Algorithms for Programming Contests - Week 06

Prof. Dr. Javier Esparza,
Vincent Fischer,
Jakob Schulz,
conpra@model.cit.tum.de

18. November 2025

Brute Force

Definition (Brute Force)

Systematically enumerate **all** solution candidates and test whether each candidate satisfies solution requirements.

a.k.a. **Exhaustive Search** or **Generate and Test**.

Brute Force: Example

Task:

Given a combination lock of 4 decimal digits, find the right key for the lock.

Brute Force Solution:

Test all combinations from 0000 to 9999 until the right one is found.

Pros and Cons

Pros

- Simple
- Sound and complete - will find (optimum) solution if there exists one
- Used in safety critical applications because of its simplicity
- Serves as a benchmark for faster/more error-prone methods

Cons

- Inefficient
- Not feasible for large input sizes (combinatorial explosion)

Combinatorial Explosion

Brute-forcing passwords (without any fancy business!)

Allowed	Length	Search Space	Time
0-9	5	10^5	$<$1s
	10	10^{10}	20s
	15	10^{15}	23 days

Combinatorial Explosion

Brute-forcing passwords (without any fancy business!)

Allowed	Length	Search Space	Time
0-9	5	10^5	$<$1s
	10	10^{10}	20s
	15	10^{15}	23 days
a-zA-Z	5	52^5	$\sim$1s
	10	52^{10}	9.1 years
	15	52^{15}	3.5 billion years

Combinatorial Explosion

Brute-forcing passwords (without any fancy business!)

Allowed	Length	Search Space	Time
0-9	5	10^5	$<$1s
	10	10^{10}	20s
	15	10^{15}	23 days
a-zA-Z	5	52^5	$\sim$1s
	10	52^{10}	9.1 years
	15	52^{15}	3.5 billion years
Printable `ASCII`	5	95^5	15s
	10	95^{10}	3795 years
	15	95^{15}	2100 x **age of universe**

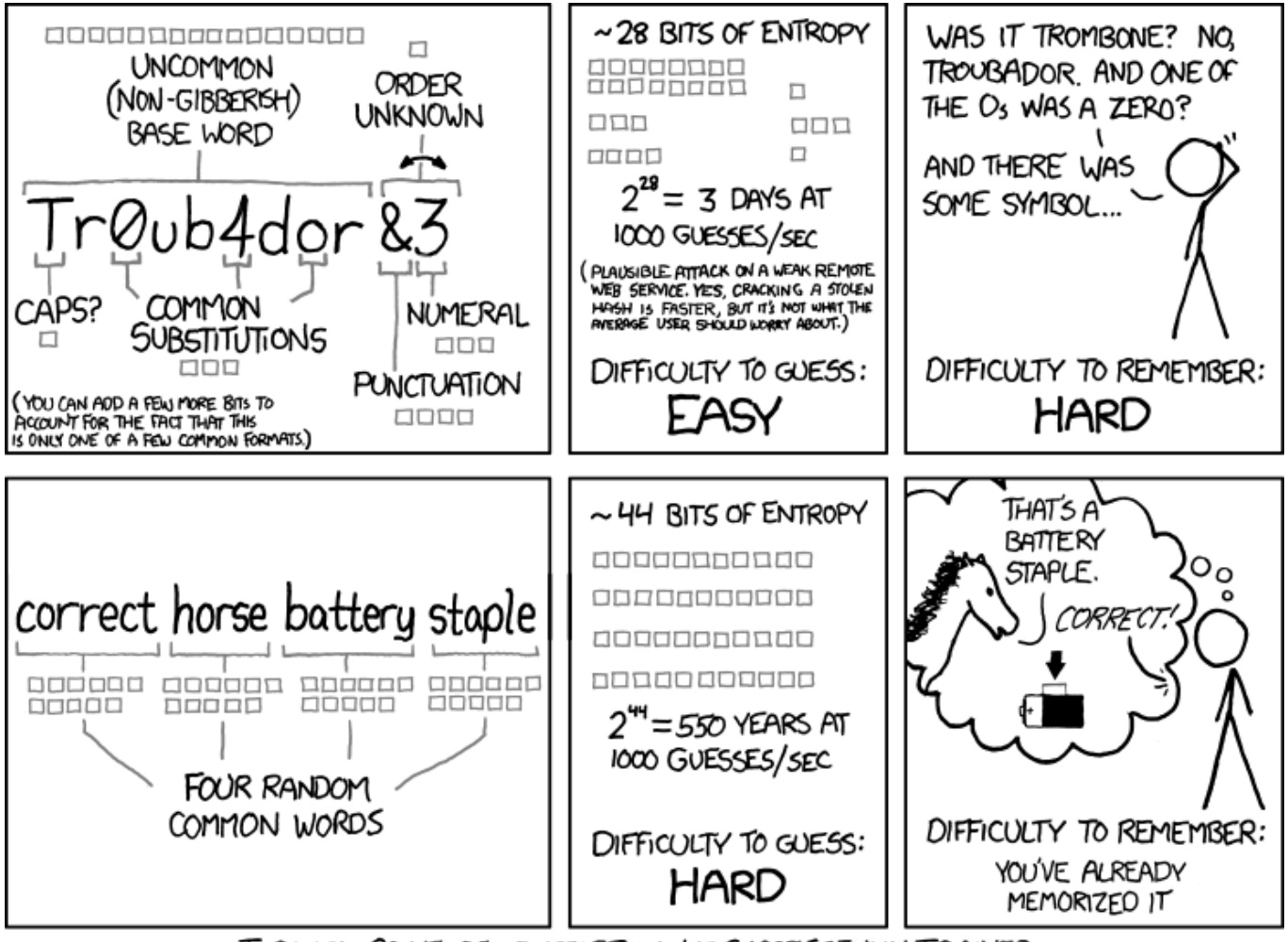
Combinatorial Explosion

Brute-forcing passwords (without any fancy business!)

Allowed	Length	Search Space	Time
0-9	5	10^5	$<$1s
	10	10^{10}	20s
	15	10^{15}	23 days
a-zA-Z	5	52^5	$\sim$1s
	10	52^{10}	9.1 years
	15	52^{15}	3.5 billion years
Printable `ASCII`	5	95^5	15s
	10	95^{10}	3795 years
	15	95^{15}	2100 x **age of universe**

You can see that it quickly grows out of hand!

Relevant XKCD



Dealing with large search spaces

- Reorder search space: **start with the most promising ones!**
  e.g. it makes sense for a password cracker to search for passwords like 1234 or *password* first.
- Reduce search space

Example: Reduce Search Space

Find non-trivial divisor of n

Brute force approach: Enumerate all numbers $i \in [2, n-1]$ and check if i divides n.

Example: Reduce Search Space

Find non-trivial divisor of n

Brute force approach: Enumerate all numbers $i \in [2, n-1]$ and check if i divides n.

But we can use our domain knowledge and search only upto $\lceil \sqrt{n} \rceil$

- 32-bit number has up to 10 decimal digits.
  $\Longrightarrow$ square root has up to 5.
  $\Longrightarrow$ 10^5 tests instead of 10^{10} tests

Example: Reduce Search Space

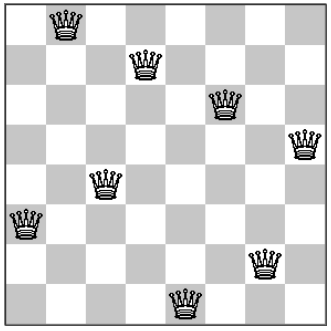
Find non-trivial divisor of n

Brute force approach: Enumerate all numbers $i \in [2, n-1]$ and check if i divides n.

