Dynamic Trogramming.	(
Example 1: LIS = longest increasing subsequence.	
Given n numbers a,,.,an find the length of the LIS.	
Example 5, 7, 4, -3, 9, 1, 4, 8, 6, 7, 5	
LIS=5 from -3,1,4,6,7	
First step, Define subproblem in words, Then define recurrence. If can't find a Then define recurrence. If can't find a recurrence then probably get an idea how to revise)	
Attempt 1: Let T(i) = length of LIS in a,, a;	
What's the recurrence? T(1)=3 from 5,7,9,	
What's the recurrence. For the above example, $T(6)=3$ from 5,7,9, but we want -3,1 so that $T(7)=3$ from -3,1,4	†
Then for T(8) can we add 8 on? Then for T(8) can we add 8 on? Yes if it's -3,1,4, no if it's 5,7,9, how do we know which? Keep track of all Possible endings, so	
Keep track of all Possible endings, so	

try the following.

Attempt 2: Let T(i) = length of LIS in a,,,, a; which includes a; Now we know the end so we know if we can all to it. Hence, T(i) = 1 + max (T(j): aj<a; } Algorithm. LIS (a, ..., an) for i=1 -> 1 for j=1>i-1
if a; < a; then T(i)=max{T(i), 1+T(j)} Max = 1

(2)

max=1
for i=2->n
if T(i)>T(max) then max=i

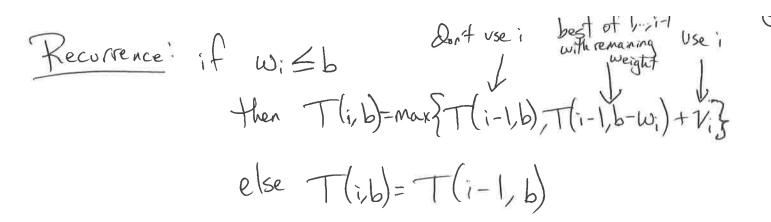
Return (T(max))

Running time: O(n2)

Knapsack: n objects with integer weights w.,.., wh & integer values Vi, ..., Vn total capacity B What's subset 5 of objects where ≥ wi ≤ B 2 which maximizes \subseteq 1; Version 1. one copy of each object. Attempt 1: Let T(i)= max value attainable using subset of objects 1,..., i But then for T(i) can we all object i to optimal Solution for T(i-1)? May want suboptimal Solution for T(i-i) which has enough capacity so that can all object i.

So want to see optimal solution for given capacity.

Attempt 2: Let T(i,b)= max value attainable using Subset of In i & total Capacity <b.



Final answer: T(n,B)

Running time: O(nB)

Version 2: unlimited sopply of each object.

Now we don't need to keep track of what objects are used so far.

Let T(b) = max value attainable using total capacity 66.

For the recurrence, try all possibilities for the last object to add.

T(b) = max {T(b-wi)+1/: wi < b}

Final result: T(B)

Running time: O(nB)

Knapsack has running time O(nB). Is this a polynomial-time algorithm? NO, the number B is Part of the input and it's input size is O(log B). Well see later that knapsack is NP-complete. Well reduce from a 3-SAT instance with N variables & m constraints and the knapsack instance will have B=exponentially large in n2m.