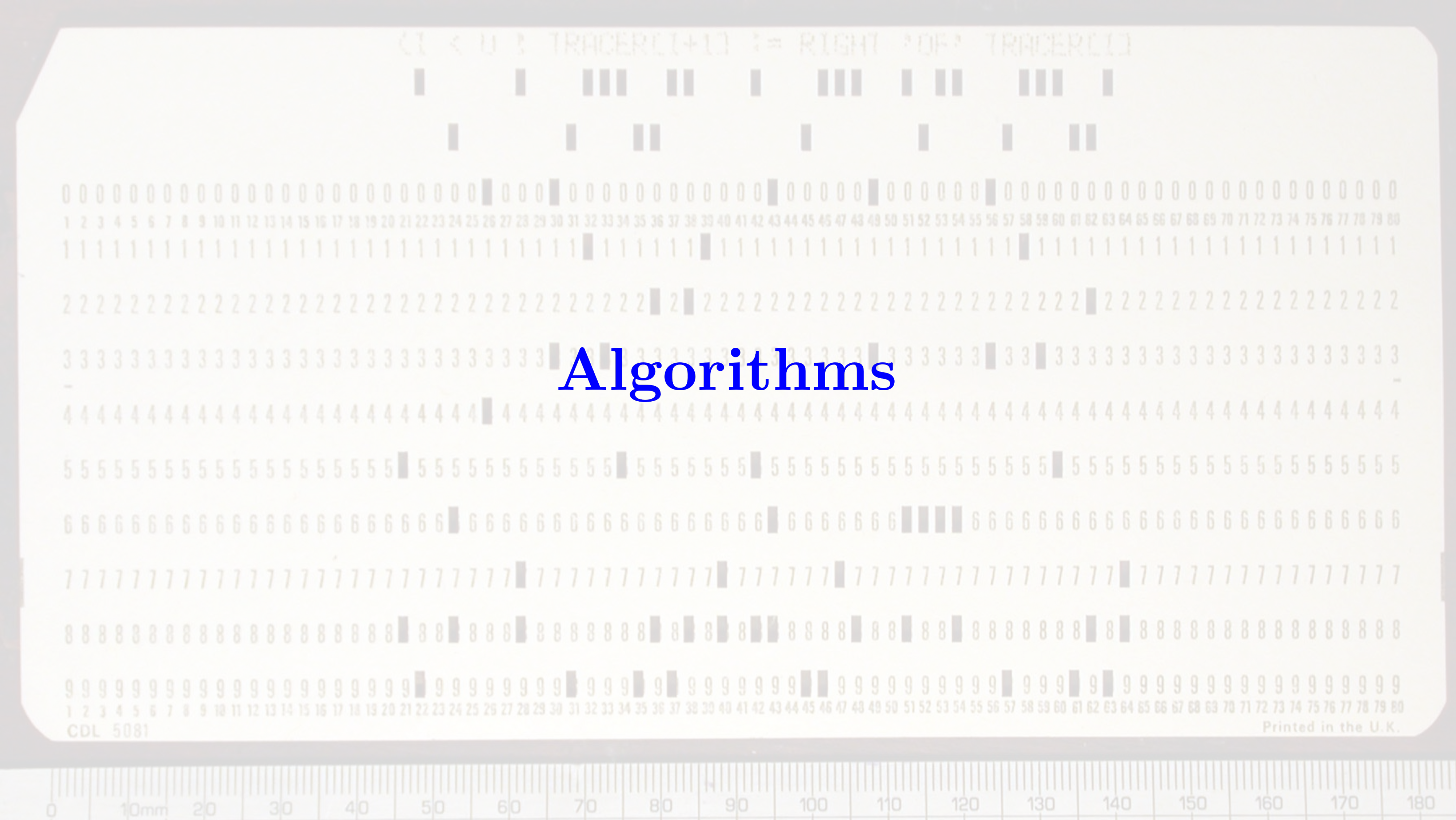

Algorithms

Subset Sum

Given an array $A[1..n]$ containing <u>positive</u> 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 $SS(i,t) = \texttt{TRUE}$ 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?

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

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.

Example:

Item	*Value*	*Weight*
1	*16*	*4*
2	*14*	*3*
3	*9*	*2*
4	*30*	*6*

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 repitition. *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 repitition. *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