But we can use our domain knowledge and search only upto $\lceil \sqrt{n} \rceil$

- 32-bit number has up to 10 decimal digits.
  $\Longrightarrow$ square root has up to 5.
  $\Longrightarrow$ 10^5 tests instead of 10^{10} tests
- 64-bit number has up to 20 decimal digits.
  $\Longrightarrow$ square root has up to 10.
  $\Longrightarrow$ 10^{10} tests instead of 10^{20}.

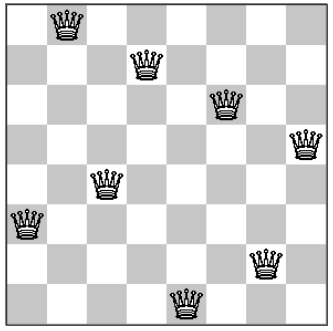
Example: Reduce Search Space



Queens Problem

Place 8 queens on a chess board such that they cannot threaten each other.

Example: Reduce Search Space

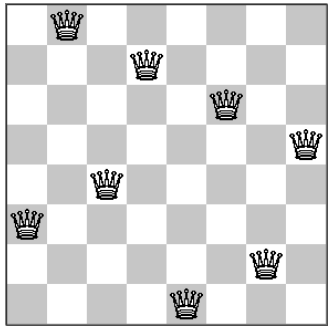


Queens Problem

Place 8 queens on a chess board such that they cannot threaten each other.

- Very naive: Each tile can have queen 1, queen 2, ..., queen 8 or no queen: $9^{64} \approx 1.2 \cdot 10^{61}$ configurations.

Example: Reduce Search Space

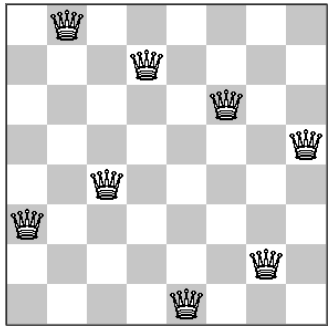


Queens Problem

Place 8 queens on a chess board such that they cannot threaten each other.

- Very naive: Each tile can have queen 1, queen 2, ..., queen 8 or no queen: $9^{64} \approx 1.2 \cdot 10^{61}$ configurations.
- Naive: Each tile can have a queen or not: $2^{64} \approx 1.8 \cdot 10^{19}$ configurations.

Example: Reduce Search Space

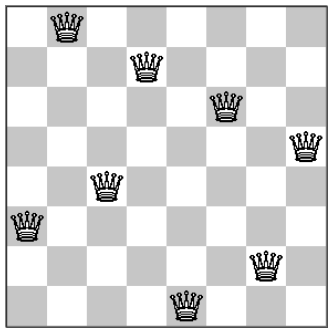


Queens Problem

Place 8 queens on a chess board such that they cannot threaten each other.

- Very naive: Each tile can have queen 1, queen 2, ..., queen 8 or no queen: $9^{64} \approx 1.2 \cdot 10^{61}$ configurations.
- Naive: Each tile can have a queen or not: $2^{64} \approx 1.8 \cdot 10^{19}$ configurations.
- Better: place eight queens, one after another: $64 \cdot 63 \cdot \ldots \cdot 57 = \frac{64!}{56!} \approx 1.8 \cdot 10^{14}$ configurations.

Example: Reduce Search Space



Queens Problem

Place 8 queens on a chess board such that they cannot threaten each other.

- Very naive: Each tile can have queen 1, queen 2, ..., queen 8 or no queen: $9^{64} \approx 1.2 \cdot 10^{61}$ configurations.
- Naive: Each tile can have a queen or not: $2^{64} \approx 1.8 \cdot 10^{19}$ configurations.
- Better: place eight queens, one after another: $64 \cdot 63 \cdot \ldots \cdot 57 = \frac{64!}{56!} \approx 1.8 \cdot 10^{14}$ configurations.
- Even better: choose 8 tiles to place queens on: $\binom{64}{8} \approx 4.4 \cdot 10^{9}$ configurations.

Problems in Practice

In practice, things are not obvious:

1. How can we represent candidates without unnecessarily increasing the search space?
2. How can we (efficiently) enumerate candidates?

Problems in Practice

In practice, things are not obvious:

1. How can we represent candidates without unnecessarily increasing the search space?
2. How can we (efficiently) enumerate candidates?
3. **How large is the search space anyway???**

Representing Candidates

- If order of something is irrelevant: do *not* consider reorderings!!!
- Use mathematical structure that can easily be enumerated, e.g.
  - Tuples (sorted, bounded, with bounds on sum, ...)
  - Sets/Multisets (containing/excluding elements, with bounds on size, ...)
  - Permutations (with fied/arbitrary number of cycles, cycles of certain length (e.g. fixed points), ...)
  - Partitions (into fixed/arbitrary number of classes, where certain elements belong to same class, ...)
  - (Equivalence) Relations
  - ...
- ...or use more difficult structure, and think about enumeration later:
  - Graphs (directed, weighted, ...)
  - Trees (rooted, ...)
  - ...
  - Combinations of all of the above

Representing Candidates

All of the "easily enumerable" structures from last slide can be represented as constrained tuples!

- Sets/Multisets: ordered tuples
- Permutations: tuples with pairwise different elements
- Partitions (+ equivalence relations): see next slides :)
- Relations: Sets of Pairs

Example: Sets

- Consider sets of elements in $[n]$
- Idea: use tuple $(x_1, \ldots, x_k)$ to represent $\{x_1, \ldots, x_k\}$

Example: Sets

- Consider sets of elements in $[n]$
- Idea: use tuple $(x_1, \ldots, x_k)$ to represent $\{x_1, \ldots, x_k\}$
- But: unnecessarily increases search space: $(1, 4, 5)$, $(5, 1, 4)$ and $(1, 4, 5, 5)$ all represent $\{1, 4, 5\}$

Example: Sets

- Consider sets of elements in $[n]$
- Idea: use tuple $(x_1, \ldots, x_k)$ to represent $\{x_1, \ldots, x_k\}$
- But: unnecessarily increases search space: $(1, 4, 5)$, $(5, 1, 4)$ and $(1, 4, 5, 5)$ all represent $\{1, 4, 5\}$
- Solution: only consider tuples $(x_1, \ldots, x_k)$ where $x_1 < \ldots < x_k$

Example: Sets

Definition

Let X be a set, and $k \in \mathbb{N}$. Define $\binom{X}{k} := \{Y \subseteq X \mid |Y| = k\}$.

Lemma

Let $(X, <)$ be a totally ordered set. Then

$$\{(x_1, \ldots, x_k) \in X^k \mid x_1 < \ldots < x_k\} \to \binom{X}{k},$$
$$(x_1, \ldots, x_k) \mapsto \{x_1, \ldots, x_k\}$$

is a bijection.

