Review
Are NP Problems Hard?
The Structure of NP

Idea: polynomial many-one reductions define an order on problems
The Structure of NP

Idea: polynomial many-one reductions define an order on problems
The Structure of NP

Idea: polynomial many-one reductions define an order on problems
The Structure of NP

Idea: polynomial many-one reductions define an order on problems
The Structure of NP

Idea: polynomial many-one reductions define an order on problems
NP-Hardness and NP-Completeness

Definition 7.1:
(1) A language $H$ is **NP-hard**, if $L \leq_p H$ for every language $L \in \text{NP}$.
(2) A language $C$ is **NP-complete**, if $C$ is NP-hard and $C \in \text{NP}$.

NP-Completeness

- NP-complete problems are the **hardest** problems in NP.
- They constitute the maximal class (wrt. $\leq_p$) of problems within NP.
- They are all equally difficult – an efficient solution to one would solve them all.

Theorem 7.2: If $L$ is NP-hard and $L \leq_p L'$, then $L'$ is NP-hard as well.
Proving NP-Completeness

How to show NP-completeness

To show that $L$ is NP-complete, we must show that every language in NP can be reduced to $L$ in polynomial time.

Alternative approach

Given an NP-complete language $C$, we can show that another language $L$ is NP-complete just by showing that

• $C \leq_p L$
• $L \in$ NP

However: Is there any NP-complete problem at all?

Markus Krötzsch, 5th Nov 2018 Complexity Theory slide 6 of 26
Proving NP-Completeness

How to show NP-completeness
To show that $L$ is NP-complete, we must show that every language in NP can be reduced to $L$ in polynomial time.

Alternative approach
Given an NP-complete language $C$, we can show that another language $L$ is NP-complete just by showing that

- $C \leq_p L$
- $L \in NP$
Proving NP-Completeness

How to show NP-completeness
To show that \( L \) is NP-complete, we must show that every language in NP can be reduced to \( L \) in polynomial time.

Alternative approach
Given an NP-complete language \( C \), we can show that another language \( L \) is NP-complete just by showing that

- \( C \leq_p L \)
- \( L \in \text{NP} \)

However: Is there any NP-complete problem at all?
Is there any NP-complete problem at all?

Of course there is: the word problem for polynomial time NTMs!

**Polytime NTM**

**Input:** A polynomial $p$, a $p$-time bounded NTM $M$, and an input word $w$.

**Problem:** Does $M$ accept $w$ (in time $p(|w|)$)?

Theorem 7.3: Polytime NTM is NP-complete.

Proof: See exercise.
The First NP-Complete Problems

Is there any NP-complete problem at all?

Of course there is: the word problem for polynomial time NTMs!

**Polytime NTM**

Input: A polynomial $p$, a $p$-time bounded NTM $M$, and an input word $w$.

Problem: Does $M$ accept $w$ (in time $p(|w|)$)?

**Theorem 7.3:** **Polytime NTM** is NP-complete.

**Proof:** See exercise.
Further NP-Complete Problem?

**Polytime NTM** is NP-complete, but not very interesting:

- not most convenient to work with
- not of much interest outside of complexity theory

Are there more natural NP-complete problems?
Further NP-Complete Problem?

**Polytime NTM** is NP-complete, but not very interesting:
- not most convenient to work with
- not of much interest outside of complexity theory

Are there more natural NP-complete problems?

Yes, thousands of them!
The Cook-Levin Theorem
Theorem 7.4 (Cook 1970, Levin 1973): \textsc{Sat} is NP-complete.
The Cook-Levin Theorem

Theorem 7.4 (Cook 1970, Levin 1973): $\text{Sat}$ is NP-complete.

Proof:

1. $\text{Sat} \in \text{NP}$
   
   Take satisfying assignments as polynomial certificates for the satisfiability of a formula.
The Cook-Levin Theorem

**Theorem 7.4 (Cook 1970, Levin 1973):** $\text{Sat}$ is NP-complete.

**Proof:**

1. $\text{Sat} \in \text{NP}$
   
   Take satisfying assignments as polynomial certificates for the satisfiability of a formula.

