

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

# Algorithms

[illegible]

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	52
--	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	----

## SUBSET SUM

Given an array  $A[1..n]$  containing positive numbers we need to check if a subset of the numbers in the array sum to a given number  $T$ .

*Example.*    1   2   5   7

*Is there a subset that sums to 13?    Yes.*

*Is there a subset that sums to 0?    Yes, the empty subset!*

*Is there a subset that sums to 4?    No.*

How fast can we solve this problem?

## SUBSET SUM

Let  $SS(i, t) = \text{TRUE}$  iff some subset of  $A[i..n]$  sums to  $t$ .

We seek  $SS(1, T)$ .

What does it mean if  
 $SS(i, t)$  is true?

$$SS(i, t) = \begin{cases} \text{TRUE} & \text{if } t = 0 \\ \text{FALSE} & \text{if } i > n \\ SS(i + 1, t) & \text{if } A[i] > t \\ SS(i + 1, t) \text{ OR } SS(i + 1, t - A[i]) & \text{otherwise} \end{cases}$$

For what range of values of  $i$  and  $t$  do we need to memoize  $SS(i, t)$ ?

*For  $i = 1$  to  $n$  and for  $t = 1$  to  $T$ .*

Running time?  $O(nT)$ .

Finalize the code

## DYNAMIC PROGRAMMING

- Dynamic programming approach is similar to divide and conquer in breaking down the problem into smaller and yet smaller possible sub-problems.
- Unlike, divide and conquer, these sub-problems are overlapping and can be used for similar sub-problems.

# DYNAMIC PROGRAMMING

## When to use dynamic programming?

When we have a recursive formulation in which there are *overlapping subproblems* but the *number of distinct subproblems is small*.

**Memoization.** Store results of computations. When recursing on a sub-problem we check if the result is stored. If not, we proceed with the computation and then store the result.

## Previous lecture:

*Computing Fibonacci numbers, Weighted Interval Scheduling, Subset Sum*

# KNAPSACK PROBLEM

A robber find a lot of valuable items which have different weights and values. Unfortunately his knapsack can hold only a certain maximum weight. How should he decide what to take?

*We assume that the weights are positive integers.*

<i>Example:</i>	<i>Item</i>	<i>Value</i>	<i>Weight</i>
	<i>1</i>	<i>16</i>	<i>4</i>
	<i>2</i>	<i>14</i>	<i>3</i>
	<i>3</i>	<i>9</i>	<i>2</i>
	<i>4</i>	<i>30</i>	<i>6</i>

*Knapsack capacity: 10*

Two variants:

1. Unlimited copies of each item.
2. Only one copy of each item.

# KNAPSACK PROBLEM

Given  $n$  items where the  $i^{th}$  item has weight  $w_i$  (a positive integer) and value  $v_i$  and a knapsack capacity  $W$ , find the maximum value of a subset of the items whose total weight is at most  $W$ .

**Variant 1. Knapsack with repetition.** *unlimited copies of each item*

Let  $V(w)$  denote the maximum value achievable with a knapsack of capacity  $w$ .

If item  $i$  is included then we get value  $v_i$  and our remaining capacity decreases by  $w_i$ .

$$V(w) = \max_{i:w_i \leq w} \{v_i + V(w - w_i)\}$$

*Convention: the maximum of an empty set is 0.*

$$V[0] = 0$$

for  $w = 1$  to  $W$ :

$$V[w] = \max_{i:w_i \leq w} \{V[w - w_i] + v_i\}$$

*$V[W]$  is our solution.*

*Running time:  $O(nW)$*

## KNAPSACK PROBLEM

**Variant 2. Knapsack without repetition.**    *one copy of each item*

Let  $V(w, i)$  be the maximum value achievable under the constraint that the total weight is  $\leq w$  and we only choose among the first  $i$  items.

We seek  $V(W, n)$ .

$V(0, i) = V(w, 0) = 0$  for all  $i$  and  $w$ .

For  $i, w > 0$ ,

$$V(w, i) = \begin{cases} \max\{V(w - w_i, i - 1) + v_i, V(w, i - 1)\} & \text{if } w_i \leq w \\ V(w, i - 1) & \text{otherwise} \end{cases}$$

How much time does it take to compute  $V(W, n)$ ?     $O(nW)$



## LONGEST INCREASING SUBSEQUENCE

### **Exercise.**

Suppose that we are given an array  $A$  with  $n$  numbers and we want to compute the length of the longest increasing subsequence in  $A$ .

*Example.*    1   4   2   8   5   7

*There are several increasing subsequences:*   1   2   8, 4   8, 1   4   7 *etc.*

*The longest ones among these are* 1   2   5   7 *and* 1   4   5   7.

How fast can we compute the longest increasing subsequence?

## LONGEST INCREASING SUBSEQUENCE

Suppose that we are given an array  $A$  with  $n$  numbers and we want to compute the length of the longest increasing subsequence in  $A$ .

Let  $L(i)$  be the length of the LIS ending at index  $i$  such that  $A[i]$  is the last element.

Then,  $L(i)$  can be recursively written as:

$L(i) = 1 + \max(L(j))$  where  $0 < j < i$  and  $A[j] < A[i]$ ; or

$L(i) = 1$ , if no such  $j$  exists.

Formally, the length of LIS ending at index  $i$ , is 1 greater than the maximum of lengths of all LIS ending at some index  $j$  such that  $A[j] < A[i]$  where  $j < i$ .

Example:

$A=[3,2,6,4,5,1]$

What is  $L(0)$ ,  $L(1)$ ,  $L(2)$ ,...?

$L(0)=1$ - (LIS: 3),

$L(1)=1$ - (LIS: 2),

$L(2)=2$ - (LIS: 2,6),

$L(3)=2$ - (LIS: 3,4),

$L(4)=3$ - (LIS: 3,4,5),

$L(5)=1$ - (LIS: 1)

At  $i=4$  ( $A[4]=5$ ),  $L(0)$ - $L(3)$  were already calculated, and we need to find  $L[4]$ .

Note that  $L(0)$ ,  $L(1)$  and  $L(3)$  have tails  $<5$ , so we append the longest to  $L(4)$ ; which is  $L(3)$ : 3,4,5