Introduction to algorithms, Big O notation

László Papp

BME

2023. 03. 03.

Course information

- Lecture: Room E302 at 12:15-13:45 on Fridays, all weeks
- Practice lecture: Room IB138 at 17:15-18:45 on Tuesdays, only on even weeks
- All official information will be announced on the Homepage, its location is `www.cs.bme.hu/~lazsa/combopt2023/combopt2023.html` Please follow it.
- Slides will be uploaded to the homepage.
- There will be no slides for practice sessions.

Contact

- You can send me a message on Teams.
- My email address: lazsa88@gmail.com
- You can meet me at my office IE217.3 at 10:00-11:30 on Fridays.
- You can ask me after the lectures/practice sessions

Requirements

There will be two midterms during the semester:

- 19th of April, 18:00-20:00 Location: TBA
- 18th of May, 18:00-20:00 ?? Location: TBA
- To complete the course you have to receive **at least 40% of the points at each midterm**.
- If you fail them or want to improve your grade, there will be two occasions at the end of the semester where you can retake a midterm.
- More details (location for example) will be announced later in homepage.

What is an optimization problem?

A set **A** and an objective function $f : \mathbf{A} \to \mathbb{R}$ is given. **A** contains the possible solutions of a problem and for any $x \in \mathbf{A}$ $f(x)$ is a real number.
We are looking for an element y in **A** which satisfies that $f(y)$ is the smallest/biggest possible.

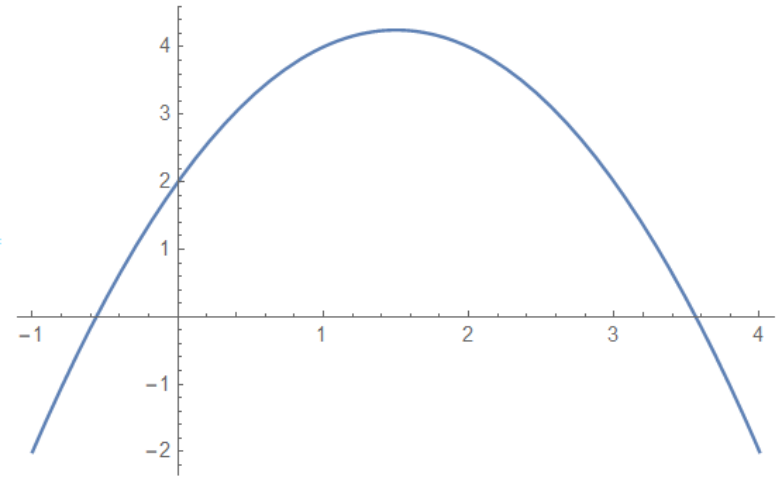
Example: $\mathbf{A} = [-1, 4]$, $f(x) = 2 - x^2 + 3x$. Find an element of A where f attains its maximum value (if such an element exists)!

What is an optimization problem?

A set **A** and an objective function $f : \mathbf{A} \to \mathbb{R}$ is given. **A** contains the possible solutions of a problem and for any $x \in \mathbf{A}$ $f(x)$ is a real number.
We are looking for an element y in **A** which satisfies that $f(y)$ is the smallest/biggest possible.

Example: $\mathbf{A} = [-1, 4]$, $f(x) = 2 - x^2 + 3x$. Find an element of A where f attains its maximum value (if such an element exists)!

