Math 2603 - Lecture 11
Section 8.1 & 8.2 Algorithms

Bo Lin

September 24th, 2019

Algorithms

How computers work

We know that nowadays computers are very powerful. They can do a lot of tasks in a very short period of time.

How computers work

We know that nowadays computers are very powerful. They can do a lot of tasks in a very short period of time.
But what are their limitations? They can only do tasks that people have instructed them to do so.

How computers work

We know that nowadays computers are very powerful. They can do a lot of tasks in a very short period of time.
But what are their limitations? They can only do tasks that people have instructed them to do so.
In general, those instructions should be clear and doable. They are called **algorithms**.

Etymology

Remark

The word "algorithm" comes from the name of a Persian mathematician, Muḥammad ibn Mūsā al-Khwārizmī (c. 780 – c. 850)., who wrote a book about arithmetic of numerals we use today. And the word "algebra" comes from the Latin title of that book.

Definition

Definition

An **algorithm** *is a clearly specified method (or procedure) for solving a problem.*

Definition

Definition

An **algorithm** *is a clearly specified method (or procedure) for solving a problem.*

Remark

An algorithm consists of the following components:

- *the input;*
- *the output;*
- *a sequence of precise steps for converting the input to the output.*

Example: Euclidean algorithm

Example

Euclidean algorithm is a procedure to find $\gcd(a, b)$.

- *Input: nonzero integers* a, b.
- *Output:* $d = \gcd(a, b)$.

Example: Euclidean algorithm

Example

Euclidean algorithm is a procedure to find $\gcd(a, b)$*.*

- *Input: nonzero integers* a, b*.*
- *Output:* $d = \gcd(a, b)$*.*
- *Steps:*
  1. *If* $a < b$*, we may switch* a *and* b*, and the* $\gcd$ *remains unchanged. So we may assume* $a \geq b$*.*

Example: Euclidean algorithm

Example

Euclidean algorithm is a procedure to find $\gcd(a, b)$.

- *Input: nonzero integers* a, b.
- *Output:* $d = \gcd(a, b)$.
- *Steps:*
  1. *If* $a < b$, *we may switch* a *and* b, *and the* $\gcd$ *remains unchanged. So we may assume* $a \geq b$.
  2. *Apply the division algorithm for* a *divided by* b, *we obtain a unique pair of quotient* q *and remainder* r.

Example: Euclidean algorithm

Example

Euclidean algorithm is a procedure to find $\gcd(a, b)$.

- *Input: nonzero integers* a, b.
- *Output:* $d = \gcd(a, b)$.
- *Steps:*
  1. *If* $a < b$, *we may switch* a *and* b, *and the* $\gcd$ *remains unchanged. So we may assume* $a \geq b$.
  2. *Apply the division algorithm for* a *divided by* b, *we obtain a unique pair of quotient* q *and remainder* r.
  3. *Now there are two cases: if* $r = 0$, *the output* d *should be* b, *and we are done;*

Example: Euclidean algorithm

Example

Euclidean algorithm is a procedure to find $\gcd(a, b)$.

- *Input: nonzero integers* a, b.
- *Output:* $d = \gcd(a, b)$.
- *Steps:*
  1. *If* $a < b$, *we may switch* a *and* b, *and the* $\gcd$ *remains unchanged. So we may assume* $a \geq b$.
  2. *Apply the division algorithm for* a *divided by* b, *we obtain a unique pair of quotient* q *and remainder* r.
  3. *Now there are two cases: if* $r = 0$, *the output* d *should be* b, *and we are done; if* $r \neq 0$, *it suffices to compute* b *divided by* r, *and thus we repeat step (2).*

Example: Euclidean algorithm

Example

Euclidean algorithm is a procedure to find $\gcd(a, b)$.

