01 Knapsack

**Problem**: <https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/>

<https://www.youtube.com/watch?v=l02UxPYRmCQ&list=PL_z_8CaSLPWekqhdCPmFohncHwz8TY2Go&index=2>

**Approach**: Start with the last element, see if it can be chosen, if yes check out both the options with or without last element by solving the problem for (n-1)

**Recursive approach:**

<https://www.youtube.com/watch?v=kvyShbFVaY8&list=PL_z_8CaSLPWekqhdCPmFohncHwz8TY2Go&index=3>

//Function to return max value that can be put in knapsack of capacity W.

int knapSack(int W, int wt[], int val[], int n)

{

// Your code here

if(n==0 || W==0)//base case, when there's no space in bag or we don't have any option to choose

return 0;

if(wt[n-1]<=W){//check if we can pick the last item

return max(val[n-1]+knapSack(W-wt[n-1],wt,val,n-1),knapSack(W,wt,val,n-1));//try out both the option

}

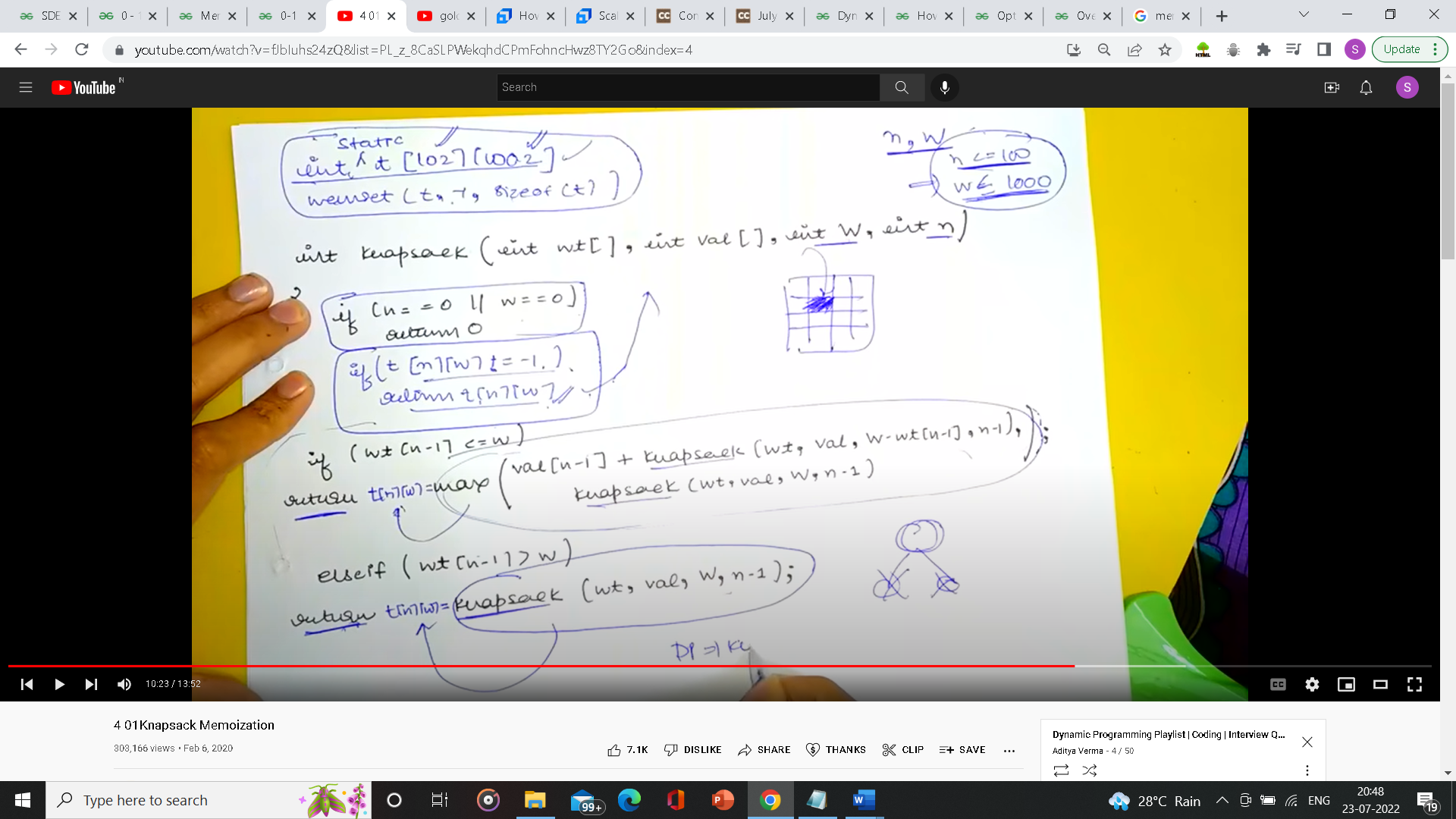
else

return knapSack(W,wt,val,n-1);//move to next if can't pick the this item

}

**Top-down approach/memorization**

<https://www.youtube.com/watch?v=fJbIuhs24zQ&list=PL_z_8CaSLPWekqhdCPmFohncHwz8TY2Go&index=4>



**Bottom-up**

//Function to return max value that can be put in knapsack of capacity W.

int knapSack(int W, int wt[], int val[], int n)

{

int kp[n+1][W+1];

int i,j;

for(i=0; i<=n; i++){

kp[i][0]=0;

}

for(i=0; i<=W; i++){

kp[0][i]=0;

}

for(i=1; i<=n; i++){

for(j=1; j<=W; j++){

if(wt[i-1]<=j)

kp[i][j]=max(val[i-1]+kp[i-1][j-wt[i-1]],kp[i-1][j]);

else

kp[i][j]=kp[i-1][j];

}

}

return kp[n][W];

}

<https://www.youtube.com/watch?v=ntCGbPMeqgg&list=PL_z_8CaSLPWekqhdCPmFohncHwz8TY2Go&index=5>

**How to identify Knapsack problem**

When given n choices to fill something with a given capacity we can consider it a variation of knapsack.