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$

$$\downarrow$$

$$\{ \qquad\qquad\qquad\qquad\qquad\qquad \}$$

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$

$$\downarrow$$

$$\{\{1, 3\} \qquad\qquad \}$$

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$

$$\downarrow$$

$$\{\{1, 3\}, \{4\} \qquad\qquad\qquad\qquad \}$$

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$

$$\downarrow$$

$$\{\{1, 3\}, \{4\}, \{2, 5\} \qquad\qquad \}$$

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- But: again, search space increased unnecessarily: $(1, 3, 1)$, $(1, 2, 1)$ and $(2, 1, 2)$ all represent $\{\{1, 3\}, \{2\}\}$

Example: Partitions

- Consider partitions of $[n]$
- Idea: use tuple $(x_1, \ldots, x_n)$, where each x_i indicates which equivalence class i belongs to

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- But: again, search space increased unnecessarily: $(1, 3, 1)$, $(1, 2, 1)$ and $(2, 1, 2)$ all represent $\{\{1, 3\}, \{2\}\}$
- Solution: only consider lexicographically smallest tuple (details: later...)

Enumerating Candidates

For enumerating constrained tuples in lexicographic order: wonderful answer on StackOverflow!

Definition

Let $S \subseteq [n]^k$ be a set of tuples.

- A tuple $(x_1, \ldots, x_l)$ with $l \leq k$ is called *valid prefix* in S if there exist $x_{l+1}, \ldots, x_k$ s.t. $(x_1, \ldots, x_l, x_{l+1}, \ldots, x_k) \in S$.
- Let $x = (x_1, \ldots, x_k) \in S$. We say x *can be incremented at* $i \in [k]$ *to* $v \in [n]$ if
  1. $x_i < v$ and
  2. $(x_1, \ldots, x_{i-1}, v)$ is a viable prefix.
- $x \in S$ is called *incrementable at* $i \in [k]$ if there exists a $v \in [n]$ s.t. x can be incremented at i to v.
- $x \in S$ is called *incrementable* if there exists an $i \in [k]$ s.t. x is incrementable at i.

Enumerating Candidates

Algorithm 1 Enumerating $S \subseteq [n]^k$ in lex. order

$x \leftarrow$ lex. smallest element of S
Output x
while x is incrementable **do**
    Let $i \in [k]$ be highest index at which x is incrementable
    Let $v \in [n]$ be the smallest element to which x can be incremented at i
    $x' \leftarrow (x_1, \ldots, x_{i-1}, v)$
    Let $s = (s_{i+1}, \ldots, s_k)$ be the lex. smallest suffix s.t. $(x', s) \in S$
    $x \leftarrow (x', s)$
    Output x
end while

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- If $k > n$, no such tuples exist.
- If $k = 0$, the empty tuple $()$ is the only one, corresponding to the empty set $\emptyset$.
- In the following, $1 \leq k \leq n$.

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)
  - resp. $k - l \leq |[x_l + 1, n]|$

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)
  - resp. $k - l \leq |[x_l + 1, n]|$
  - resp. $k - l \leq n - x_l$

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)
  - resp. $k - l \leq |[x_l + 1, n]|$
  - resp. $k - l \leq n - x_l$ (intuitive!)

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)
  - resp. $k - l \leq |[x_l + 1, n]|$
  - resp. $k - l \leq n - x_l$ (intuitive!)
- Hence, $(x_1, \ldots, x_k) \in S$ can be incremented at i to v iff $v > x_i$ and $k - i \leq n - v$

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)
  - resp. $k - l \leq |[x_l + 1, n]|$
  - resp. $k - l \leq n - x_l$ (intuitive!)
- Hence, $(x_1, \ldots, x_k) \in S$ can be incremented at i to v iff $v > x_i$ and $k - i \leq n - v$
- Note: here, if x can be incremented at i to v', and $x_i < v < v'$, x can also be incremented at i to v

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)
  - resp. $k - l \leq |[x_l + 1, n]|$
  - resp. $k - l \leq n - x_l$ (intuitive!)
- Hence, $(x_1, \ldots, x_k) \in S$ can be incremented at i to v iff $v > x_i$ and $k - i \leq n - v$
- Note: here, if x can be incremented at i to v', and $x_i < v < v'$, x can also be incremented at i to v
- Hence, $(x_1, \ldots, x_k)$ is incrementable at i iff $k - i \leq n - (x_i + 1)$, and in that case, $x_i + 1$ is the smallest v to which x can be incremented at i

Example: Enumerating Sets

Enumerating $S := \{(x_1, \ldots, x_k) \in [n]^k \mid x_1 < \ldots < x_k\}$ in lex. order:

- Smallest tuple: $(1, 2, \ldots, k)$.
- A prefix $(x_1, \ldots, x_l)$ with $l \leq k$ is valid iff $x_1 < \ldots < x_l$ and...
  - $\exists x_{l+1}, \ldots, x_k : x_l < x_{l+1} < \ldots < x_k \leq n$
  - resp. $\{(x_{l+1}, \ldots, x_k) \in [x_l + 1, n]^{k-l} \mid x_{l+1} < \ldots < x_n\} \neq \emptyset$
  - resp. $\binom{[x_l+1,n]}{k-l} \neq \emptyset$ (see Lemma!)
  - resp. $k - l \leq |[x_l + 1, n]|$
  - resp. $k - l \leq n - x_l$ (intuitive!)
- Hence, $(x_1, \ldots, x_k) \in S$ can be incremented at i to v iff $v > x_i$ and $k - i \leq n - v$
- Note: here, if x can be incremented at i to v', and $x_i < v < v'$, x can also be incremented at i to v
- Hence, $(x_1, \ldots, x_k)$ is incrementable at i iff $k - i \leq n - (x_i + 1)$, and in that case, $x_i + 1$ is the smallest v to which x can be incremented at i
- For a valid prefix $x' = (x_1, \ldots, x_l)$, the lex. smallest suffix s s.t. $(x', s) \in S$ is $(x_l + 1, x_l + 2, \ldots, x_l + k - l)$

Enumerating Sets: Remarks

- For small k, one can alternatively use nested loops.
  - Most programmers draw the line at $k \leq 3$.
- Many itertools packages support sorted tuple enumeration.

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(\ ,\ ,\ ,\ ,\ ,\ ,\ ,\ )$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, \ , 1, \ , \ , \ , \ , \ )$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, \ , 1, \ , \ , \ , \ , \ )$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, 2, 1, \ , 2, \ , \ , \ )$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, 2, 1, \ , 2, \ , \ , \ )$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, 2, 1, 3, 2, \ , \ , \ )$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, 2, 1, 3, 2, \ , \ , \ )$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, 2, 1, 3, 2, 4, 4, 4)$$

Example: Enumerating Partitions

Reminder:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$

- Want to consider only lex. smallest sequence
- Can be obtained from above example by going over equivalence classes in order in which they appear in the tuple, and assign identifiers increasingly:

$$(1, 3, 1, 2, 3, 4, 4, 4)$$
$$\downarrow$$
$$\{\{1, 3\}, \{4\}, \{2, 5\}, \{6, 7, 8\}\}$$
$$\downarrow$$
$$(1, 2, 1, 3, 2, 4, 4, 4)$$

Example: Enumerating Partitions

$$(1, 2, 1, 3, 2, 4, 4, 4)$$

Note:

- Every valid tuple starts with 1
- There must not be any "gaps", i.e. whenever some x_i appears, there must be some $j < i$ where $x_j + 1 \geq x_i$
- These two conditions are also sufficient for a tuple to be a valid!