- *Input: nonzero integers* a, b.
- *Output:* $d = \gcd(a, b)$.
- *Steps:*
  1. *If* $a < b$, *we may switch* a *and* b, *and the* $\gcd$ *remains unchanged. So we may assume* $a \geq b$.
  2. *Apply the division algorithm for* a *divided by* b, *we obtain a unique pair of quotient* q *and remainder* r.
  3. *Now there are two cases: if* $r = 0$, *the output* d *should be* b, *and we are done; if* $r \neq 0$, *it suffices to compute* b *divided by* r, *and thus we repeat step (2).*

Remark

Like induction, there may be steps that are repeated many times.

Example: loop and counter

Example

Find an algorithm to compute $\sum_{k=1}^{n} a_k$.

Example: loop and counter

Example

Find an algorithm to compute $\sum_{k=1}^{n} a_k$.

Solution

Input: numbers $a_1, a_2, \cdots, a_n$. *Output: their sum* $S = \sum_{k=1}^{n} a_k$.

1. *Set* $S = 0$.
2. *For* $i = 1$ *to* n, *replace* S *by* $S + a_i$.
3. *Output* S.

Example: loop and counter

Example

Find an algorithm to compute $\sum_{k=1}^{n} a_k$.

Solution

Input: numbers $a_1, a_2, \cdots, a_n$. *Output: their sum* $S = \sum_{k=1}^{n} a_k$.

1. *Set* $S = 0$.
2. *For* $i = 1$ *to* n, *replace* S *by* $S + a_i$.
3. *Output* S.

Remark

Step (2) *is called a* **loop**. *The variable* i *is called a* **counter**.

Horner's Algorithm

Example

Let integer $n \geq 0$. Given integers $a_0, a_1, \cdots, a_n, x$, evaluate the expression

$$\sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + \cdots + a_n x^n.$$

Horner's Algorithm

Example

Let integer $n \geq 0$. *Given integers* $a_0, a_1, \cdots, a_n, x$, *evaluate the expression*

$$\sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + \cdots + a_n x^n.$$

Solution (Horner's Algorithm)

Input: integers $a_0, a_1, \cdots, a_n, x$; *output: the above sum* S.

1. *Set* $S = a_n$.
2. *For* $i = 1$ *to* n, *replace* S *by* $a_{n-i} + S \cdot x$.
3. *Output* S.

Correctness

Remark

When $n = 0$, Horner's algorithm is correct, as the output is $a_n = a_0$.

Correctness

Remark

When $n = 0$, Horner's algorithm is correct, as the output is $a_n = a_0$.
In general, note that the term a_{n-k} is introduced when $i = k$, so it is multiplied by x for exactly $n - k$ times later, and results in a summand of $a_{n-k}x^{n-k}$.

An application of Horner's algorithm

Example

Evaluate $f(-2)$ *where* $f(x) = 4x^3 - 2x + 1$.

An application of Horner's algorithm

Example

Evaluate $f(-2)$ *where* $f(x) = 4x^3 - 2x + 1$.

Solution

We have $n = 3$, $a_0 = 1, a_1 = -2, a_2 = 0, a_3 = 4$ *and* $x = -2$.

An application of Horner's algorithm

Example

Evaluate $f(-2)$ *where* $f(x) = 4x^3 - 2x + 1$.

Solution

We have $n = 3$, $a_0 = 1, a_1 = -2, a_2 = 0, a_3 = 4$ *and* $x = -2$. *The initial value of* S *is* $S = a_3 = 4$.

An application of Horner's algorithm

Example

Evaluate $f(-2)$ *where* $f(x) = 4x^3 - 2x + 1$.

Solution

We have $n = 3$, $a_0 = 1, a_1 = -2, a_2 = 0, a_3 = 4$ *and* $x = -2$. *The initial value of* S *is* $S = a_3 = 4$. *Next*

$$S = a_2 + Sx = 0 + 4 \cdot (-2) = -8.$$
$$S = a_1 + Sx = -2 + (-8) \cdot (-2) = 14.$$
$$S = a_0 + Sx = 1 + 14 \cdot (-2) = -27.$$

So the output is $S = -27$.

Complexity

Motivation

It's good if we have algorithms to solve problems. However, in practice we need to consider other issues.

