Lecture 2: Computation and Polynomial Time Complexity

We will cover:

Lecture 2: Computation and Polynomial Time Complexity

We will cover:

- A recap in asymptotics of functions.

Lecture 2: Computation and Polynomial Time Complexity

We will cover:

- A recap in asymptotics of functions.
- Time complexity of a TM, algorithm, and problem.

Lecture 2: Computation and Polynomial Time Complexity

We will cover:

- A recap in asymptotics of functions.
- Time complexity of a TM, algorithm, and problem.
- Polynomial time and its importance.

Lecture 2: Computation and Polynomial Time Complexity

We will cover:

- A recap in asymptotics of functions.
- Time complexity of a TM, algorithm, and problem.
- Polynomial time and its importance.

- **Reading material:** "Computers and Intractability", sec. 1.3.: Polynomial Time Algorithms and Intractable Problems.

In order terms, rank from highest to lowest

- n^{10}
- $\sqrt{n}$
- 2^n
- $n!$
- 2^{n^2}
- $2^{\sqrt{log_2 n}}$
- n

Answer on Mentimeter: .

In order terms, rank from highest to lowest

$$\sqrt{n}$$

In order terms, rank from highest to lowest

$$\sqrt{n} = (n)^{1/2}$$

In order terms, rank from highest to lowest

$$\sqrt{n} = (n)^{1/2} = (2^{\log n})^{1/2}$$

In order terms, rank from highest to lowest

$$\sqrt{n} = (n)^{1/2} = (2^{\log n})^{1/2} = 2^{\frac{\log n}{2}}$$

In order terms, rank from highest to lowest

$$\sqrt{n} = (n)^{1/2} = (2^{\log n})^{1/2} = 2^{\frac{\log n}{2}} \geq 2^{\sqrt{\log_2 n}}.$$

In order terms, rank from highest to lowest

$$2^{\sqrt{\log_2 n}} \leq \sqrt{n} \leq n \leq n^{10}$$

In order terms, rank from highest to lowest

$$2^{\sqrt{\log_2 n}} \leq \sqrt{n} \leq n \leq n^{10} = 2^{10 \log n}$$

In order terms, rank from highest to lowest

$$2^{\sqrt{\log_2 n}} \leq \sqrt{n} \leq n \leq n^{10} \leq 2^n \leq 2^{n^2}$$

In order terms, rank from highest to lowest

$$2^{\sqrt{\log_2 n}} \leq \sqrt{n} \leq n \leq n^{10} \leq 2^n \leq 2^{n^2}$$

Is $n!$ larger or smaller than 2^n ? How about 2^{n^2} ?

In order terms, rank from highest to lowest

$$2^{\sqrt{\log_2 n}} \le \sqrt{n} \le n \le n^{10} \le 2^n \le 2^{n^2}$$

Is $n!$ larger or smaller than 2^n ? How about 2^{n^2} ?
Stirling's approximation:

$$n! = \Theta\left(\sqrt{n}\,(n/e)^n\right)$$

In order terms, rank from highest to lowest

$$2^{\sqrt{\log_2 n}} \leq \sqrt{n} \leq n \leq n^{10} \leq 2^n \leq 2^{n^2}$$

Is $n!$ larger or smaller than 2^n ? How about 2^{n^2} ?
Stirling's approximation:

$$n! = \Theta\left(\sqrt{n}\,(n/e)^n\right) = 2^{n\log n - O(n)} \leq 2^{n^2}$$

In order terms, rank from highest to lowest

$$2^{\sqrt{\log_2 n}} \leq \sqrt{n} \leq n \leq n^{10} \leq 2^n \leq n! \leq 2^{n^2}.$$

Definition: Time Complexity of a TM

Definition: Time Complexity of a TM

- It is now time for us to define the notion of time complexity!

Definition: Time Complexity of a TM

- It is now time for us to define the notion of time complexity!
- Let M be a TM. Let $T : \mathbb{N} \to \mathbb{N}$ be a function such that if input tape has n symbols then M must halt in $\leq T(n)$ steps.