Example: Enumerating Partitions

Enumerating
$S :=$ set of all partition tuples of $[n]$ as obtained in last slides:

- Smallest tuple: $(1, 1, \ldots, 1)$
- $(x_1, \ldots, x_k) \in S$ is incrementable at $i > 1$ iff $\exists j < i : x_j = x_i$
  - Equivalently: iff $x_i \leq \max_{j<i} x_j$
  - Hence: maintain vector $(\max_{j<i} x_j)_i$ to decide this in $\mathcal{O}(1)$
  - $x \in S$ is never incrementable at $i = 1$
- For a valid prefix x', the lex. smallest suffix s.t. $(x', s) \in S$ is $(1, 1, \ldots, 1)$

Enumerating Partitions: Remarks

- Implementation can generate next partition in amortized constant time if maximum vector is maintained
- Itertools packages usually do not support this enumeration

Enumerating Permutations

Enumerating all permutations $\pi \in S_n := \{\pi : [n] \to [n] \mid \pi \text{ bijective}\}$ in lex. order can be done similarly.

A succinct description of the increment step:

1. Find largest index k: $a[k] < a[k+1]$
   If k does not exist then this is the last permutation
2. Find largest index l: $a[k] < a[l]$
3. Swap values $a[k]$ and $a[l]$
4. Reverse sequence from $a[k+1]$ up to the final element $a[n]$.

Each increment operation takes $\mathcal{O}(n)$ time.

Enumerating Tuples: Remarks

- When tuples need not be incremented in lex. order, slightly faster enumeration algorithms might exist.
  - E.g. Steinhaus-Johnson-Trotter algorithm for permutations
  - E.g. Gray Code for $[n]^k$
- However, keep in mind that in most cases, these only give a very slight constant-factor improvement when using them in a brute-force algorithm – except if enumeration is in-place and checking a candidate is sublinear.
- More relevant: as number of permutations, partitions, etc. is very large, usually it does *not* make sense to construct and iterate over an array with all of them. Instead, generate next tuple on the fly.

Enumerating complex objects

When enumerating more complex objects, e.g. graphs/trees up to isomorphism:

- Usually very difficult!
- Sometimes redundant enumeration can be faster than making sure no object is enumerated twice
- Still, even simple heuristics can decrease search space drastically!
  - E.g. only (redundantly) enumerate all graphs with certain degree sequences
  - E.g. enumerate rooted trees instead of trees

Estimating size of search space

Let $n, k \in \mathbb{N}$. Define

$$n! := \prod_{i=1}^{n} i$$

$$n^{\underline{k}} := \prod_{i=0}^{k-1} (n-i) = \frac{n!}{(n-k)!}$$

$$\binom{n}{k} := \frac{1}{k!} n^{\underline{k}} = \frac{n!}{k!(n-k)!}$$

$$\left\{ {n \atop k} \right\} := S_{n,k} := \text{number of partitions of } [n] \text{ into } k \text{ (non-empty) classes}$$

("Stirling numbers of 2nd kind")

$$B_n := \text{number of partitions of } [n] \text{ ("Bell numbers")}$$

$$C_n := \frac{1}{n+1} \binom{2n}{n} \text{ ("Catalan numbers")}$$

Estimating size of search space

Let X and Y be sets. Define

$$
\begin{aligned}
X^{\underline{k}} &:= \{(x_1, \ldots, x_k) \in X^k \mid |\{x_1, \ldots, x_k\}| = k\}, \\
\binom{X}{k} &:= \{Y \subseteq X \mid |Y| = k\}, \\
\text{"power set"} \quad \mathfrak{P}(X) &:= \mathcal{P}(X) := 2^X := \{Y \subseteq X\}, \\
Y^X &:= \{f : X \to Y\} \\
\text{"symmetric group"} \quad S_X &:= \{f : X \to X \mid f \text{ bijective}\} \\
\text{"integer partitions"} \quad P_{n,k} &:= \{(x_1, \ldots, x_k) \in \mathbb{N}_+^k \mid x_1 + \cdots + x_k = n, \\
&\qquad\qquad x_1 \leq \cdots \leq x_k\}
\end{aligned}
$$

Estimating size of search space

Let X, Y be (finite) sets. Then

$$|X^{\underline{k}}| = |X|^{\underline{k}}$$
$$\left|\binom{X}{k}\right| = \binom{|X|}{k}$$
$$|2^X| = 2^{|X|}$$
$$|Y^X| = |Y|^{|X|}$$
$$|S_X| = |X|!$$

More combinatorial identities

Here: $0 \in \mathbb{N}$. Let $n, k \in \mathbb{N}$. Then

$$\binom{n+k-1}{k-1} = \left|\{(x_1, \ldots, x_k) \in \mathbb{N}^k \mid x_1 + \cdots + x_k = n\}\right|$$

$$\sum_{k=0}^{n} \left\{ n \atop k \right\} = B_n$$

Moreover, basically every combinatorial coefficient has a recursive formula making its computation feasible.

Identities with Binomial Coefficient

$$\sum_{k=0}^{n} \binom{n}{k} = 2^n$$

$$\sum_{i=k}^{n} \binom{i}{k} = \binom{n+1}{k+1}$$

$$\sum_{k=0}^{n} k \cdot \binom{n}{k} = n \cdot 2^{n-1}$$

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}$$

Identities with Binomial Coefficient

Note: the identity $\sum_{i=k}^{n} \binom{i}{k} = \binom{n+1}{k+1}$ implies:

Corollary

Let R be any ring. For every polynomial $p \in R[x]$, there exists a polynomial $q \in R[x]$ s.t. for all $n \in \mathbb{N}$

$$\sum_{i=0}^{n} p(i) = q(n)$$

Example

- For $p(i) = i$: $q(n) = \frac{n(n+1)}{2}$
- For $p(i) = i^2$: $q(n) = \frac{n(n+1)(2n+1)}{6}$
- For $p(i) = i^3$: $q(n) = \left(\frac{n(n+1)}{2}\right)^2$

Catalan numbers

The Catalan numbers $C_n = \frac{1}{n+1}\binom{2n}{n}$ appear in many applications (see OEIS A000108):

- Number of well-formed words in $\{(,)\}^{2n}$
- Number of ways to evaluate a product of $n+1$ numbers by using the law of associativity. Equivalently: number of syntax tree shapes with n inner binary nodes.
- Number of ways to triangulate a convex polygon with $n+2$ vertices using non-crossing lines

Bounds and Approximations

In practice:

- Exact size not needed to determine whether program terminates in some time frame (also: unknown coefficients!).
- Instead, use bounds and approximations.
- Advantage: much easier to obtain results.

Bounds

In order to bound $\sum_{i \in I} f(i)$, there are two main approaches:

Bounds

In order to bound $\sum_{i \in I} f(i)$, there are two main approaches:

1. Partition I and bound f on each class, i.e. with partition $I = \biguplus_{k=1}^{m} I_k$:

$$\sum_{k=1}^{m} |I_k| \cdot \min_{i \in I_k} f(i) \leq \sum_{i \in I} f(i) \leq \sum_{k=1}^{m} |I_k| \cdot \max_{i \in I_k} f(i)$$

Bounds

In order to bound $\sum_{i \in I} f(i)$, there are two main approaches:

1. Partition I and bound f on each class, i.e. with partition $I = \biguplus_{k=1}^{m} I_k$:

$$\sum_{k=1}^{m} |I_k| \cdot \min_{i \in I_k} f(i) \leq \sum_{i \in I} f(i) \leq \sum_{k=1}^{m} |I_k| \cdot \max_{i \in I_k} f(i)$$

   - Special case $m = 1$: if $c \leq f(i) \leq d$ for each $i \in I$, then $c\,|I| \leq \sum_{i \in I} f(i) \leq d\,|I|$

Bounds

In order to bound $\sum_{i \in I} f(i)$, there are two main approaches:

1. Partition I and bound f on each class, i.e. with partition $I = \biguplus_{k=1}^{m} I_k$:

$$\sum_{k=1}^{m} |I_k| \cdot \min_{i \in I_k} f(i) \leq \sum_{i \in I} f(i) \leq \sum_{k=1}^{m} |I_k| \cdot \max_{i \in I_k} f(i)$$

   - Special case $m = 1$: if $c \leq f(i) \leq d$ for each $i \in I$, then $c\,|I| \leq \sum_{i \in I} f(i) \leq d\,|I|$
   - Special case $m = 2$: if $f(i) \geq 0$ for all $i \in I_1$ and $f(i) \geq c$ for all $i \in I_2$: $c\,|I_2| \leq \sum_{i \in I} f(i)$

Bounds

In order to bound $\sum_{i \in I} f(i)$, there are two main approaches:

1. Partition I and bound f on each class, i.e. with partition $I = \biguplus_{k=1}^{m} I_k$:

$$\sum_{k=1}^{m} |I_k| \cdot \min_{i \in I_k} f(i) \leq \sum_{i \in I} f(i) \leq \sum_{k=1}^{m} |I_k| \cdot \max_{i \in I_k} f(i)$$

   - Special case $m = 1$: if $c \leq f(i) \leq d$ for each $i \in I$, then $c\,|I| \leq \sum_{i \in I} f(i) \leq d\,|I|$
   - Special case $m = 2$: if $f(i) \geq 0$ for all $i \in I_1$ and $f(i) \geq c$ for all $i \in I_2$: $c\,|I_2| \leq \sum_{i \in I} f(i)$

2. For $I = [n]$, if f can be extended to a monotonically increasing function $f : [0, n+1] \to \mathbb{R}$:

Bounds

In order to bound $\sum_{i \in I} f(i)$, there are two main approaches:

1. Partition I and bound f on each class, i.e. with partition $I = \biguplus_{k=1}^{m} I_k$:

$$\sum_{k=1}^{m} |I_k| \cdot \min_{i \in I_k} f(i) \leq \sum_{i \in I} f(i) \leq \sum_{k=1}^{m} |I_k| \cdot \max_{i \in I_k} f(i)$$

   - Special case $m = 1$: if $c \leq f(i) \leq d$ for each $i \in I$, then $c\,|I| \leq \sum_{i \in I} f(i) \leq d\,|I|$
   - Special case $m = 2$: if $f(i) \geq 0$ for all $i \in I_1$ and $f(i) \geq c$ for all $i \in I_2$: $c\,|I_2| \leq \sum_{i \in I} f(i)$

2. For $I = [n]$, if f can be extended to a monotonically increasing function $f : [0, n+1] \to \mathbb{R}$:
   - For each $i \in [n]$, have $\int_{i-1}^{i} f(x)\mathrm{d}x \leq f(i) \leq \int_{i}^{i+1} f(x)\mathrm{d}x$

Bounds

In order to bound $\sum_{i \in I} f(i)$, there are two main approaches:

1. Partition I and bound f on each class, i.e. with partition $I = \biguplus_{k=1}^{m} I_k$:

$$\sum_{k=1}^{m} |I_k| \cdot \min_{i \in I_k} f(i) \leq \sum_{i \in I} f(i) \leq \sum_{k=1}^{m} |I_k| \cdot \max_{i \in I_k} f(i)$$

   - Special case $m = 1$: if $c \leq f(i) \leq d$ for each $i \in I$, then $c\,|I| \leq \sum_{i \in I} f(i) \leq d\,|I|$
   - Special case $m = 2$: if $f(i) \geq 0$ for all $i \in I_1$ and $f(i) \geq c$ for all $i \in I_2$: $c\,|I_2| \leq \sum_{i \in I} f(i)$

2. For $I = [n]$, if f can be extended to a monotonically increasing function $f : [0, n+1] \to \mathbb{R}$:
   - For each $i \in [n]$, have $\int_{i-1}^{i} f(x)\mathrm{d}x \leq f(i) \leq \int_{i}^{i+1} f(x)\mathrm{d}x$
   - Hence:

$$\int_{0}^{n} f(x)\mathrm{d}x \leq \sum_{i=1}^{n} f(i) \leq \int_{1}^{n+1} f(x)\mathrm{d}x$$

Example: Estimating $\sum_{i=1}^{n} i^k$

Example: we want to bound $\sum_{i=1}^{n} i^k$. For simplicity, assume n is even.

Example: Estimating $\sum_{i=1}^{n} i^k$

Example: we want to bound $\sum_{i=1}^{n} i^k$. For simplicity, assume n is even.

1. $i^k \geq \left(\frac{n}{2}\right)^k$ on $[\frac{n}{2}+1, n]$, and on $[n]$, $0 \leq i^k \leq n^k$.

Example: Estimating $\sum_{i=1}^{n} i^k$

Example: we want to bound $\sum_{i=1}^{n} i^k$. For simplicity, assume n is even.

1. $i^k \geq \left(\frac{n}{2}\right)^k$ on $[\frac{n}{2}+1, n]$, and on $[n]$, $0 \leq i^k \leq n^k$. Hence:

$$\left(\frac{n}{2}\right)^{k+1} = \left(\frac{n}{2}\right)^k \cdot \frac{n}{2} \leq \sum_{i=1}^{n} i^k \leq n^k \cdot n = n^{k+1}$$

Example: Estimating $\sum_{i=1}^{n} i^k$

Example: we want to bound $\sum_{i=1}^{n} i^k$. For simplicity, assume n is even.

1. $i^k \geq \left(\frac{n}{2}\right)^k$ on $[\frac{n}{2}+1, n]$, and on $[n]$, $0 \leq i^k \leq n^k$. Hence:

$$\left(\frac{n}{2}\right)^{k+1} = \left(\frac{n}{2}\right)^k \cdot \frac{n}{2} \leq \sum_{i=1}^{n} i^k \leq n^k \cdot n = n^{k+1}$$

2. With the obvious extension to $f : \mathbb{R} \to \mathbb{R}, x \mapsto x^k$:

$$\int_0^n x^k \mathrm{d}x \leq \sum_{i=1}^{n} i^k \leq \int_1^{n+1} x^k \mathrm{d}x$$

Example: Estimating $\sum_{i=1}^{n} i^k$

Example: we want to bound $\sum_{i=1}^{n} i^k$. For simplicity, assume n is even.

1. $i^k \geq \left(\frac{n}{2}\right)^k$ on $[\frac{n}{2}+1, n]$, and on $[n]$, $0 \leq i^k \leq n^k$. Hence:

$$\left(\frac{n}{2}\right)^{k+1} = \left(\frac{n}{2}\right)^k \cdot \frac{n}{2} \leq \sum_{i=1}^{n} i^k \leq n^k \cdot n = n^{k+1}$$

2. With the obvious extension to $f : \mathbb{R} \to \mathbb{R}, x \mapsto x^k$:

$$\int_0^n x^k \mathrm{d}x \leq \sum_{i=1}^{n} i^k \leq \int_1^{n+1} x^k \mathrm{d}x$$

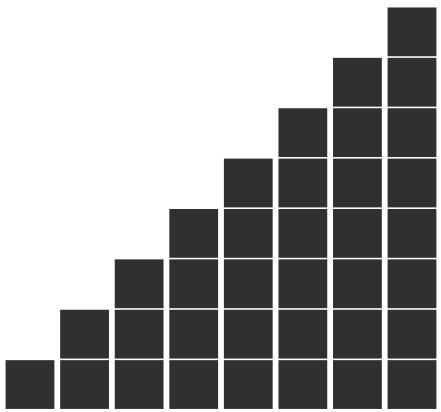
i.e.