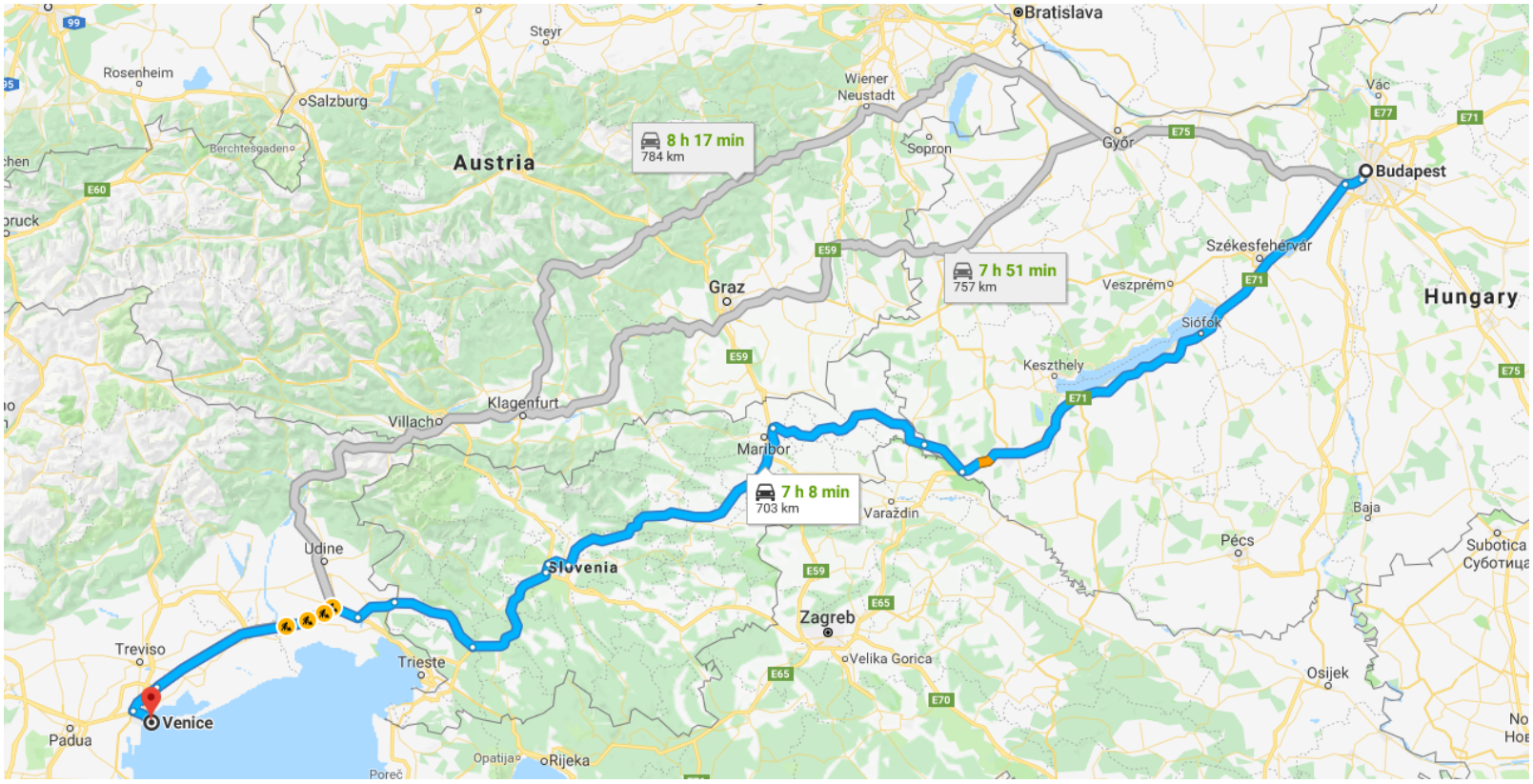
We can find such an element by using calculus:
$f'(x) = -2x + 3$, $f'(\frac{3}{2}) = 0$ and $f''(x) = -2$. Therefore $\frac{3}{2}$ is a maximum point and the maximum value is $f(\frac{3}{2}) = \frac{17}{4}$.



What is combinatorial optimization?

Now the set of possible solutions **A** is finite and contains combinatorial objects. We are looking for a solution *x* which maximizes (or minimizes) the objective function.

Example:



Find the fastest route from Budapest to Venice!

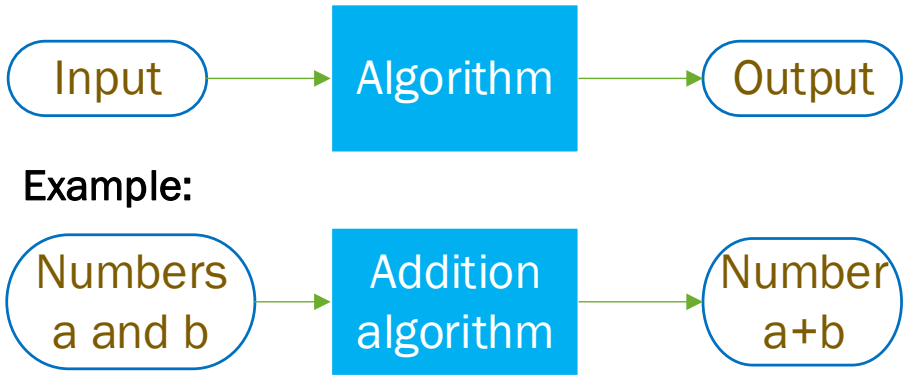
What is the difference between continuous and combinatorial (discrete) optimization?

Optimization	Continuous	Combinatorial
Search space		
Number of elements	infinit	finite
Topology	continuous	discrete
Tools to solve		
Calculus	important	less useful
Linear algebra	useful	useful
Algorithmic theory	useful	very important
Graph theory	useless	important

During the semester we are going to focus on three mayor areas: Algorithmic theory, Graph theory and Linear algebra.

What is an algorithm?

We do not give a formal definition for what an algorithm is. We usually say the following:
An algorithm is a method which can be implemented as a computer program and can be executed on a computer.



An algorithm usually has an input and an output. The algorithm receives its input, works with it, then it gives us its output.