Definition: Time Complexity of a TM

- It is now time for us to define the notion of time complexity!
- Let M be a TM. Let $T : \mathbb{N} \to \mathbb{N}$ be a function such that if input tape has n symbols then M must halt in $\leq T(n)$ steps.
- $T(n)$ is upper bound for time complexity of M.
  If M does not halt on some inputs then time complexity is undefined (or infinite).

Definition: Time Complexity

Definition: Time Complexity

- The time complexity of a problem is the time complexity of the best algorithm that solves it.

Definition: Time Complexity

- The time complexity of a problem is the time complexity of the best algorithm that solves it.
- The time complexity of an algorithm is the time complexity of the best implementation of it.

Definition: Time Complexity

- The time complexity of a problem is the time complexity of the best algorithm that solves it.
- The time complexity of an algorithm is the time complexity of the best implementation of it.
- Time complexity depends on the computation model (e.g., TM, multi-headed TM, RAM machine).

Definition: Time Complexity

- The time complexity of a problem is the time complexity of the best algorithm that solves it.
- The time complexity of an algorithm is the time complexity of the best implementation of it.
- Time complexity depends on the computation model (e.g., TM, multi-headed TM, RAM machine).
- It turns out that the choice of computing model does not make much difference. [1]

[1] That is, most deterministic models are polynomially equivalent.

Polynomials

Example:

$$n^3 - 3n^2 + 7n + 4$$

Polynomials

Example:

$$n^3 - 3n^2 + 7n + 4$$

In general

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0$$

Polynomials

Example:

$$n^3 - 3n^2 + 7n + 4$$

In general

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0$$

But not

$$\sqrt{n}, \quad \frac{1}{n}, \quad 2^n$$

Polynomial Time

Polynomial Time

- A Turing machine runs in polynomial time (p-time) if its complexity is bound by $p(n)$, for some polynomial p.

Polynomial Time

- A Turing machine runs in polynomial time (p-time) if its complexity is bound by $p(n)$, for some polynomial p.
- An algorithm has p-time complexity if it can be implemented by a deterministic TM that runs in p-time.

Polynomial Time

- A Turing machine runs in polynomial time (p-time) if its complexity is bound by $p(n)$, for some polynomial p.
- An algorithm has p-time complexity if it can be implemented by a deterministic TM that runs in p-time.
- A problem is in $\mathbf{P} \subsetneq \Sigma^*$ if it can be solved by a p-time algorithm. These problems are called tractable.

Examples

Polynomial Time

- A Turing machine runs in polynomial time (p-time) if its complexity is bound by $p(n)$, for some polynomial p.
- An algorithm has p-time complexity if it can be implemented by a deterministic TM that runs in p-time.
- A problem is in $\mathbf{P} \subsetneq \Sigma^*$ if it can be solved by a p-time algorithm. These problems are called tractable.

Examples

- Checking if an array has duplicates.

Polynomial Time

- A Turing machine runs in polynomial time (p-time) if its complexity is bound by $p(n)$, for some polynomial p.
- An algorithm has p-time complexity if it can be implemented by a deterministic TM that runs in p-time.
- A problem is in $\mathbf{P} \subsetneq \Sigma^*$ if it can be solved by a p-time algorithm. These problems are called tractable.

Examples

- Checking if an array has duplicates.
- Determining the longest increasing subsequence in an array.

Polynomial Time

- A Turing machine runs in polynomial time (p-time) if its complexity is bound by $p(n)$, for some polynomial p.
- An algorithm has p-time complexity if it can be implemented by a deterministic TM that runs in p-time.
- A problem is in $\mathbf{P} \subsetneq \Sigma^*$ if it can be solved by a p-time algorithm. These problems are called tractable.

Examples

- Checking if an array has duplicates.
- Determining the longest increasing subsequence in an array.
- Finding a minimal spanning tree.