Motivation

It's good if we have algorithms to solve problems. However, in practice we need to consider other issues.
If an algorithm takes an huge amount of time to generate the output, it is not very useful. So we want to measure the efficiency of algorithms. This is the motivation of our analysis of complexity.

Definition

Definition

For a given algorithm, we can define a **complexity function** $f : \mathbb{N} \to \mathbb{N}$ *such that for some measure* n *of the size of the input,* $f(n)$ *is the upper bound for the number of operations required to carry out the algorithm.*

Definition

Definition

For a given algorithm, we can define a **complexity function** $f : \mathbb{N} \to \mathbb{N}$ *such that for some measure* n *of the size of the input,* $f(n)$ *is the upper bound for the number of operations required to carry out the algorithm.*

Example

In the algorithm summing n *integers,* $f(n) = n$.

Definition

Definition

For a given algorithm, we can define a **complexity function** $f : \mathbb{N} \to \mathbb{N}$ *such that for some measure* n *of the size of the input,* $f(n)$ *is the upper bound for the number of operations required to carry out the algorithm.*

Example

In the algorithm summing n *integers,* $f(n) = n$*. In Horner's algorithm,* $f(n) = 2n + 1$*.*

Complexity is approximate

Remark

In most cases, it is impossible to rigorously evaluate $f(n)$. *For example, we don't know exactly how many divisions we need to do for the Euclidean algorithm.*

Complexity is approximate

Remark

In most cases, it is impossible to rigorously evaluate $f(n)$. For example, we don't know exactly how many divisions we need to do for the Euclidean algorithm.Fortunately, it does not matter very much whether $f(n)$ is $2n$ or $2n+1$ when n is large. As a result, keep in mind that when we talk about complexity, we almost always use approximations.

Complexity is approximate

Remark

In most cases, it is impossible to rigorously evaluate $f(n)$. For example, we don't know exactly how many divisions we need to do for the Euclidean algorithm.Fortunately, it does not matter very much whether $f(n)$ is $2n$ or $2n+1$ when n is large. As a result, keep in mind that when we talk about complexity, we almost always use approximations.

Remark

Another issue is: does it make sense to just counter the number of operations? Is it possible that different operations are quite different in nature?

Complexity is approximate

Remark

In most cases, it is impossible to rigorously evaluate $f(n)$. For example, we don't know exactly how many divisions we need to do for the Euclidean algorithm.Fortunately, it does not matter very much whether $f(n)$ is $2n$ or $2n+1$ when n is large. As a result, keep in mind that when we talk about complexity, we almost always use approximations.

Remark

Another issue is: does it make sense to just counter the number of operations? Is it possible that different operations are quite different in nature?It is possible, for example, adding $13+21$ is way much simpler than adding two 50-digit integers. Nonetheless, in most cases and in this course, we can still analyze approximately like this.

The Big Oh notation

As a result, we introduce symbols to describe **asymptotic** behaviors of complexity functions.

The Big Oh notation

As a result, we introduce symbols to describe **asymptotic** behaviors of complexity functions.

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions. We say that f is **Big Oh** *of g, written $f = \mathcal{O}(g)$ (LaTeX symbol \mathcal{O}), if there is an integer n_0 and a positive real number c such that*

$$|f(n)| \leq c|g(n)| \,\forall\, n \geq n_0.$$

The Big Oh notation

As a result, we introduce symbols to describe **asymptotic** behaviors of complexity functions.

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$ *be functions. We say that* f *is* **Big Oh** *of* g, *written* $f = \mathcal{O}(g)$ *(LaTeX symbol \mathcal{O}), if there is an integer* n_0 *and a positive real number* c *such that*

$$|f(n)| \leq c|g(n)| \, \forall \, n \geq n_0.$$

Remark

Intuitively, $f = \mathcal{O}(g)$ *if for sufficiently large* n, $|f(n)|$ *is "dominated" by* $|g(n)|$.

