Algorithm Analyses

Hoang Anh Quan

June 22, 2018

Outline

Abstract

This is a brief introduction to algorithm analyses. Firstly, the report introduce a definition of an algorithm and how it can be performed. Secondly, we consider five properties of an algorithm and the criteria to judge it to be efficient or not. Nextly, some notations are used to define the growing speed of time execution of algorithms. Finally, we analyze algorithms to sovle the proplem finding the shortest distance between two points in the set of given points in plane.

The first ‘algorithm’

Problem: find the real zeros of the equation: $ax^2 + bx + c = 0$

Cases			Set of zeros
$a = 0$	$b = 0$	$c = 0$	$\mathbb{R}$
		$c \neq 0$	$\emptyset$
	$b \neq 0$	$\forall c$	$\left\{\frac{-c}{b}\right\}$
$a \neq 0, \Delta = b^2 - 4ac$		$\Delta \geqslant 0$	$\left\{\frac{-b+\sqrt{\Delta}}{2a}, \frac{-b-\sqrt{\Delta}}{2a}\right\}$
		$\Delta < 0$	$\emptyset$

Table 1: Cases and their real zeros

What is an algorithm?

Definition

Algorithm is a finite list of well-defined instructions for calculating a function.

Something magically beautiful happens when a sequence of commands and decisions is able to marshal a collection of data into organized patterns or to discover hidden structure.

Donald Knuth

Expressing an algorithm

[1]An algorithm may be expressed in a number of ways, including:

-Natural language:	verbose and ambiguous;
-Flowchart:	avoid most issues of ambiguity, largely standardized;
-Pseudo-code:	avoids most issues of ambiguity, resembles common elements of programming languages, no specific agreement on syntax;
-Programming language:	need to express low-level deatails that are not necessary for a high-level understanding.

[1]Paraphrased slide 5, http://courses.cs.vt.edu/cs2104/Fall12/notes/T16_Algorithms.pdf

Expressing an algorithm

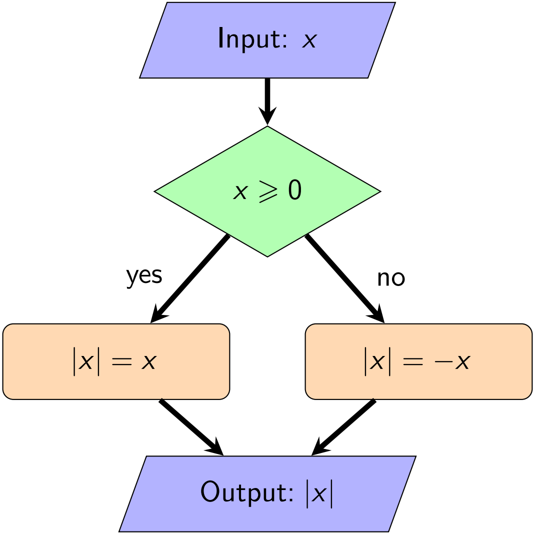
Problem: Demonstrate an algorithm to calculate the absolute value, denoted $|.|$, of a given real number.

Expressing an algorithm

Problem: Demonstrate an algorithm to calculate the absolute value, denoted $|.|$, of a given real number.
a) Natural language: If x is a nonnegative number, then the absolute value of x is x. If x is a negative number, then the absolute value of x is $-x$.

Expressing an algorithm

b) Flowchart:



Expressing an algorithm

c) Pseudo-code:

$x \leftarrow$ input
if $x \geqslant 0$ **then**
  $abs \leftarrow x$
else
  $abs \leftarrow -x$
end if
return abs

Expressing an algorithm

d) Programming language:

```cpp
#include <iostream>
int x;

int main()
{
    std::cin >> x;
    if (x>=0)   {std::cout <<  x;}
    else        {std::cout << -x;};
    return 0;
}
```

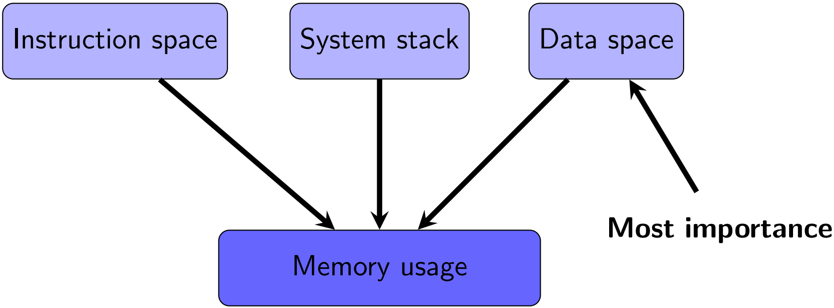
Figure 1: Algorithm written in C++

How to judge an algorithm?