Example: An algorithm for addition

Input: Integers a and b.
The algorithm is what we have learnt in elementary school:

We add multi digit numbers by digits starting from the ones column at the right and if the sum of the two digits is bigger than one then we carry the “extra” digit to the next column.

$$\begin{array}{r} 147 \\ +105 \\ \hline 252 \end{array} \qquad\qquad \begin{array}{r} 10010011 \\ +1101001 \\ \hline 11111100 \end{array}$$

There are several algorithms for addition, but most of us have learnt this one. Why?
Because it is simple and “fast”, but what does “ fast” mean?

How to measure the efficiency of an algorithm?

Problem: We have a problem and we found two or more algorithms which solve it. Which one should we use?

How to measure the efficiency of an algorithm?

Problem: We have a problem and we found two or more algorithms which solve it. Which one should we use?

There are several metrics which measure how good an algorithm is. For example: How much memory does it need, how many processor cores does it utilize, how much time does a computer need to execute it on a specified input, etc.

How to measure the efficiency of an algorithm?

Problem: We have a problem and we found two or more algorithms which solve it. Which one should we use?

There are several metrics which measure how good an algorithm is. For example: How much memory does it need, how many processor cores does it utilize, how much time does a computer need to execute it on a specified input, etc.

In this course we are focusing only at the running time of an algorithm. The running time of an algorithm depends on several things, for example the type of the computer which we use. To get rid of the differences between computers, instead of measuring time, we count the number of elementary operations which the algorithm makes before it gives back its output.

The obtained number still depends on the input. For example multiplying big numbers requires more time than multiplying small ones.

The size of the input

Definition

Fix an alphabet. The **size of the input** over this alphabet is the number of symbols (contained in the alphabet) needed to encode the input.

Example

The lenght of an integer:

- Input: An integer a
- Alphabet: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$ (we use the decimal number system now)
- If the input is 168, then its size is 3.
- If the input is an integer a, then its size is $\lceil \log_{10}(a+1) \rceil$.

The size of the input

Definition

Fix an alphabet. The **size of the input** over this alphabet is the number of symbols (contained in the alphabet) needed to encode the input.

Example

The lenght of an integer:

- Input: An integer a
- Alphabet: $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$ (we use the decimal number system now)
- If the input is 168, then its size is 3.
- If the input is an integer a, then its size is $\lceil \log_{10}(a+1) \rceil$.

In the computer's memory everything is a binary number.
Therefore the input is encoded as a binary number.
If the input is a natural number a, then its size is $\lceil \log_2(a+1) \rceil$.

We will write $\log_2(a)$ in the later slides for simplicity.

The size of the input

Question: What is the size of 168 if we encode it as a binary number?

The size of the input

Question: What is the size of 168 if we encode it as a binary number?
Answer: $\lceil \log_2(169) \rceil = 8$. 168 in binary form is 10101000 and

the number of digits is 8.

Remark: The logarithmic identity $\log_k(x) = \frac{\log_2(x)}{\log_2(k)}$ guarantees that using binary encoding instead of an alphabet containing k symbols results in an $\log_2 k$ increase of the size of the input. For example the size of a number in the binary number system is approximately $\log_2 10 \approx 3.2$ times its size in the decimal system.
Therefore the size of the alphabet is not important.

From now, $\log(n)$ denotes the base 2 logarithm of n.

Estimating the running time

The running time depends on the size of the input, but it also depensd on the structure of the input.
For example calculating $10000 \cdot 1$ is much easier than calculating $432 \cdot 167$, but the sizes of these inputs are the same.

Estimating the running time

The running time depends on the size of the input, but it also depensd on the structure of the input.
For example calculating $10000 \cdot 1$ is much easier than calculating $432 \cdot 167$, but the sizes of these inputs are the same.
For safety reasons, (think about mission critical applications like airplanes, self-driving cars, powerplants, etc.) we are generally interested in the worst case scenario.

Estimating the running time

The running time depends on the size of the input, but it also depensd on the structure of the input.
For example calculating $10000 \cdot 1$ is much easier than calculating $432 \cdot 167$, but the sizes of these inputs are the same.
For safety reasons, (think about mission critical applications like airplanes, self-driving cars, powerplants, etc.) we are generally interested in the worst case scenario.

Definition

The **time complexity of an algorithm** A is an integer-valued function f whose value at n tells us the maximum number of steps (elementary operations) which need to be executed on any input of size n.

Estimating the running time

The running time depends on the size of the input, but it also depensd on the structure of the input.
For example calculating $10000 \cdot 1$ is much easier than calculating $432 \cdot 167$, but the sizes of these inputs are the same.
For safety reasons, (think about mission critical applications like airplanes, self-driving cars, powerplants, etc.) we are generally interested in the worst case scenario.