Example: comparing functions

Example

By induction we already proved that $2^n > n$ *for all* $n \in \mathbb{N}$. *So if* $f(n) = n$ *and* $g(n) = 2^n$, *we can take* $c = 1$ *and* $n_0 = 1$ *and we have* $f = \mathcal{O}(g)$.

Example: comparing functions

Example

By induction we already proved that $2^n > n$ *for all* $n \in \mathbb{N}$*. So if* $f(n) = n$ *and* $g(n) = 2^n$*, we can take* $c = 1$ *and* $n_0 = 1$ *and we have* $f = \mathcal{O}(g)$*.*

Example

If $f(n) = 100n$*,* $g(n) = n^2$*. Is* $f = \mathcal{O}(g)$*?*

Example: comparing functions

Example

By induction we already proved that $2^n > n$ *for all* $n \in \mathbb{N}$*. So if* $f(n) = n$ *and* $g(n) = 2^n$*, we can take* $c = 1$ *and* $n_0 = 1$ *and we have* $f = \mathcal{O}(g)$*.*

Example

If $f(n) = 100n$*,* $g(n) = n^2$*. Is* $f = \mathcal{O}(g)$*?*

Solution

Yes. We can take $c = 1, n_0 = 100$*, or alternatively* $c = 100, n_0 = 1$*.*

Some properties of Big Oh

Proposition

Let $f, g, f_1, g_1 : \mathbb{N} \to \mathbb{R}$.

1. *If* $f = \mathcal{O}(g)$, *then* $f + g = \mathcal{O}(g)$.
2. *If* $f = \mathcal{O}(f_1)$ *and* $g = \mathcal{O}(g_1)$, *then* $fg = \mathcal{O}(f_1 g_1)$.

Some properties of Big Oh

Proposition

Let $f, g, f_1, g_1 : \mathbb{N} \to \mathbb{R}$.

1. *If* $f = \mathcal{O}(g)$, *then* $f + g = \mathcal{O}(g)$.
2. *If* $f = \mathcal{O}(f_1)$ *and* $g = \mathcal{O}(g_1)$, *then* $fg = \mathcal{O}(f_1 g_1)$.

Sketch of proof.

(1) When $|f(n)| \leq c|g(n)|$, we also have

$$|(f+g)(n)| = |f(n) + g(n)| \leq |f(n)| + |g(n)| \leq (c+1)|g(n)|.$$

Some properties of Big Oh

Proposition

Let $f, g, f_1, g_1 : \mathbb{N} \to \mathbb{R}$*.*

1. *If* $f = \mathcal{O}(g)$*, then* $f + g = \mathcal{O}(g)$*.*
2. *If* $f = \mathcal{O}(f_1)$ *and* $g = \mathcal{O}(g_1)$*, then* $fg = \mathcal{O}(f_1 g_1)$*.*

Sketch of proof.

(1) When $|f(n)| \leq c|g(n)|$, we also have

$$|(f+g)(n)| = |f(n) + g(n)| \leq |f(n)| + |g(n)| \leq (c+1)|g(n)|.$$

(2) When $|f(n)| \leq c_1|f_1(n)|$ and $|g(n)| \leq c_2|g_1(n)|$, we also have

$$\begin{aligned} |(fg)(n)| &= |f(n)g(n)| = |f(n)| \cdot |g(n)| \\ &\leq c_1 c_2 |f_1(n)| \cdot |g_1(n)| = c_1 c_2 |(f_1 g_1)(n)|. \end{aligned}$$

Order of functions

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions. We say that f has **smaller order** *than g, written $f \prec g$ (LaTeX symbol \prec), if $f = \mathcal{O}(g)$ and $g \neq \mathcal{O}(f)$.*
If $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$, we say that f and g have the **same order** *and write $f \asymp g$ (LaTeX symbol \asymp).*

Order of functions

Definition

Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions. We say that f has **smaller order** *than g, written $f \prec g$ (LaTeX symbol \prec), if $f = \mathcal{O}(g)$ and $g \neq \mathcal{O}(f)$.*
If $f = \mathcal{O}(g)$ and $g = \mathcal{O}(f)$, we say that f and g have the **same order** *and write $f \asymp g$ (LaTeX symbol \asymp).*

Proposition

$\asymp$ is an equivalence relation defined on the set of functions $\mathbb{N} \to \mathbb{R}$.

