is given to you. a set of item havings weights $w_1, w_2 -- w_n$ are given with their profit/values $P_1, P_2 -- P_n$.

Objective is to add an items to the bag in such a way that profit earned is maximum. Items are added to the bag completely i.e fractioned of items can't be added.

Item	i	1	2	3	4
Value/Profit	V	100	20	60	40
weight	W	3	2	4	1

weight of knapsack (m) = 5

$\Rightarrow$ Create a value table V[i,w] where i denotes no of items & w denotes weight of the items

⇒ rows denote the items, columns denote the weight
(As there are 4 items so we have rows from 0 to 4)

⇒ The weight limit of the knapsack is 5. so we have 6 columns from 0 to 5

Soln

V[i,w]	w=0	1	2	3	4	5
i=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	$\boxed{140}$

⇒ fill the first row, i=0 with 0. this means when ~~w=0 weight is~~ 0 item is considred weight is 0

⇒ fill the first column, w=0 with 0. This means when weight is 0, item considered is 0.

⇒ Rule

if wt[i] > w then
V[i,w] = V[i-1,w]
else if wt[i] <= w then
V[i,w] = max (V[i-1,w], val[i] + V[i-1, w-wt[i]])

⇒ i=1, w=1 we will fill V[1,1]

Is wt[i] > w ? ⇒ NO
~~V[1,1] = max(V[0,1], val[1] + V[0, 5-wt[1]])~~
~~= max(0, 100 + V[0,2])~~
=

wt[1] = 3
3 > 1
⇓ yes
so, V[1,1] = V[0,1]
V[1,1] = 0

now $i=1, w=2$

Is $wt[i] > w$

$3 > 2$

? yes

$V[i,w] = V[i-1,w]$

$V[1,2] = V[0,2]$

$= 0$

now $i=1, w=3$ $V[1,3]$

Is $weight[i] > w$

$3 > 3$

No?

$V[i,w] = max(V[i-1,w], val[i] + V[i-1, w-wt[i]])$

$= max(V[0,3], 100 + V[0, 3-3])$

$= max(0, 100 + V[0,0])$

$= max(0, 100+0)$

$V[1,3] = 100$

Now $i=1, w=4$

~~V[i,w]~~ is ~~V[i,~~ $wt[i] > w$

$3 > 4$

No

$V[i,w] = max(V[i-1,w], val[i] + V[i-1, w-wt[i]])$

$V[1,4] = max(V[0,4], val[1] + V[0, 4-3]$

$= max(0, 100+0)$

$= 100$

$V[1,5] =$

Is $3 > 5$ then

$V[1,5] = \max(V[0,5], 100 + V[0,2])$

$= \max(0, 100+0)$

$V[1,5] = 100$

now, $i=2$, $w=1$

$V[2,1] = wt[2] > w$

$= 2 > 1$

? yes

$V[2,1] = V[0,1]$

$V[2,1] = 0$

now, $i=2$, $w=2$, we will find $V[2,2]$

if $wt[2] > 2$

$2 > 2$

? ~~yes~~ NO

$V[2,2] = \max(V[1,2], val[2] + V[1, 2-2])$

$= \max(0, 20 + V[1,0])$

$= \max(0, 20+0)$

$V[2,2] = 20$

now, $i=2$, $w=3$ we will find $V[2,3]$

if $wt[2] > 3$

$2 > 3$

? NO

$\max(V[1,3], val[2] + V[1, 3-2])$

$\max(100, 20 + V[0,1])$

$V[2,3] = 100$

now $i=2,\ w=4$

$V[2,4] =$

If $(2>4)$

$\downarrow$ no

$= \max(V[i-1, w], val[i] + V[i-1, w-wt[i]])$

$= \max(V[1,4], 20 + V[1, 4-2])$

$= \max(V[1,4], 20 + V[1,2])$

$= \max(100, 20+0)$

$= 100$

now $i=2,\ w=5$

$V[2,5] =$

~~If~~ $2>5$

? $\downarrow$ no

$V[2,5] = \max(V[1,5], val[2] + V[1,3])$

$= \max(100, 20+100)$

$= 120$

Scanned by CamScanner

algo to Print which items should be added to knapsack

set $i = n$ and $w = W$

while i & $w > 0$ do
if $V[i, w] \; != V[i-1, w]$ then
mark the i^{th} item
set $w = w - w[i]$
$i = i - 1$
else
set $i = i - 1$
endif
end while

So, Items we are putting inside knapsack are 1 & 4

Profit earned is $100 + 40 = 140$