Definition

The **time complexity of an algorithm** A is an integer-valued function f whose value at n tells us the maximum number of steps (elementary operations) which need to be executed on any input of size n.

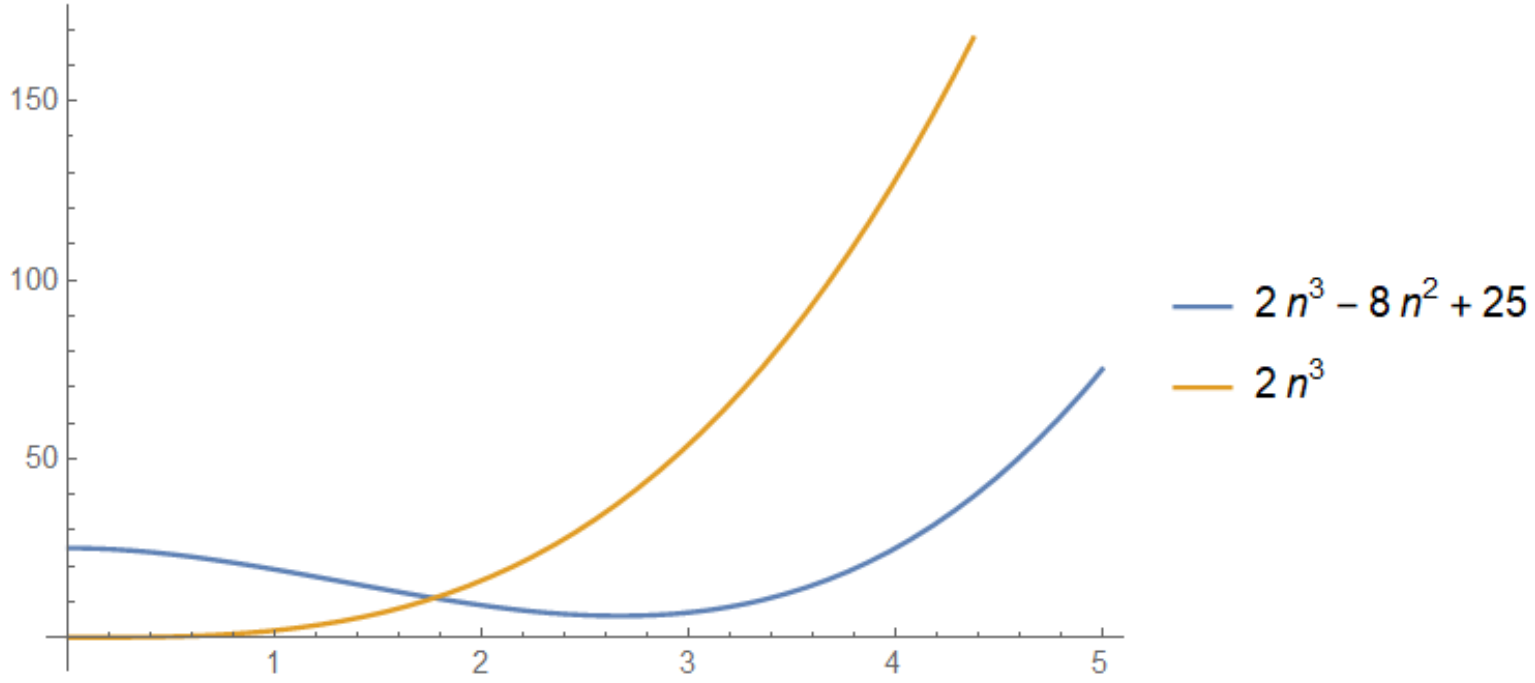
Usually it is hard to calculate the time complexity of an algorithm, but a close upper bound is good enough for us. For example if the time complexity is $3n^2 - 2n + 1$ then $3n^2$ is an upper bound and it is much easier to work with.