Perspective from limits

Proposition

Let $f, g : \mathbb{N} \to \mathbb{R}$ *be functions.*

1. *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, *then* $f \prec g$.
2. *if* $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty$, *then* $g \prec f$.
3. *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$, *where* L *is a nonzero real number, then* $f \asymp g$.

Perspective from limits

Proposition

Let $f, g : \mathbb{N} \to \mathbb{R}$ be functions.

a. *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, *then* $f \prec g$.
b. *if* $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = \infty$, *then* $g \prec f$.
c. *if* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$, *where* L *is a nonzero real number, then* $f \asymp g$.

Corollary

A polynomial function of n has the same order as its highest power: if $f(n) = a_t n^t + \cdots + a_1 n + a_0$ is a polynomial with degree t, then $f(n) \asymp n^t$.

A hierarchy of orders

Remark

We have the following hierarchy of orders:

$$\frac{1}{n} \prec 1 \prec \log n \prec \sqrt{n} \prec \frac{n}{\log n} \prec n \prec n\log n \prec n^2 \prec n^3 \prec \cdots$$

A hierarchy of orders

Remark

We have the following hierarchy of orders:

$$\frac{1}{n} \prec 1 \prec \log n \prec \sqrt{n} \prec \frac{n}{\log n} \prec n \prec n \log n \prec n^2 \prec n^3 \prec \cdots$$

$$n^t \prec 2^n \prec 3^n \prec \cdots \prec n! \prec n^n \prec n^{n^n} \prec \cdots$$

Example: complexity of additions

Example

How many steps of single-digit additions are there at most when adding two n-digit integers?

Example: complexity of additions

Example

How many steps of single-digit additions are there at most when adding two n-digit integers?

Solution

Unit digit: 1 *step;*

Example: complexity of additions

Example

How many steps of single-digit additions are there at most when adding two n-digit integers?

Solution

Unit digit: 1 step; carry may happen at every digit, so for each subsequent digit, 2 steps. In total $1 + 2(n-1) = 2n - 1$ steps.

Example: complexity of additions

Example

How many steps of single-digit additions are there at most when adding m n*-digit integers?*

Example: complexity of additions

Example

How many steps of single-digit additions are there at most when adding m n-digit integers?

Solution

First addition: $2n-1$ *steps.*

Example: complexity of additions

Example

How many steps of single-digit additions are there at most when adding m n-digit integers?

Solution

First addition: $2n-1$ *steps. Next, note that the first sum may have one more digit, so* $2(n+1)-1=2n+1$. *And subsequent additions:* $2n+3, 2n+5, \cdots$ *In total we have* $m-1$ *additions:*

$$(2n-1)+\cdots+(2n+2m-5)=(m-1)(2n+2m-3)<m(2n+m).$$

Example: complexity of additions

Example

How many steps of single-digit additions are there at most when adding m n-digit integers?

Solution

First addition: $2n-1$ *steps. Next, note that the first sum may have one more digit, so* $2(n+1)-1=2n+1$*. And subsequent additions:* $2n+3, 2n+5, \cdots$ *In total we have* $m-1$ *additions:*

$$(2n-1)+\cdots+(2n+2m-5)=(m-1)(2n+2m-3)<m(2n+m).$$

Corollary

When $m \leq n$*, this complexity* $< n(2n+n)=3n^2$*, which is* $\mathcal{O}(n^2)$*; when* $n \leq m$*, this complexity* $< m(2m+m)=3m^2$*, which is* $\mathcal{O}(m^2)$*.*

Homework Assignment #6

Section 8.1 Exercise 8,9,16,17(d)-Horner's algorithm only.
Section 8.2 Exercise 7(b)(f),12,17,19(a)(c)(f).