

# Algorithms & Complexity

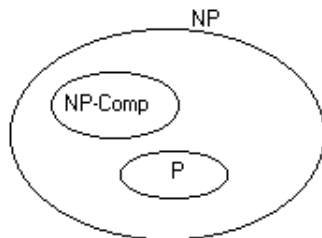
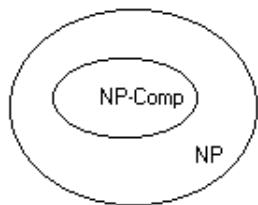
## Space Complexity

Nicolas Stroppa - [nstroppa@computing.dcu.ie](mailto:nstroppa@computing.dcu.ie)

CA313@Dublin City University. 2006-2007.

December 4, 2006

# Hierarchy of problems



## *NP*-intermediate Languages

If  $P \neq NP$ , then are there languages which neither in  $P$  or  $NP$  – complete?

The answer is yes.

These languages are called *NP*-intermediate languages, and their class is called *NPI*. There are real problems in  $NP$  for which neither polynomial-time algorithms nor  $NP$ -completeness proofs have been found. It makes sense to assume that these belong to *NPI*, until we know differently.

**Examples of such problems are:**

- ▶ Graph isomorphism
- ▶ Linear programming
- ▶ Composite number (Given  $N \in \mathbb{Z}_+$ , find  $m, n \in \mathbb{Z}_+$  such that  $mn = N$ )

# Co- $NP$ Problems

## Definition (Co- $NP$ )

A problem  $X$  is a member of co- $NP$  if and only if its complement  $\overline{X}$  is in complexity class  $NP$ . In simple terms, co- $NP$  is the class of problems for which efficiently verifiable proofs of no instances, sometimes called counterexamples, exist.

# The subset-sum problem

## Definition (The subset-sum problem (SSP))

Given a set of integers  $S = \{i_1, i_2, \dots, i_n\}$ , does any (non-empty) subset  $A \subseteq S$  sum to 0?

## The complementary problem of the subset-sum problem asks:

Given a finite set of integers does every non-empty subset have a nonzero sum?

To give a proof of a "no" instance one must specify a non-empty subset which does sum to zero. This proof is then easy to verify.

**The complementary problem of the subset-sum problem is in co-NP.**

# Space complexity

## Two main characteristics for programs

- ▶ Time complexity:  $\simeq$  CPU usage
- ▶ Space complexity:  $\simeq$  RAM usage

# Space complexity: an informal definition

## Definition (Space complexity)

The *space complexity* of a program (for a given input) is the number of elementary objects that this program needs to store during its execution.

This number is computed with respect to the size  $n$  of the input data.

## Space complexity: a formal definition

### Definition (Space complexity)

For an algorithm  $T$  and an input  $x$ ,  $DSPACE(T, x)$  denotes the number of cells used during the (deterministic) computation  $T(x)$ . We will note  $DSPACE(T) = O(f(n))$  if  $DSPACE(T, x) = O(f(n))$  with  $n = |x|$  (length of  $x$ ).

Note:  $DSPACE(T)$  is undefined whenever  $T(x)$  does not halt.

## Space complexity: example 1

```
// note: x is an unsorted array
int findMin(int[] x) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        if (x[i] < x[k]) {
            k = i;
        }
    }
    return k;
}
```

## Space complexity: example 1

```
// note: x is an unsorted array
int findMin(int[] x) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        if (x[i] < x[k]) {
            k = i;
        }
    }
    return k;
}
```

$$T(\text{findMin}, n) = n + 2$$

$$T(\text{findMin}, n) = O(n)$$

## Space complexity: example 2

```
// note: x is an unsorted array
void multVect(int[] x, int[][] a) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            a[i][j] = x[i] * x[j]
        }
    }
}
```

## Space complexity: example 2

```
// note: x is an unsorted array
void multVect(int[] x, int[][] a) {
    int k = 0; int n = x.length;
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < n; j++) {
            a[i][j] = x[i] * x[j]
        }
    }
}
```

$$T(\text{multVect}, n) = n \times n + 2$$

$$T(\text{multVect}, n) = O(n^2)$$

# Polynomial space complexity

The class  $PSPACE$  is defined as:

$$PSPACE = \cup_{k \in \mathbb{N}} DSPACE(n^k).$$

$PSPACE$  is the (complexity) class of decision problems that can be solved using a deterministic Turing Machine and a polynomial amount of space ( $\Rightarrow$  polynomial space complexity).

## Non-Deterministic *SPACE*

For nondeterministic Turing machine the space complexity is denoted *NSPACE*.

$NSPACE(T, x)$  denotes the number of cells used by the emphnon-deterministic computation  $T(x)$ . We will note  $NSPACE(T) = O(f(n))$  if  $SPACE(T, x) = O(f(n))$  with  $n = |x|$  (length of  $x$ ).

## Space complexity: some results

### Savitch's theorem:

For a function  $f(n) \geq \log(n)$ :

$$NSPACE(f(n)) \subseteq DSPACE((f(n))^2).$$

### Corollary:

$$PSPACE = NPSPACE$$

This follows directly from the fact that the square of a polynomial function is still a polynomial function.

The set of problems with polynomial space complexity is the same if we consider deterministic or non-deterministic Turing machines. . .

## Relations between time and space

$$P \subseteq NP \subseteq PSPACE$$

$$P \subseteq \text{co-NP} \subseteq PSPACE$$

Note:  $P$  is clearly a subset of  $PSPACE$ , because you cannot use more than polynomial space in only polynomial time.

## Relations between time and space

$$P \subseteq NP \subseteq PSPACE$$

$$P \subseteq \text{co-NP} \subseteq PSPACE$$