$$\frac{1}{k+1} n^{k+1} \leq \sum_{i=1}^{n} i^k \leq \frac{1}{k+1}(n+1)^{k+1} - \frac{1}{k+1} < \frac{1}{k+1}(n+1)^{k+1}$$

Example: Estimating $n!$

Example: we want to estimate $n!$. Since log is monotonic, we can use our tricks on $\log(n!) = \sum_{i=1}^{n} \log(i)$ to obtain bounds for $n!$. Skipping details:

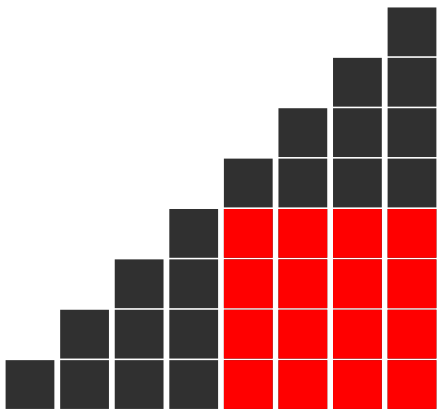
- Partition bound imprecise, but easy to remember:



Example: Estimating $n!$

Example: we want to estimate $n!$. Since log is monotonic, we can use our tricks on $\log(n!) = \sum_{i=1}^{n} \log(i)$ to obtain bounds for $n!$. Skipping details:

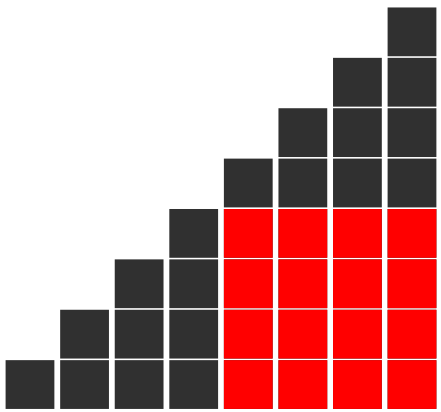
- Partition bound imprecise, but easy to remember:



Example: Estimating $n!$

Example: we want to estimate $n!$. Since log is monotonic, we can use our tricks on $\log(n!) = \sum_{i=1}^{n} \log(i)$ to obtain bounds for $n!$. Skipping details:

- Partition bound imprecise, but easy to remember:

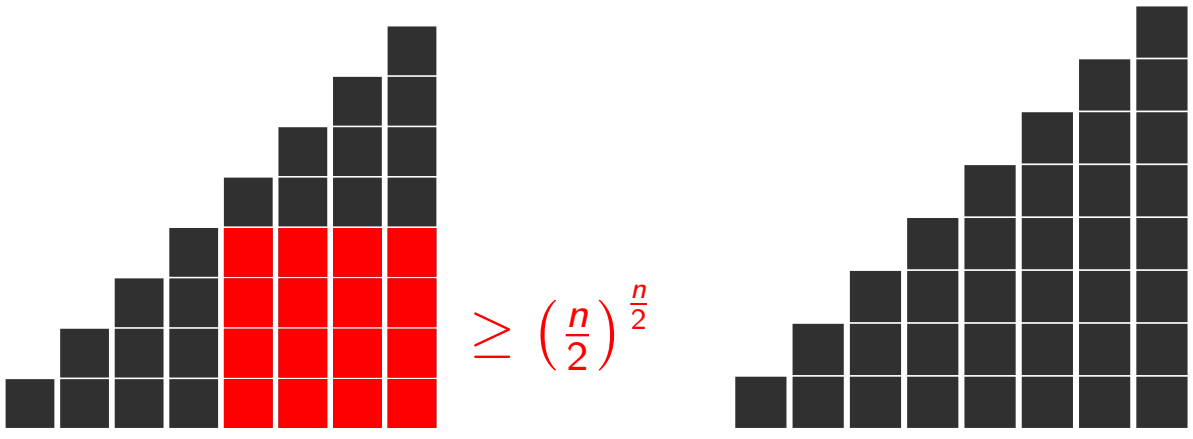


$\geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

Example: Estimating $n!$

Example: we want to estimate $n!$. Since log is monotonic, we can use our tricks on $\log(n!) = \sum_{i=1}^{n} \log(i)$ to obtain bounds for $n!$. Skipping details:

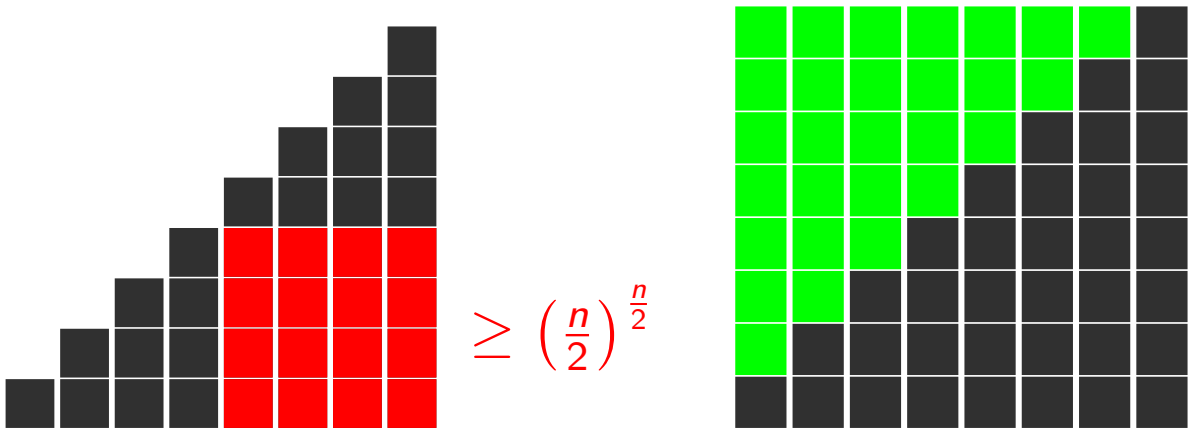
- Partition bound imprecise, but easy to remember:



Example: Estimating $n!$

Example: we want to estimate $n!$. Since log is monotonic, we can use our tricks on $\log(n!) = \sum_{i=1}^{n} \log(i)$ to obtain bounds for $n!$. Skipping details:

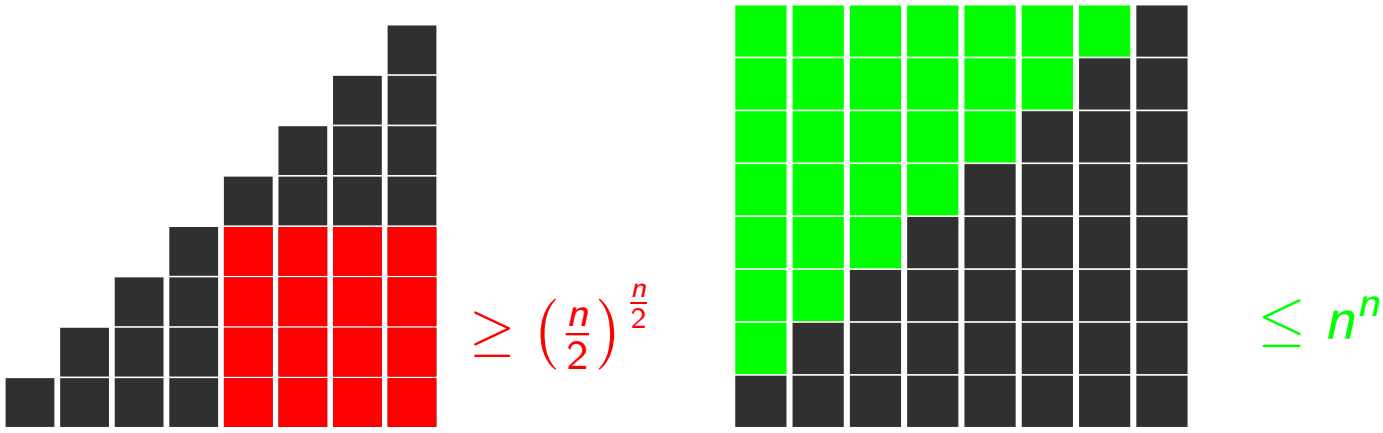
- Partition bound imprecise, but easy to remember:



Example: Estimating $n!$

Example: we want to estimate $n!$. Since log is monotonic, we can use our tricks on $\log(n!) = \sum_{i=1}^{n} \log(i)$ to obtain bounds for $n!$. Skipping details:

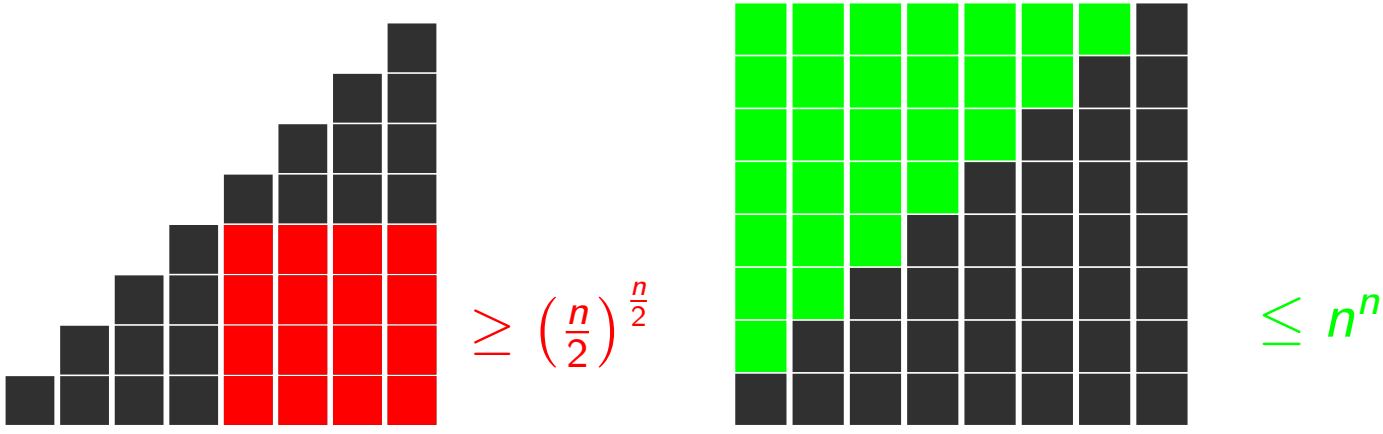
- Partition bound imprecise, but easy to remember:



Example: Estimating $n!$

Example: we want to estimate $n!$. Since log is monotonic, we can use our tricks on $\log(n!) = \sum_{i=1}^{n} \log(i)$ to obtain bounds for $n!$. Skipping details:

- Partition bound imprecise, but easy to remember:



- Integral estimation:

$$e \cdot \left(\frac{n}{e}\right)^n \leq n! \leq \frac{e^2}{4} \cdot \left(\frac{n+1}{e}\right)^{n+1}$$

Injection bounds

Other simple and easy to remember bounds can be obtained be identifying an injection $f : X \to Y$ (which implies $|X| \leq |Y|$):

Injection bounds

Other simple and easy to remember bounds can be obtained be identifying an injection $f : X \to Y$ (which implies $|X| \leq |Y|$):

- Every subset with k elements is a subset: $\binom{n}{k} \leq 2^n$

Injection bounds

Other simple and easy to remember bounds can be obtained be identifying an injection $f : X \to Y$ (which implies $|X| \leq |Y|$):

- Every subset with k elements is a subset: $\binom{n}{k} \leq 2^n$
- For every subset $\emptyset \neq I \subsetneq X$ with $I < X \setminus I$ (where $<$ is some total order on 2^X), $\{I, X \setminus I\}$ is a partition: $\frac{1}{2} \cdot (2^n - 2) \leq S_{n,2} \leq B_n$

Injection bounds

Other simple and easy to remember bounds can be obtained be identifying an injection $f : X \to Y$ (which implies $|X| \leq |Y|$):

- Every subset with k elements is a subset: $\binom{n}{k} \leq 2^n$
- For every subset $\emptyset \neq I \subsetneq X$ with $I < X \setminus I$ (where $<$ is some total order on 2^X), $\{I, X \setminus I\}$ is a partition: $\frac{1}{2} \cdot (2^n - 2) \leq S_{n,2} \leq B_n$
- For every partition P of X, one can totally order each class, and leave elements from different classes incomparable. Hence: $B_n \leq$ number of partial orders on $[n]$

Injection bounds

Other simple and easy to remember bounds can be obtained be identifying an injection $f : X \to Y$ (which implies $|X| \leq |Y|$):

- Every subset with k elements is a subset: $\binom{n}{k} \leq 2^n$
- For every subset $\emptyset \neq I \subsetneq X$ with $I < X \setminus I$ (where $<$ is some total order on 2^X), $\{I, X \setminus I\}$ is a partition: $\frac{1}{2} \cdot (2^n - 2) \leq S_{n,2} \leq B_n$
- For every partition P of X, one can totally order each class, and leave elements from different classes incomparable. Hence: $B_n \leq$ number of partial orders on $[n]$
- Let G_n be the number of undirected graphs with n nodes up to isomorphism. Then, for each vector $(c_1, \ldots, c_k)$ with $c_1 \leq \ldots \leq c_k$ and $\sum_{i=1}^{k} c_i = n$, we get a unique graph by putting k cliques next to each other, where the i-th clique has size c_i. Hence, $G_n \geq p(n) := \sum_{k=1}^{n} |P_{n,k}|$.

Remarks

All combinatorial coefficients have well-known bounds. Sometimes, researching tighter bounds can be useful.

- E.g. Berend, Tassa (2010):

$$\left(\frac{n}{e \log n}\right)^n < B_n < \left(\frac{0.792n}{\log(n+1)}\right)$$

- E.g. Mazumdar, Choudhury (2018):

$$e^{\sqrt{2n}\cdot\zeta(3)} < p(n) < e^{\frac{2n\pi}{\sqrt{6n}}}$$

where $p(n) := \sum_{k=1}^{n} |P_{n,k}|$ and $\zeta(3) = \sum_{n=1}^{\infty} \frac{1}{n^3}$

However, because of their complexity, often simpler bounds or approximations are better.

Approximations

Definition

Let $f, g : \mathbb{N} \to \mathbb{N}$ s.t. they eventually stay positive, i.e. $\exists n_0 : \forall n \geq n_0 : f(n), g(n) > 0$. We define

$$f \sim g :\iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 1,$$

implicitly also requiring that the limit has to exist.

Note that $f \sim g$ is a strictly stronger statement than $f \in \Theta(g)$.