The big O notation

Definition

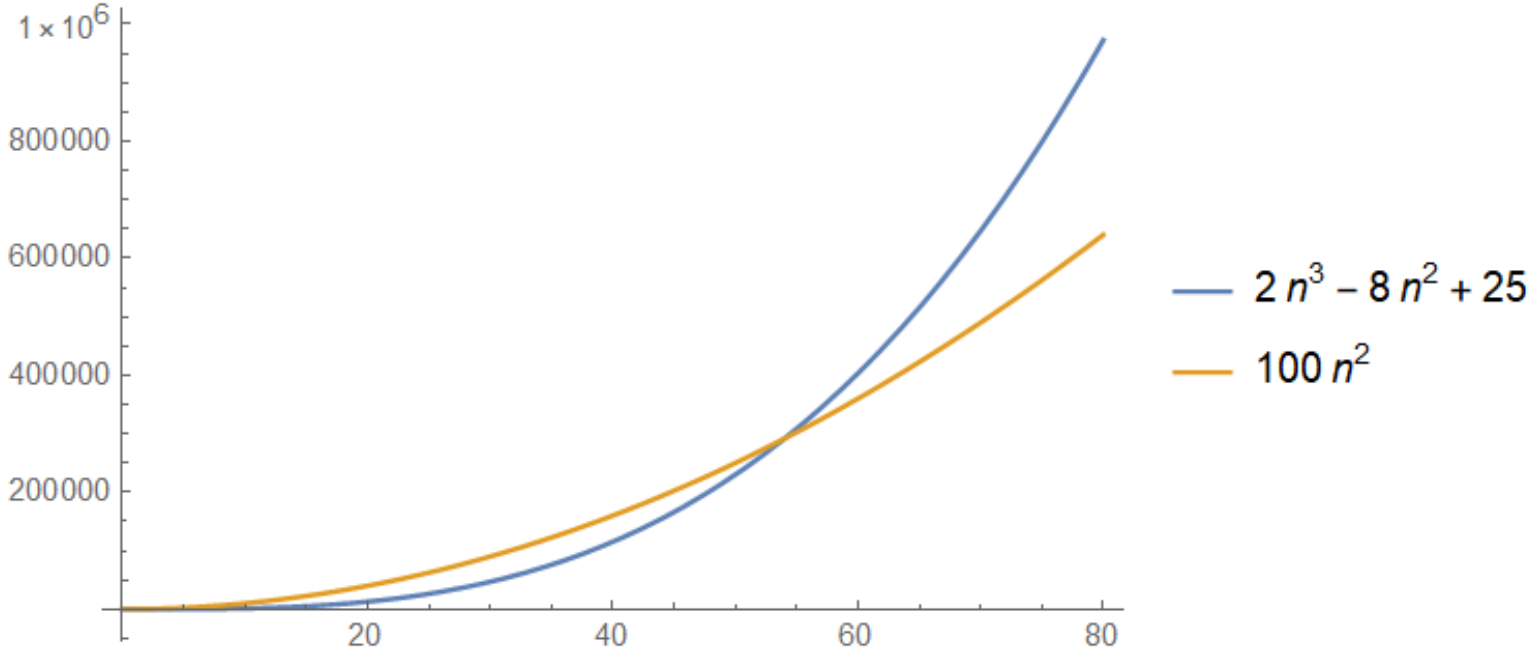
Let $f(n)$ and $g(n)$ be real functions. Then $f(n) \in O(g(n))$ means that there exists a natural number N and a positive constant c, such that for every $n \geq N$, $|f(n)| \leq cg(n)$.

Example: $2n^3 - 8n^2 + 25 \in O(n^3)$ because $|2n^3 - 8n^2 + 25| \leq 2n^3$ for all $n > 2 = N$. **This requires some reasoning!**



The meaning of the big O notation

- $2n^3 - 8n^2 + 25 \in O(n^3)$ means that after a while (the absolute value of) $2n^3 - 8n^2 + 25$ does not grow faster than a constant multiple of n^3.
- $2n^3 - 8n^2 + 25 \notin O(n^2)$ means that $2n^3 - 8n^2 + 25$ grows faster than any quadratic function.



- If you are good at calculus, then you can think about the following equivalent definition: $f(n) \in O(g(n))$ if and only if $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$.

Using the big O notation for time complexities

The time complexity of an algorithm is function whose domain and codomain are both nonnegative. (The lenght of the input is nonnegative and a computer cannot make -2 steps.) So if we talk about time complexities we can ommit the absolut vale from the definition of the big O notation.

Definition (Big O for time complexities)

Let $f(n)$ be an $\mathbb{R}^+ \to \mathbb{R}^+ \cup \{0\}$ (nonnegative) funcion and $g(n)$ be a real function. Then $f(n) \in O(g(n))$ means that there exists a natural number N and a positive constant c, such that for every $n \geq N$, $f(n) \leq cg(n)$.

Since in this course we are talking about algorithms we will use this defintion for big O from now.

Examples for O notation

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^3)$?

Examples for O notation

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^3)$?
Answer: Yes, because there is a pair of c, N which satisfies the definition:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3$ if $n \geq 2$, so $f(n) \leq cn^3$ for all $n \geq N$ if $c = 5$, $N = 2$.
Note that the pair $c = 6$, $N = 3$ is also good.

Examples for O notation

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^3)$?
Answer: Yes, because there is a pair of c, N which satisfies the definition:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3$ if $n \geq 2$, so $f(n) \leq cn^3$ for all $n \geq N$ if $c = 5$, $N = 2$.
Note that the pair $c = 6$, $N = 3$ is also good.

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^4)$?