Note

Note that the terminology can get a bit confusing:

Note

Note that the terminology can get a bit confusing:

- TMs that run in times $\log n$, $n \log n$, or $2^{\sqrt{\log n}}$, which are not polynomial, are referred to as polynomial-time!

Note

Note that the terminology can get a bit confusing:

- TMs that run in times $\log n$, $n \log n$, or $2^{\sqrt{\log n}}$, which are not polynomial, are referred to as polynomial-time!
- Some problems are outside **P**, in that they have a super-polynomial time complexity of, e.g., $2^{(\log n)^2}$.

Super

Super

$\rightarrow$

- Super-polynomial?

Sub- and Super

- A super-polynomial function $f(n)$ is one that satisfies $f(n) = n^{\omega(1)}$.

Sub- and Super

- A super-polynomial function $f(n)$ is one that satisfies $f(n) = n^{\omega(1)}$.
- A sub-polynomial function $f(n)$ satisfies $f(n) = n^{o(1)}$, e.g., $f(n) = 2^{\sqrt{\log n}}$.

Sub- and Super

- A super-polynomial function $f(n)$ is one that satisfies $f(n) = n^{\omega(1)}$.
- A sub-polynomial function $f(n)$ satisfies $f(n) = n^{o(1)}$, e.g., $f(n) = 2^{\sqrt{\log n}}$.
- Can be used for other time classes, e.g., super-linear or sub-quadratic.

Subtleties

- Exceptions to difference between efficient polynomial time and inefficient exponential time, e.g. 2^n is faster than n^{100} for

Subtleties

- Exceptions to difference between efficient polynomial time and inefficient exponential time, e.g. 2^n is faster than n^{100} for $n \leq 996$.

Subtleties

- Exceptions to difference between efficient polynomial time and inefficient exponential time, e.g. 2^n is faster than n^{100} for $n \leq 996$.
- Some exponential time algorithms are quite useful in practice!

Subtleties

- Exceptions to difference between efficient polynomial time and inefficient exponential time, e.g. 2^n is faster than n^{100} for $n \leq 996$.
- Some exponential time algorithms are quite useful in practice! An example is the Simplex algorithm of linear programming. This algorithm has an exp time complexity, but in practice runs quite quickly.

Subtleties

- Exceptions to difference between efficient polynomial time and inefficient exponential time, e.g. 2^n is faster than n^{100} for $n \leq 996$.
- Some exponential time algorithms are quite useful in practice! An example is the Simplex algorithm of linear programming. This algorithm has an exp time complexity, but in practice runs quite quickly.
- Some problems are provably intractable and the first one of these is Turing's halting problem.

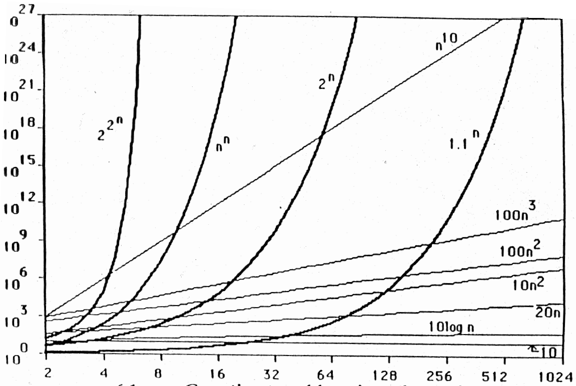
Quote of the day

Quote of the day

"The greatest shortcoming of the human race is our inability to understand the exponential function."

Prof. Albert Bartlett, 1969.

Illustration



6.1 Growth rates of functions (log scales)

Faster computers

3 computers (or TMs) C_1 (ordinary) , C_2 (2^{12} times as fast as C_1), and C_3 (10^{12} times as fast as C_1).

Faster computers

3 computers (or TMs) C_1 (ordinary) , C_2 (2^{12} times as fast as C_1), and C_3 (10^{12} times as fast as C_1).