Some approximations

Theorem (Stirling's formula)

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Corollary

$$C_n \sim \frac{4^n}{\sqrt{\pi} \cdot n^{3/2}}$$

Some approximations

Other approximations:

$$\sum_{i=1}^{n} i^k \sim \frac{1}{k+1} n^{k+1}$$

$$p(n) \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$$

$$B_n \sim \frac{1}{\sqrt{n}} \left(\frac{n}{W(n)} \right)^{n+\frac{1}{2}} e^{\frac{n}{W(n)}-n-1}$$

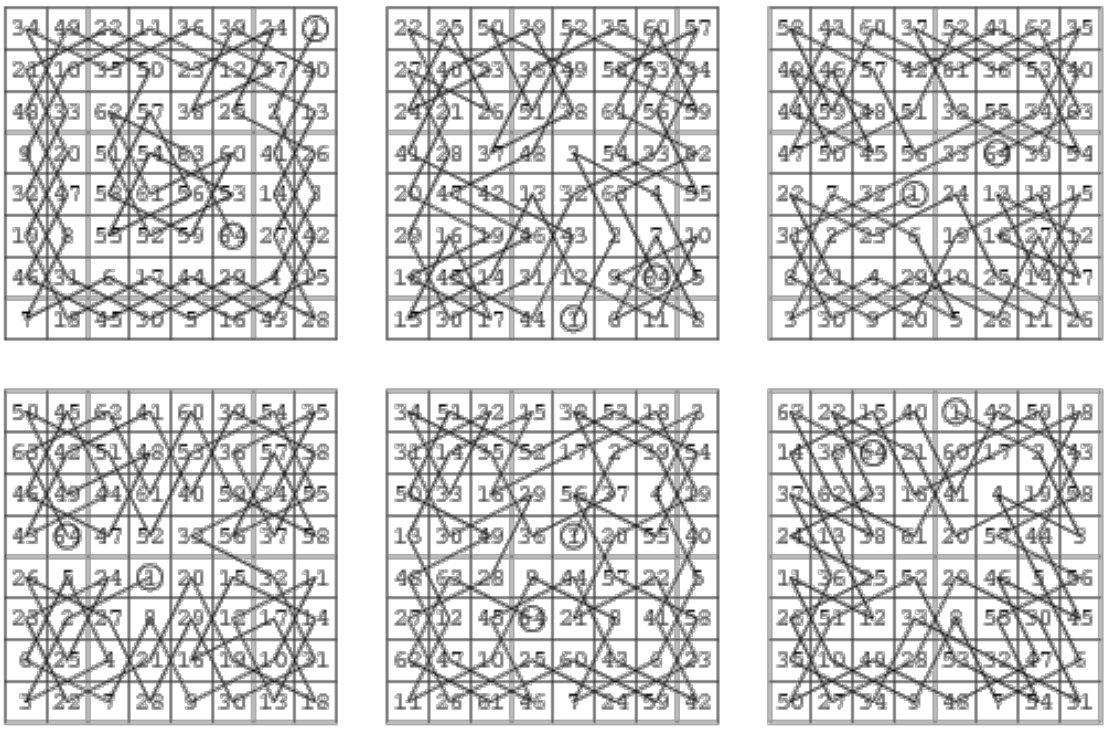
where W is the Lambert W function.

Bounds and Approximations: Final Remarks

Which one to use?

- Try to keep it simple!
- Use bounds whenever possible.
- Use approximations only if they make life easier.

Knight's Tour



Solving Knight's Tour

Naive Solution

Generate all tours (permutations of [64]) and check whether the Knight can travel along such a path.

Solving Knight's Tour

Naive Solution

Generate all tours (permutations of [64]) and check whether the Knight can travel along such a path.

$64! \approx 10^{89}$ — impossible!

What other methods can you think of?

Backtracking

- Applicable when there exist
  - *Partial candidate solutions*
  - *Fast way of semi-checking if the partial candidate cannot be completed*
- Consider search space as a tree
  Internal nodes represent partial solutions
- Dismiss subtree – **prune/backtrack** – if partial solution can't be completed

Example: CNF SAT

Given a boolean formula $\varphi(x_1, \ldots, x_n)$, is there a variable assignment such that φ is satisfied?
We may represent the space of all variable assignments as a tree.

Backtracking Pseudocode

Given a problem which admits partial solutions:

- *valid(s)*: Is partial solution s worth completing?
- *completed(c)*: Is c a complete solution?
- *next(c)*: Set of extensions of c by one step.

Backtracking Pseudocode

Given a problem which admits partial solutions:

- *valid(s)*: Is partial solution *s* worth completing?
- *completed(c)*: Is *c* a complete solution?
- *next(c)*: Set of extensions of *c* by one step.

Algorithm 3 Backtracking

```
function BACKTRACK(c)
    if !valid(c) then
        return false
    if completed(c) then
        output(c)
        return true
    for all c' in next(c) do
        if BACKTRACK(c') then
            return true
```

Constraint Satisfaction Problem

Constraint Satisfaction Problem: Find assignment $\mathcal{X} \to R$ over variables $\mathcal{X}$ such that some constraints $\mathcal{C}$ are satisfied.

Many discrete optimization/search problems can be specified as CSPs.

- SAT
- Puzzles (Crossword, Sudoku, Kakuro, Battleships, ...)
- Graph Coloring
- Combinatorial Optimization (e.g. Knapsack)

Remark: Usually, CSPs are NP-complete.

CSP: Backtracking: Sudoku

Goal: Find $y = (y_1, y_2, \ldots, y_{81})$ in $\{1, \ldots, 9\}^{81}$ satisfying $\mathcal{C}$ = sudoku constraints.

In order to use backtracking, we need $\mathit{valid}(c), \mathit{completed}(c)$ and $\mathit{next}(c)$ where c is a partial solution.

CSP: Backtracking: Sudoku

Goal: Find $y = (y_1, y_2, \ldots, y_{81})$ in $\{1, \ldots, 9\}^{81}$ satisfying $\mathcal{C}$ = sudoku constraints.

In order to use backtracking, we need $\mathit{valid}(c), \mathit{completed}(c)$ and $\mathit{next}(c)$ where c is a partial solution.

Given partial solution, $c = (y_1, y_2, \ldots, y_k), k \leq 81$:

- $\mathit{valid}(c)$: iterate over each row/column/block and check that the partial assignment does not put any number twice in one of them
- $\mathit{completed}(c)$: return "$k = 81$"
- $\mathit{next}(c) = \{(y_1, y_2, \ldots y_k, 1), (y_1, y_2, \ldots y_k, 2), \ldots (y_1, y_2, \ldots y_k, 9)\}$

CSP: Backtracking

More generally: Goal: Find $y = (y_1, \ldots, y_n) \in R^n$ (where R is some finite set) satisfying a set of constraints $\mathcal{C}$.

Assume each constraint is of the form $C : R^n \to \{\text{false}, \text{true}\}$, and can be partially evaluated w.r.t. a partial assignment $c \in R^I$ for $I \subseteq [n]$.

Given partial solution, $c = (y_1, y_2, \ldots, y_k), k \leq n$:

- $\mathit{valid}(c)$: for each $C \in \mathcal{C}$, check whether c already violates C, i.e. whether the partial evaluation $C(c)$ is already trivially false. If so (for any C), return false. Otherwise, return true.
- $\mathit{completed}(c)$: return "$k = n$".
- $\mathit{next}(c) = \{(y_1, y_2, \ldots y_k, v) \mid v \in R\}$

Backtracking: Tips

- The order in which you complete your solution candidates matters.
- The better the order, the more branches of the tree can be cut off.