Examples for O notation

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^3)$?
Answer: Yes, because there is a pair of c, N which satisfies the definition:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3$ if $n \geq 2$, so $f(n) \leq cn^3$ for all $n \geq N$ if $c = 5$, $N = 2$.
Note that the pair $c = 6$, $N = 3$ is also good.

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^4)$?
Answer: Yes, we can verify the definition again:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3 \leq 5n^4$ if $n \geq 2$, so $f(n) \leq cn^4$ for all $n \geq B$ if $c = 5$, $N = 2$.

Examples for O notation

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^3)$?
Answer: Yes, because there is a pair of c, N which satisfies the definition:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3$ if $n \geq 2$, so $f(n) \leq cn^3$ for all $n \geq N$ if $c = 5$, $N = 2$.
Note that the pair $c = 6$, $N = 3$ is also good.

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^4)$?
Answer: Yes, we can verify the definition again:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3 \leq 5n^4$ if $n \geq 2$, so $f(n) \leq cn^4$ for all $n \geq B$ if $c = 5$, $N = 2$.

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^2)$?

Examples for O notation

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^3)$?
Answer: Yes, because there is a pair of c, N which satisfies the definition:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3$ if $n \geq 2$, so $f(n) \leq cn^3$ for all $n \geq N$ if $c = 5$, $N = 2$.
Note that the pair $c = 6$, $N = 3$ is also good.

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^4)$?
Answer: Yes, we can verify the definition again:
$3n^3 + 2n\log(n) \leq 3n^3 + 2n^2 \leq 5n^3 \leq 5n^4$ if $n \geq 2$, so $f(n) \leq cn^4$ for all $n \geq B$ if $c = 5$, $N = 2$.

Question: Is $f(n) = 3n^3 + 2n\log(n) \in O(n^2)$?
Answer: No.
Homework: Prove it!

Proof that $n^3 \notin O(n^2)$:

We prove it by contradiction.

Assume the contrary, so suppose that $n^3 \in O(n^2)$.
According to the definition of big O notation, there is a natural number N and a positive constant c such that $n^3 \leq cn^2$ for all $n \geq N$.

Proof that $n^3 \notin O(n^2)$:

We prove it by contradiction.

Assume the contrary, so suppose that $n^3 \in O(n^2)$.
According to the definition of big O notation, there is a natural number N and a positive constant c such that $n^3 \leq cn^2$ for all $n \geq N$.

After dividing by n^2 we get that $n \leq c$ for all $n \geq N$.

Proof that $n^3 \notin O(n^2)$:

We prove it by contradiction.

Assume the contrary, so suppose that $n^3 \in O(n^2)$.
According to the definition of big O notation, there is a natural number N and a positive constant c such that $n^3 \leq cn^2$ for all $n \geq N$.

After dividing by n^2 we get that $n \leq c$ for all $n \geq N$.

On the other hand, we know that $\lim_{n\to\infty} n = \infty$, so the function $f(n) = n$ can not be bounded by a positive constant.

Proof that $n^3 \notin O(n^2)$:

We prove it by contradiction.

Assume the contrary, so suppose that $n^3 \in O(n^2)$.
According to the definition of big O notation, there is a natural number N and a positive constant c such that $n^3 \leq cn^2$ for all $n \geq N$.

After dividing by n^2 we get that $n \leq c$ for all $n \geq N$.

On the other hand, we know that $\lim_{n\to\infty} n = \infty$, so the function $f(n) = n$ can not be bounded by a positive constant.

Therefore we obtained a contradiction, which means that our initial assumption was false. $\square$

Proof that $n^3 \notin O(n^2)$:

We prove it by contradiction.

Assume the contrary, so suppose that $n^3 \in O(n^2)$.
According to the definition of big O notation, there is a natural number N and a positive constant c such that $n^3 \leq cn^2$ for all $n \geq N$.

After dividing by n^2 we get that $n \leq c$ for all $n \geq N$.

On the other hand, we know that $\lim_{n\to\infty} n = \infty$, so the function $f(n) = n$ can not be bounded by a positive constant.

Therefore we obtained a contradiction, which means that our initial assumption was false. $\square$

Remark: The same reasoning shows that $f(n) = n^k \notin O(n^l)$ if $l < k$ for all $k, l \in \mathbb{R}$.

A property of big O notation:

Claim

If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$.