2. $\text{Sat}$ is hard for NP
   
   Proof by reduction from the word problem for NTMs.
Proving the Cook-Levin Theorem

Given:
- a polynomial \( p \)
- a \( p \)-time bounded 1-tape NTM \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}) \)
- a word \( w \)

Intended reduction
Define a propositional logic formula \( \varphi_{p,M,w} \) such that
\( \varphi_{p,M,w} \) is satisfiable if and only if \( M \) accepts \( w \) in time \( p(|w|) \).
Proving the Cook-Levin Theorem

Given:
- a polynomial $p$
- a $p$-time bounded 1-tape NTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$
- a word $w$

Intended reduction
Define a propositional logic formula $\varphi_{p,M,w}$ such that $\varphi_{p,M,w}$ is satisfiable if and only if $M$ accepts $w$ in time $p(|w|)$.

Note
On input $w$ of length $n := |w|$, every computation path of $M$ is of length $\leq p(n)$ and uses $\leq p(n)$ tape cells.

Idea
Use logic to describe a run of $M$ on input $w$ by a formula.
Use propositional variables for describing configurations:

- \( Q_q \) for each \( q \in Q \) means “\( M \) is in state \( q \in Q \)”
- \( P_i \) for each \( 0 \leq i < p(n) \) means “the head is at Position \( i \)”
- \( S_{a,i} \) for each \( a \in \Gamma \) and \( 0 \leq i < p(n) \) means “tape cell \( i \) contains Symbol \( a \)”
Use propositional variables for describing configurations:

- \( Q_q \) for each \( q \in Q \) means “\( M \) is in state \( q \in Q \)”
- \( P_i \) for each \( 0 \leq i < p(n) \) means “the head is at Position \( i \)”
- \( S_{a,i} \) for each \( a \in \Gamma \) and \( 0 \leq i < p(n) \) means “tape cell \( i \) contains Symbol \( a \)”

Represent configuration \((q, p, a_0 \ldots a_p(n))\) by assigning truth values to variables from the set

\[
\overline{C} := \{Q_q, P_i, S_{a,i} \mid q \in Q, \ a \in \Gamma, \ 0 \leq i < p(n)\}
\]

using the truth assignment \( \beta \) defined as

\[
\begin{align*}
\beta(Q_s) &:= \begin{cases} 
1 & s = q \\
0 & s \neq q 
\end{cases} \\
\beta(P_i) &:= \begin{cases} 
1 & i = p \\
0 & i \neq p 
\end{cases} \\
\beta(S_{a,i}) &:= \begin{cases} 
1 & a = a_i \\
0 & a \neq a_i 
\end{cases}
\end{align*}
\]
Proving Cook-Levin: Validating Configurations

We define a formula $\text{Conf}(\overline{C})$ for a set of configuration variables

$$\overline{C} = \{Q_q, P_i, S_{a,i} \mid q \in Q, \ a \in \Gamma, \ 0 \leq i < p(n)\}$$

as follows:

$$\text{Conf}(\overline{C}) := \quad \text{“the assignment is a valid configuration”:}$$

$$\bigvee_{q \in Q} \big( Q_q \land \bigwedge_{q' \neq q} \neg Q_{q'} \big)$$

$$\land \bigvee_{p < p(n)} \big( P_p \land \bigwedge_{p' \neq p} \neg P_{p'} \big)$$

$$\land \bigwedge_{0 \leq i < p(n)} \bigwedge_{a \in \Gamma} \big( S_{a,i} \land \bigwedge_{b \neq a \in \Gamma} \neg S_{b,i} \big)$$

“TM in exactly one state $q \in Q$”

“head in exactly one position $p \leq p(n)$”

“exactly one $a \in \Gamma$ in each cell”
For an assignment $\beta$ defined on variables in $\overline{C}$ define

$$\text{conf}(\overline{C}, \beta) := \left\{ (q, p, w_0 \ldots w_{p(n)}) \mid \begin{array}{l}
\beta(Q_q) = 1, \\
\beta(P_p) = 1, \\
\beta(S_{w_i,i}) = 1 \text{ for all } 0 \leq i < p(n)
\end{array} \right\}$$

Note: $\beta$ may be defined on other variables besides those in $\overline{C}$. 
Proving Cook-Levin: Validating Configurations

For an assignment $\beta$ defined on variables in $\overline{C}$ define

$$
\text{conf}(\overline{C}, \beta) := \left\{ \begin{array}{l}
(q, p, w_0 \ldots w_{p(n)}) | \\
\beta(Q_q) = 1, \\
\beta(P_p) = 1, \\
\beta(S_{w_i, i}) = 1 \text{ for all } 0 \leq i < p(n)
\end{array} \right. 
$$

Note: $\beta$ may be defined on other variables besides those in $\overline{C}$.