Run time	Max. n in t seconds		
	C_1	C_2	C_3
$O(n)$	α_0		
$O(n^2)$	α_1		
$O(n^3)$	α_2		
$O(2^n)$	α_3		
$O(2^{2^n})$	α_4		

Faster computers

3 computers (or TMs) C_1 (ordinary) , C_2 (2^{12} times as fast as C_1), and C_3 (10^{12} times as fast as C_1).

Run time	Max. n in t seconds		
	C_1	C_2	C_3
$O(n)$	α_0	$\alpha_0 \times 2^{12}$	$\alpha_0 \times 10^{12}$
$O(n^2)$	α_1		
$O(n^3)$	α_2		
$O(2^n)$	α_3		
$O(2^{(2^n)})$	α_4		

Faster computers

3 computers (or TMs) C_1 (ordinary) , C_2 (2^{12} times as fast as C_1), and C_3 (10^{12} times as fast as C_1).

Run time	Max. n in t seconds		
	C_1	C_2	C_3
$O(n)$	α_0	$\alpha_0 \times 2^{12}$	$\alpha_0 \times 10^{12}$
$O(n^2)$	α_1	$\alpha_1 \times 2^6$	$\alpha_1 \times 10^6$
$O(n^3)$	α_2		
$O(2^n)$	α_3		
$O(2^{(2^n)})$	α_4		

Faster computers

3 computers (or TMs) C_1 (ordinary) , C_2 (2^{12} times as fast as C_1), and C_3 (10^{12} times as fast as C_1).

Run time	Max. n in t seconds		
	C_1	C_2	C_3
$O(n)$	α_0	$\alpha_0 \times 2^{12}$	$\alpha_0 \times 10^{12}$
$O(n^2)$	α_1	$\alpha_1 \times 2^6$	$\alpha_1 \times 10^6$
$O(n^3)$	α_2	$\alpha_2 \times 2^4$	$\alpha_2 \times 10^4$
$O(2^n)$	α_3		
$O(2^{(2^n)})$	α_4		

Faster computers

3 computers (or TMs) C_1 (ordinary) , C_2 (2^{12} times as fast as C_1), and C_3 (10^{12} times as fast as C_1).

Run time	Max. n in t seconds		
	C_1	C_2	C_3
$O(n)$	α_0	$\alpha_0 \times 2^{12}$	$\alpha_0 \times 10^{12}$
$O(n^2)$	α_1	$\alpha_1 \times 2^6$	$\alpha_1 \times 10^6$
$O(n^3)$	α_2	$\alpha_2 \times 2^4$	$\alpha_2 \times 10^4$
$O(2^n)$	α_3	$\alpha_3 + 12$	$\alpha_3 + 40$
$O(2^{(2^n)})$	α_4		

Faster computers

3 computers (or TMs) C_1 (ordinary) , C_2 (2^{12} times as fast as C_1), and C_3 (10^{12} times as fast as C_1).

Run time	Max. n in t seconds		
	C_1	C_2	C_3
$O(n)$	α_0	$\alpha_0 \times 2^{12}$	$\alpha_0 \times 10^{12}$
$O(n^2)$	α_1	$\alpha_1 \times 2^6$	$\alpha_1 \times 10^6$
$O(n^3)$	α_2	$\alpha_2 \times 2^4$	$\alpha_2 \times 10^4$
$O(2^n)$	α_3	$\alpha_3 + 12$	$\alpha_3 + 40$
$O(2^{(2^n)})$	α_4	α_4	α_4

The input size, n

- Consider the problem of determining whether $k \in \mathbb{N}$ is prime.