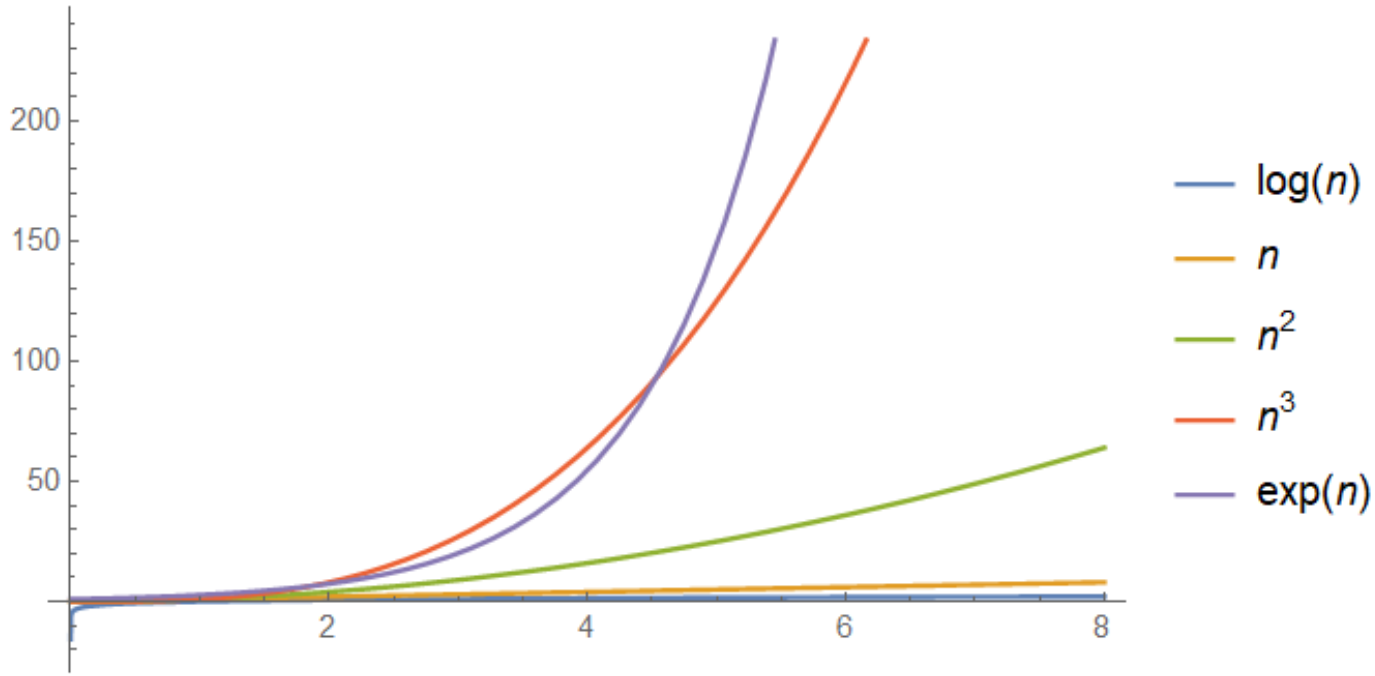
Proof: $f(n) \in O(g(n))$ means that there is a $c_1 > 0$, N_1 pair such that $f(n) \leq c_1 g(n)$ for all $n \geq N_1$.
Similary $g(n) \in O(h(n))$ means that there is a $c_2 > 0$, N_2 pair such that $g(n) \leq c_2 h(n)$ for all $n \geq N_2$.
Combining these we obtain that:
$f(n) \leq c_1 g(n) \leq c_1 c_2 h(n)$ for all $n \geq \max(N_1, N_2)$.
So $f(n) \in O(h(n))$, because $c_1 c_2 > 0$ so $c_1 c_2$, $\max(N_1, N_2)$ is a good pair which satisfies the defintion of $O(h(n))$. $\square$

The hierarchy of functions

Let $f(n) << g(n)$ denote that $f(n) \in O(g(n))$ but $g(n) \notin O(f(n))$.