**Lemma 7.5:** If $\beta$ satisfies $\text{Conf}(\overline{C})$ then $|\text{conf}(\overline{C}, \beta)| = 1$.

We can therefore write $\text{conf}(\overline{C}, \beta) = (q, p, w)$ to simplify notation.
For an assignment $\beta$ defined on variables in $C$ define

$$
\text{conf}(C, \beta) := \begin{cases}
\beta(Q_q) = 1, \\
(q, p, w_0 \ldots w_{p(n)}) & \beta(P_p) = 1, \\
\beta(S_{w_i,i}) = 1 \text{ for all } 0 \leq i < p(n)
\end{cases}
$$

Note: $\beta$ may be defined on other variables besides those in $C$.

**Lemma 7.5:** If $\beta$ satisfies $\text{Conf}(C)$ then $|\text{conf}(C, \beta)| = 1$.
We can therefore write $\text{conf}(C, \beta) = (q, p, w)$ to simplify notation.

**Observations:**

- $\text{conf}(C, \beta)$ is a potential configuration of $M$, but it may not be reachable from the start configuration of $M$ on input $w$.
- Conversely, every configuration $(q, p, w_1 \ldots w_{p(n)})$ induces a satisfying assignment $\beta$ or which $\text{conf}(C, \beta) = (q, p, w_1 \ldots w_{p(n)})$. 
Consider the following formula $\text{Next}(\overline{C}, \overline{C}')$ defined as

$$\text{Conf}(\overline{C}) \land \text{Conf}(\overline{C}') \land \text{NoChange}(\overline{C}, \overline{C}') \land \text{Change}(\overline{C}, \overline{C}')$$

**NoChange** := $\bigvee_{0 \leq p < p(n)} (P_p \land \bigwedge_{i \neq p, a \in \Gamma} (S_{a,i} \rightarrow S'_{a,i}))$

**Change** := $\bigvee_{0 \leq p < p(n)} (P_p \land \bigvee_{q \in Q} (Q_q \land S_{a,p} \land \bigvee_{(q', b, D) \in \delta(q, a)} (Q'_{q'} \land S'_{b,p} \land P'_{D(p)})))$

where $D(p)$ is the position reached by moving in direction $D$ from $p$. 
Consider the following formula \( \text{Next}(\bar{C}, \bar{C}') \) defined as

\[
\text{Conf}(\overline{C}) \land \text{Conf}(\overline{C}') \land \text{NoChange}(\overline{C}, \overline{C}') \land \text{Change}(\overline{C}, \overline{C}').
\]

\[
\text{NoChange} := \bigvee_{0 \leq p < p(n)} \left( P_p \land \bigwedge_{i \neq p, a \in \Gamma} (S_{a,i} \rightarrow S'_{a,i}) \right)
\]

\[
\text{Change} := \bigvee_{0 \leq p < p(n)} \left( P_p \land \bigvee_{q \in Q} (Q_q \land S_{a,p} \land \bigvee_{(q',b,D) \in \delta(q,a)} (Q'_{q'} \land S'_{b,p} \land P'_{D(p)})) \right)
\]

where \( D(p) \) is the position reached by moving in direction \( D \) from \( p \).

**Lemma 7.6:** For any assignment \( \beta \) defined on \( \overline{C} \cup \overline{C}' \):

\( \beta \) satisfies \( \text{Next}(\overline{C}, \overline{C}') \) if and only if \( \text{conf}(\overline{C}, \beta) \vdash_M \text{conf}(\overline{C}', \beta) \)
Defined so far:

- \( \text{Conf}(\overline{C}) \): \( \overline{C} \) describes a potential configuration
- \( \text{Next}(\overline{C}, \overline{C}') \): \( \text{conf}(\overline{C}, \beta) \vdash_M \text{conf}(\overline{C}', \beta) \)
Proving Cook-Levin: Start and End

Defined so far:

- \text{Conf}(\overline{C}) : \overline{C} describes a potential configuration
- \text{Next}(\overline{C}, \overline{C}') : \text{conf}(\overline{C}, \beta) \vdash_M \text{conf}(\overline{C}', \beta)

Start configuration:

For an input word \( w = w_0 \cdots w_{n-1} \in \Sigma^* \), we define:

\[
\text{Start}_{M,w}(\overline{C}) := \text{Conf}(\overline{C}) \land Q_{q_0} \land P_0 \land \bigwedge_{i=0}^{n-1} S_{w,i} \land \bigwedge_{i=n}^{p(n)-1} S_{\omega,i}
\]

Then an assignment \( \beta \) satisfies \( \text{Start}_{M,w}(\overline{C}) \) if and only if \( \overline{C} \) represents the start configuration of \( M \) on input \( w \).
Proving Cook-Levin: Start and End

Defined so far:
- \( \text{Conf}(\overline{C}) \): \( \overline{C} \) describes a potential configuration
- \( \text{Next}(\overline{C}, \overline{C}') \): \( \text{conf}(\overline{C}, \beta) \vdash_M \text{conf}(\overline{C}', \beta) \)

Start configuration:
For an input word \( w = w_0 \cdots w_{n-1} \in \Sigma^* \), we define:

\[
\text{Start}_{M,w}(\overline{C}) := \text{Conf}(\overline{C}) \land Q_{q_0} \land P_0 \land \bigwedge_{i=0}^{n-1} S_{w_i,i} \land \bigwedge_{i=n}^{p(n)-1} S_{w,i}
\]

Then an assignment \( \beta \) satisfies \( \text{Start}_{M,w}(\overline{C}) \) if and only if \( \overline{C} \) represents the start configuration of \( M \) on input \( w \).

Accepting stop configuration:

\[
\text{Acc-Conf}(\overline{C}) := \text{Conf}(\overline{C}) \land Q_{q_{\text{accept}}}
\]

Then an assignment \( \beta \) satisfies \( \text{Acc-Conf}(\overline{C}) \) if and only if \( \overline{C} \) represents an accepting configuration of \( M \).
Since $\mathcal{M}$ is $p$-time bounded, each run may contain up to $p(n)$ steps
$\leadsto$ we need one set of configuration variables for each

**Propositional variables**

$Q_{q,t}$ for all $q \in Q$, $0 \leq t \leq p(n)$ means “at time $t$, $\mathcal{M}$ is in state $q \in Q$”

$P_{i,t}$ for all $0 \leq i, t \leq p(n)$ means “at time $t$, the head is at position $i$”

$S_{a,i,t}$ for all $a \in \Gamma$ and $0 \leq i, t \leq p(n)$ means “at time $t$, tape cell $i$ contains symbol $a$”

**Notation**

$\overline{C}_t := \{Q_{q,t}, P_{i,t}, S_{a,i,t} \mid q \in Q, 0 \leq i \leq p(n), a \in \Gamma\}$
Proving Cook-Levin: The Formula

Given:
- a polynomial $p$
- a $p$-time bounded 1-tape NTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}})$
- a word $w$

We define the formula $\varphi_{p,M,w}$ as follows:

\[
\varphi_{p,M,w} := \text{Start}_{M,w} (\overline{C}_0) \land \bigvee_{0 \leq t \leq p(n)} \left( \text{Acc-Conf}(\overline{C}_t) \land \bigwedge_{0 \leq i < t} \text{Next}(\overline{C}_i, \overline{C}_{i+1}) \right)
\]

“$C_0$ encodes the start configuration” and for some polynomial time $t$:
- “$M$ accepts after $t$ steps” and “$\overline{C}_0, \ldots, \overline{C}_t$ encode a computation path”

**Lemma 7.7:** $\varphi_{p,M,w}$ is satisfiable if and only if $M$ accepts $w$ in time $p(|w|)$.

Note that an accepting or rejecting stop configuration has no successor.
The Cook-Levin Theorem

**Theorem 7.4 (Cook 1970, Levin 1973):** $\text{Sat}$ is NP-complete.

**Proof:**

1. **$\text{Sat} \in \text{NP}**

   Take satisfying assignments as polynomial certificates for the satisfiability of a formula.