Space complexity

Space complexity is the amount of memory used by the algorithm (including the input values to the algorithm) to execute and produce the result. It contains:
-Instruction space: It's the amount of memory used to save the compiled version of instructions.
-System stack: If a function A() calls function B() inside it, then all the variables of the function A() will get stored on the system stack temporarily, while the function B() is called and executed inside the funciton A().
-Data space: Amount of space used by the variables and constants.

Space complexity



Space complexity

Type	Size
bool, char	1 byte
short	2 bytes
float, int	4 bytes
double, long	8 bytes

Table 2: Size of variable types in C++

Time complexity

System independent effects:
-Algorithm
-Input data
System dependent effects:
-Hardware: CPU, memory, cache,...
-Software: complier, interpreter, garbage collector,...
-System: operating system, network, other apps,...
Mathematical models for running time.

Total running time = sum of cost x frequency for all operations

$$T = c_1.f_1 + c_2.f_2 + ... + c_n.f_n$$

Time complexity

Operations	**Cost**	**Frequency**
1^{st}	c_1	f_1
2^{nd}	c_2	f_2
3^{rd}	c_3	f_3
...	...	...

Table 3: Cost & frequency of operations

[2]The RAM Model of Computation:
-Each *simple* operation (+,-,*,/,if,call) takes exactly one time step.
-Loops and subroutines are *not* considered simple operations.
-Each memory access takes exactly one time step.

[2]Page 31, The algorithm design manual, Steven S. Skiena

Time complexity

Running time of an algorithm $\xleftrightarrow{\text{RAM Model}}$ Number of steps the algorithm takes

If we **assume** that our RAM executes a given number of steps per second, which means it takes T seconds to executes a step, then:

$$\boxed{\text{Running time} = \text{T x number of steps}}$$

The RAM Model is not true. However, it is useful in practice.

Best, Average, and Worst-case complexities

[3]Using the RAM model of computation, we can count how many steps our algorithm takes on any given input instance by executing it. However, to understand how good or bad an algorithm is in general, we must know how it works over all instances.

[3]Page 32, The algorithm design manual, Steven S. Skiena

Case complexity

Type	Definition	Input	Result
Best case	the minimum number of steps taken in any instance of size n	easi-est	a goal for all inputs
Worst case	the maximum number of steps taken in any instance of size n	most diffi-cult	a guarantee for all inputs
Average case	the average number of steps over all instances of size n	ran-dom	a way to predict performance

Table 4: Type of case complexity

Useful notations

The best, worst, and average-case time complexities for any given algorithm are numerical functions over the size of possible problem instances. However, it is hard to deal with these complicated functions. Such as

$$f(n) = n! + n^4 + \log n$$

Dominance relations

We say g dominates f, denoted as $g \gg f$, when $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$.
Chain of dominance

$$\boxed{n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \sqrt{n} \gg \log n \gg 1}$$

Using dominance relations, we can simplify our analysis by ignoring some terms without impact our overall judgement of algorithms. For instance, $f(n) = n! + n^4 + \log n$ is 'the same' as $g(n) = n!$, for large enough n.

The Big Oh, Omega, Theta notations

There are notations to make the comparing functions easier. They are $O, \Omega,$ and Θ.

$\bullet f(n) = O(g(n))$ if there exists some constant c such that $f(n) \leqslant c \cdot g(n)$, for large enough n.(▲)

$\bullet f(n) = \Omega(g(n))$ if there exists some constant c such that $f(n) \geqslant c \cdot g(n)$, for large enough n.

$\bullet f(n) = \Theta(g(n))$ if there exists some constant c_1 and c_2 such that $c_1 \cdot g(n) \leqslant f(n) \leqslant c \cdot g(n)$, for large enough n.