The input size, n

- Consider the problem of determining whether $k \in \mathbb{N}$ is prime.
- Here is one possible algorithm:

The input size, n

- Consider the problem of determining whether $k \in \mathbb{N}$ is prime.
- Here is one possible algorithm:

  $i \leftarrow 2$
  While $(i \leq \sqrt{k})$
    If (i divides k)
     Return False
    $i \leftarrow i + 1$
  Return True

The input size, n

- Consider the problem of determining whether $k \in \mathbb{N}$ is prime.
- Here is one possible algorithm:

  $i \leftarrow 2$
  While $(i \leq \sqrt{k})$
  If (i divides k)
  Return False
  $i \leftarrow i + 1$
  Return True
- The runtime is $O(\sqrt{k})$ in the worst case.[2]

The input size, n

- Consider the problem of determining whether $k \in \mathbb{N}$ is prime.
- Here is one possible algorithm:

  $i \leftarrow 2$
  While $(i \leq \sqrt{k})$
          If (i divides k)
              Return False
          $i \leftarrow i + 1$
  Return True

- The runtime is $O(\sqrt{k})$ in the worst case.[2]
- But we only need $n \approx \log_2 k$ bits to represent k!

The input size, n

- Consider the problem of determining whether $k \in \mathbb{N}$ is prime.
- Here is one possible algorithm:

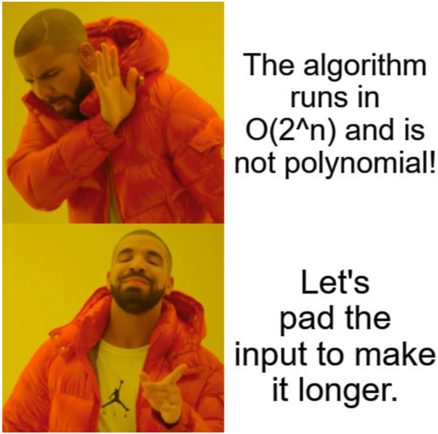
  $i \leftarrow 2$
  While $(i \leq \sqrt{k})$
      If (i divides k)
        Return False
      $i \leftarrow i + 1$
  Return True
- The runtime is $O(\sqrt{k})$ in the worst case.[2]
- But we only need $n \approx \log_2 k$ bits to represent k!
- The complexity is $O(\sqrt{2^n})$ which is not polynomial.

- Consider the problem of determining whether $k \in \mathbb{N}$ is prime.
- Here is one possible algorithm:

```
i ← 2
While (i ≤ √k)
        If (i divides k)
            Return False
        i ← i + 1
Return True
```

- The runtime is $O(\sqrt{k})$ in the worst case.[2]
- But we only need $n \approx \log_2 k$ bits to represent k!
- The complexity is $O(\sqrt{2^n})$ which is not polynomial.

[2] The complexity is correct for a RAM machine.


The algorithm runs in O(2^n) and is not polynomial!
Let's pad the input to make it longer.

Problems vs. Languages

- Formally, **P** is a set of languages $\mathbf{P} \subsetneq \Sigma^*$.

Problems vs. Languages

- Formally, **P** is a set of <ins>languages</ins> $\mathbf{P} \subsetneq \Sigma^*$.
- When we say that a <ins>problem</ins> is in **P**, we must agree on the encoding of its inputs.

Problems vs. Languages

- Formally, **P** is a set of languages $\mathbf{P} \subsetneq \Sigma^*$.
- When we say that a problem is in **P**, we must agree on the encoding of its inputs.
- For example, $L_1 = \{1^k 0 \mid k \text{ is prime}\}$ is polynomial-time recognizable by the last algorithm while $L_2 = \{k \in \{0,1\}^* \mid k \text{ is prime}\}$ is not.

Problems vs. Languages

- Formally, **P** is a set of languages **P** $\subsetneq \Sigma^*$.
- When we say that a problem is in **P**, we must agree on the encoding of its inputs.
- For example, $L_1 = \{1^k 0 \mid k \text{ is prime}\}$ is polynomial-time recognizable by the last algorithm while $L_2 = \{k \in \{0,1\}^* \mid k \text{ is prime}\}$ is not.
- Most often, we will ask if the most succinct representation of the inputs allow polynomial-time recognizability.

Which ones you think are solvable in polynomial time?

- Is n prime?

Which ones you think are solvable in polynomial time?

- Is n prime?
- Is graph G connected?

Which ones you think are solvable in polynomial time?