2. **$\text{Sat}$ is hard for NP**

   Proof by reduction from the word problem for NTMs.
Further NP-complete Problems
Towards More NP-Complete Problems

Starting with **Sat**, one can readily show more problems $P$ to be NP-complete, each time performing two steps:

1. Show that $P \in \text{NP}$
2. Find a known NP-complete problem $P'$ and reduce $P' \leq_p P$

Thousands of problems have now been shown to be NP-complete. (See Garey and Johnson for an early survey)
Towards More NP-Complete Problems

Starting with SAT, one can readily show more problems P to be NP-complete, each time performing two steps:

1. Show that $P \in \text{NP}$
2. Find a known NP-complete problem $P'$ and reduce $P' \leq_p P$

Thousands of problems have now been shown to be NP-complete. (See Garey and Johnson for an early survey)

In this course:

- $\leq_p \text{CLIQUE}$
- $\leq_p \text{INDEPENDENT SET}$
- $\text{SAT} \leq_p \text{3-SAT}$
- $\leq_p \text{DIR. HAMILTONIAN PATH}$
- $\leq_p \text{SUBSET SUM}$
- $\leq_p \text{KNAPSACK}$
NP-Completeness of \textbf{Clique}

\textbf{Theorem 7.8: Clique} is NP-complete.

\textbf{Clique}: Given $G,k$, does $G$ contain a clique of order $\geq k$?

\textbf{Proof}:

(1) \textbf{Clique} $\in$ NP

Take the vertex set of a clique of order $k$ as a certificate.

(2) \textbf{Clique} is NP-hard

We show $\text{Sat} \leq_p \text{Clique}$

To every CNF-formula $\varphi$ assign a graph $G_\varphi$ and a number $k_\varphi$ such that

\[ \varphi \text{ satisfiable } \iff G_\varphi \text{ contains clique of order } k_\varphi \]
SAT $\leq_p$ CLIQUE

To every CNF-formula $\varphi$ assign a graph $G_{\varphi}$ and a number $k_{\varphi}$ such that

$\varphi$ satisfiable if and only if $G_{\varphi}$ contains clique of order $k_{\varphi}$

Given $\varphi = C_1 \land \cdots \land C_k$:

- Set $k_{\varphi} := k$
- For each clause $C_j$ and literal $L \in C_j$ add a vertex $v_{L,j}$
- Add edge $\{v_{L,j}, v_{K,i}\}$ if $i \neq j$ and $L \land K$ is satisfiable (that is: if $L \neq \neg K$ and $\neg L \neq K$)

Example 7.9:

\[
\begin{align*}
C_1 & : (X \lor Y \lor \neg Z) \\
C_2 & : (X \lor \neg Y) \\
C_3 & : (\neg X \lor Z)
\end{align*}
\]
To every CNF-formula $\varphi$ assign a graph $G_\varphi$ and a number $k_\varphi$ such that

$\varphi$ satisfiable if and only if $G_\varphi$ contains clique of order $k_\varphi$

Given $\varphi = C_1 \land \cdots \land C_k$:

- Set $k_\varphi := k$
- For each clause $C_j$ and literal $L \in C_j$ add a vertex $v_{L,j}$
- Add edge $\{v_{L,j}, v_{K,i}\}$ if $i \neq j$ and $L \land K$ is satisfiable (that is: if $L \neq \neg K$ and $\neg L \neq K$)

**Example 7.9:**

\[
\left( X \lor Y \lor \neg Z \right) \land \left( X \lor \neg Y \right) \land \left( \neg X \lor Z \right)
\]

\[
C_1 \land C_2 \land C_3
\]
To every CNF-formula $\varphi$ assign a graph $G_\varphi$ and a number $k_\varphi$ such that

$\varphi$ satisfiable if and only if $G_\varphi$ contains clique of order $k_\varphi$

Given $\varphi = C_1 \land \cdots \land C_k$:

- Set $k_\varphi := k$
- For each clause $C_j$ and literal $L \in C_j$ add a vertex $v_{L,j}$
- Add edge $\{v_{L,j}, v_{K,i}\}$ if $i \neq j$ and $L \land K$ is satisfiable (that is: if $L \neq \neg K$ and $\neg L \neq K$)

Example 7.9:

$\begin{align*}
(C_1 &\equiv (X \lor Y \lor \neg Z) \land (X \lor \neg Y) \land (\neg X \lor Z)) \\
C_2 &\equiv (X \lor Y) \\
C_3 &\equiv (\neg X \lor Z)
\end{align*}$
SAT $\leq_p$ CLIQUE

To every CNF-formula $\varphi$ assign a graph $G_\varphi$ and a number $k_\varphi$ such that

$\varphi$ satisfiable if and only if $G_\varphi$ contains clique of order $k_\varphi$

Given $\varphi = C_1 \land \cdots \land C_k$:

- Set $k_\varphi := k$
- For each clause $C_j$ and literal $L \in C_j$ add a vertex $v_{L,j}$
- Add edge $\{v_{L,j}, v_{K,i}\}$ if $i \neq j$ and $L \land K$ is satisfiable (that is: if $L \neq \neg K$ and $\neg L \neq K$)

Example 7.9:

$(X \lor Y \lor \neg Z) \land (X \lor \neg Y) \land (\neg X \lor Z)$

Given $\varphi = C_1 \land C_2 \land C_3$:

- Set $k_\varphi := 3$
- For each clause $C_j$ and literal $L \in C_j$ add a vertex $v_{L,j}$
- Add edge $\{v_{L,j}, v_{K,i}\}$ if $i \neq j$ and $L \land K$ is satisfiable (that is: if $L \neq \neg K$ and $\neg L \neq K$)
SAT $\leq_p$ CLIQUE

To every CNF-formula $\varphi$ assign a graph $G_{\varphi}$ and a number $k_{\varphi}$ such that

$\varphi$ satisfiable if and only if $G_{\varphi}$ contains clique of order $k_{\varphi}$

Given $\varphi = C_1 \land \cdots \land C_k$:

- Set $k_{\varphi} := k$
- For each clause $C_j$ and literal $L \in C_j$ add a vertex $v_{L,j}$
- Add edge $\{u_{L,j}, v_{K,i}\}$ if $i \neq j$ and $L \land K$ is satisfiable
  (that is: if $L \neq \neg K$ and $\neg L \neq K$

Correctness:

$G_{\varphi}$ has clique of order $k$ iff $\varphi$ is satisfiable.

Complexity:

The reduction is clearly computable in polynomial time.
NP-Completeness of **Independent Set**

**Input:** An undirected graph $G$ and a natural number $k$

**Problem:** Does $G$ contain $k$ vertices that share no edges (independent set)?

**Theorem 7.10:** Independent Set is NP-complete.
NP-Completeness of **INDEPENDENT SET**

**INDEPENDENT SET**

Input: An undirected graph $G$ and a natural number $k$

Problem: Does $G$ contain $k$ vertices that share no edges (independent set)?

**Theorem 7.10**: **INDEPENDENT SET** is NP-complete.

**Proof**: Hardness by reduction $\text{CLIQUE} \leq_p \text{INDEPENDENT SET}$:

- Given $G := (V, E)$ construct $\overline{G} := (V, \{\{u, v\} \mid \{u, v\} \notin E \text{ and } u \neq v\})$
NP-Completeness of **INDEPENDENT SET**

**INDEPENDENT SET**

Input: An undirected graph $G$ and a natural number $k$

Problem: Does $G$ contain $k$ vertices that share no edges (independent set)?

**Theorem 7.10:** **INDEPENDENT SET** is NP-complete.

**Proof:** Hardness by reduction $\textbf{CLIQUE} \leq_p \textbf{INDEPENDENT SET}$:

- Given $G := (V, E)$ construct $\overline{G} := (V, \{\{u, v\} \mid \{u, v\} \notin E \text{ and } u \neq v\})$

- A set $X \subseteq V$ induces a clique in $G$ iff $X$ induces an independent set in $\overline{G}$.

- Reduction: $G$ has a clique of order $k$ iff $\overline{G}$ has an independent set of order $k$.

\[\square\]
Summary and Outlook

NP-complete problems are the hardest in NP

Polynomial runs of NTMs can be described in propositional logic (Cook-Levin)

\textbf{CLIQUE} and \textbf{INDEPENDENT SET} are also NP-complete

What’s next?

- More examples of problems
- The limits of NP
- Space complexities