Claim

- $\log(n) << n^k$ for any positive k
- $n^k << 2^n$ for any k



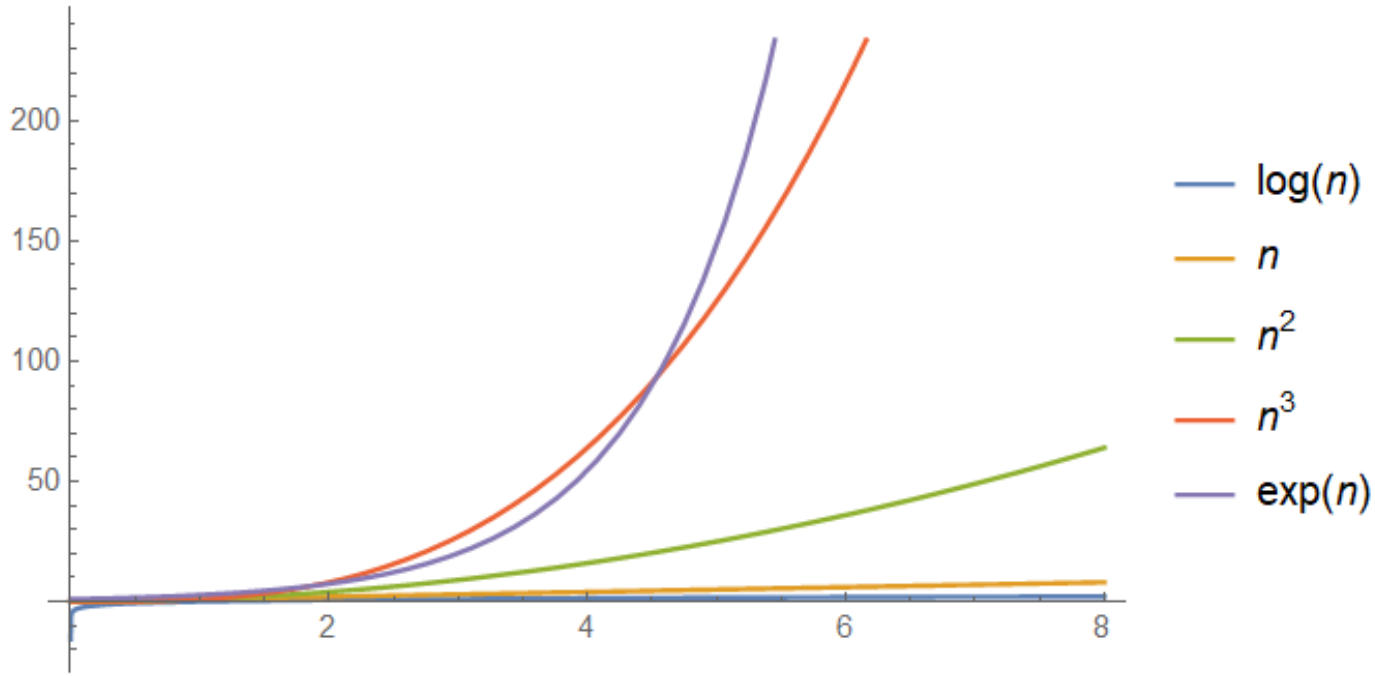
$\log(n) << \sqrt{n} << n << n^2 << n^3 << n^{1000} << 2^n << e^n$.

The hierarchy of functions

Let $f(n) << g(n)$ denote that $f(n) \in O(g(n))$ but $g(n) \notin O(f(n))$.

Claim

- $\log(n) << n^k$ for any positive k
- $n^k << 2^n$ for any k



$\log(n) << \sqrt{n} << n << n^2 << n^3 << n^{1000} << 2^n << e^n$.

The time complexity of addition

Input: Integers a and b. The length of the input is $\log a + \log b = n$. An elementary step is the addition of two bits.

$$\begin{array}{r} 147 \\ +105 \\ \hline 252 \end{array} \qquad\qquad \begin{array}{r} 10010011 \\ +1101001 \\ \hline 11111100 \end{array}$$

Since we make at most two additions at each column (one is comming from the carry bit), the number of operations is at most $2(\max(\log a, \log b)) \leq 2(\log a + \log b) = 2n \in O(n)$.

So the running time is linear in the size of the input. Note that we can not have much faster algorithm for addition since to add two numbers we have to read them and reading require n steps.

The time complexity of addition

Input: Integers a and b. The length of the input is $\log a + \log b = n$. An elementary step is the addition of two bits.

$$\begin{array}{r} 147 \\ +105 \\ \hline 252 \end{array} \qquad\qquad \begin{array}{r} 10010011 \\ +1101001 \\ \hline 11111100 \end{array}$$

Since we make at most two additions at each column (one is comming from the carry bit), the number of operations is at most $2(\max(\log a, \log b)) \leq 2(\log a + \log b) = 2n \in O(n)$.

So the running time is linear in the size of the input. Note that we can not have much faster algorithm for addition since to add two numbers we have to read them and reading require n steps.
Question: We know that we can add any two integers of lenght 8 in 1 minute. How much time do we need to add two integers of lenght 40?

The time complexity of addition

Input: Integers a and b. The length of the input is $\log a + \log b = n$. An elementary step is the addition of two bits.

$$\begin{array}{r} 147 \\ +105 \\ \hline 252 \end{array} \qquad\qquad \begin{array}{r} 10010011 \\ +1101001 \\ \hline 11111100 \end{array}$$

Since we make at most two additions at each column (one is comming from the carry bit), the number of operations is at most $2(\max(\log a, \log b)) \leq 2(\log a + \log b) = 2n \in O(n)$.

So the running time is linear in the size of the input. Note that we can not have much faster algorithm for addition since to add two numbers we have to read them and reading require n steps.
Question: We know that we can add any two integers of lenght 8 in 1 minute. How much time do we need to add two integers of lenght 40?
Answer: Since the time complexity is linear and $40 = 5 \cdot 8$ we can do this task approximately in $5 \cdot 1 = 5$ minutes.