- Is n prime?
- Is graph G connected?
- Find shortest path from x to y in weighted graph G

Which ones you think are solvable in polynomial time?

- Is n prime?
- Is graph G connected?
- Find shortest path from x to y in weighted graph G
- Given regexp r and string s, is $s \in L(r)$?

Which ones you think are solvable in polynomial time?

- Is n prime?
- Is graph G connected?
- Find shortest path from x to y in weighted graph G
- Given regexp r and string s, is $s \in L(r)$?
- Given regexps r, s is $L(r) = L(s)$?

Which ones you think are solvable in polynomial time?

- Is n prime?
- Is graph G connected?
- Find shortest path from x to y in weighted graph G
- Given regexp r and string s, is $s \in L(r)$?
- Given regexps r, s is $L(r) = L(s)$?
- Travelling Salesman Problem

Which ones you think are solvable in polynomial time?

- Is n prime?
- Is graph G connected?
- Find shortest path from x to y in weighted graph G
- Given regexp r and string s, is $s \in L(r)$?
- Given regexps r, s is $L(r) = L(s)$?
- Travelling Salesman Problem
- Does the program P halt?

Which ones you think are solvable in polynomial time?

- Is n prime?
- Is graph G connected?
- Find shortest path from x to y in weighted graph G
- Given regexp r and string s, is $s \in L(r)$?
- Given regexps r, s is $L(r) = L(s)$?
- Travelling Salesman Problem
- Does the program P halt?
- From position P, which player has a winning strategy in generalised chess?

Answer on Mentimeter: .

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]
- Is graph G connected? Yes, BFS

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]
- Is graph G connected? Yes, BFS
- Find shortest path from x to y in weighted graph G Yes, Dijkstra's

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]
- Is graph G connected? Yes, BFS
- Find shortest path from x to y in weighted graph G Yes, Dijkstra's
- Given regexp r and string s, is $s \in L(r)$? Yes

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]
- Is graph G connected? Yes, BFS
- Find shortest path from x to y in weighted graph G Yes, Dijkstra's
- Given regexp r and string s, is $s \in L(r)$? Yes
- Given regexps r, s is $L(r) = L(s)$? Probably not (PSPACE-complete)

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]
- Is graph G connected? Yes, BFS
- Find shortest path from x to y in weighted graph G Yes, Dijkstra's
- Given regexp r and string s, is $s \in L(r)$? Yes
- Given regexps r, s is $L(r) = L(s)$? Probably not (PSPACE-complete)
- Travelling Salesman Problem Probably not (NP-complete)

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]
- Is graph G connected? Yes, BFS
- Find shortest path from x to y in weighted graph G Yes, Dijkstra's
- Given regexp r and string s, is $s \in L(r)$? Yes
- Given regexps r, s is $L(r) = L(s)$? Probably not (PSPACE-complete)
- Travelling Salesman Problem Probably not (NP-complete)
- Does the program P halt? No (undecidable)

Which ones you think are solvable in polynomial time?

- Is n prime? Yes, [AKS 2002]
- Is graph G connected? Yes, BFS
- Find shortest path from x to y in weighted graph G Yes, Dijkstra's
- Given regexp r and string s, is $s \in L(r)$? Yes
- Given regexps r, s is $L(r) = L(s)$? Probably not (PSPACE-complete)
- Travelling Salesman Problem Probably not (NP-complete)
- Does the program P halt? No (undecidable)
- From position P, which player has a winning strategy in generalised chess? Probably not (PSPACE-complete)

Summary

Summary

- We recalled useful asymptotic notations.

Summary

- We recalled useful asymptotic notations.
- Defined the time complexity of a TM, algorithm, and problem.

Summary

- We recalled useful asymptotic notations.
- Defined the time complexity of a TM, algorithm, and problem.
- Discussed what is polynomial time and its importance.

Summary

- We recalled useful asymptotic notations.
- Defined the time complexity of a TM, algorithm, and problem.
- Discussed what is polynomial time and its importance.
- Pondered about the affect the input representation has on the notion of polynomiality.