The Big Oh, Omega, Theta notations

Abuse of notations

Some consider $O, \Omega,$ and Θ to be abuse of notations.
$O(n) = O(n^2)$ is true but $O(n^2) = O(n)$ is not. To be more precise, for instance, $O(g(n))$ is the class of all functions $f(n)$ satisfy ($\blacktriangle$). In that case, $f(n) \in O(g(n))$
Knuth pointed out that "mathematicians customarily use the equal sign '=' as they use the word 'is' in English: Aristotle is a man, but a man isn't necessarily Aristotle."

Comparing running times & algorithms

Processor:	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Installed memory (RAM):	4.00 GB (3.89 GB usable)
System type:	64-bit Operating System, x64-based processor

This CPU has a frequency of 1.6GHz then this means that it can produce 1.6 billion cycles per second. So we can assume an average computer can perform 1 billion operations in a second.

Comparing running times & algorithms

With that assumption, now we can estimate the running time by knowing the complexity of an algorithms[4].

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Figure 2: Time executed with given complexities

[4]Page 38, The algorithm design manual, Steven S. Skiena

Algorithms analyses

Best, Worst, and Average-case analyses

Example 1: Given n real numbers $a_1; a_2; ...; a_n$. Point out the index i such that

$$a_i = \min\{a_j | j = 1, ..., n; a_j \geqslant 0\}.$$

Algorithm: Create a loop with index i from 1 to n to find the minimum value of $a_1, a_2, ..., a_n$. If it exists $a_i = 0$ then break the loop.

Analyses:

-Best case: 1 step $= O(1)$ with the input $(0; 0; ...; 0)$.

-Worst case: n steps $= O(n)$ with the input $(1; 1; ...; 1)$.

-Average case:

+If there is no '0' in the given input, the algorithm executes in n steps.

+If there is a '0' in the given input, the algorithm stops where the first '0' takes place.

On average, the algorithm stops after n steps, which is $O(n)$.

Best, Worst, and Average-case analyses

Example 2: Given n real numbers $a_1; a_2; ...; a_n$ which belong to the set $\{0; 1; 2\}$. Point out a index i such that $a_i = 0$.

Algorithm: Create a loop with index i from 1 to n to test whether $a_i = 0$ or not.

Analyses:

-Best case: 1 step $= O(1)$ with the input $(0; 0; ...; 0)$.

-Worst case: n steps $= O(n)$ with the input $(2; 2; ...; 2)$.

-Average case: 3 steps $= O(1)$.

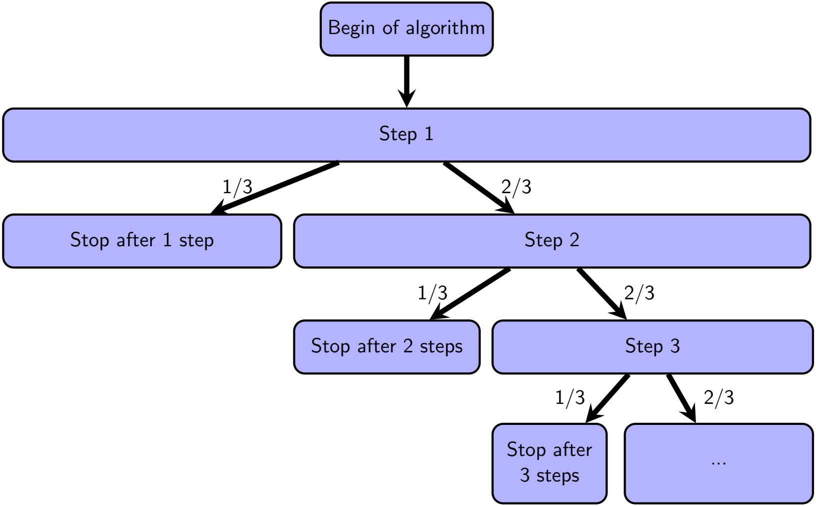
In each step, the algorithm have a $\frac{1}{3}$ chance of stopping, a $\frac{2}{3}$ chance of moving to next step (if it's not the end).

The average number of steps executed by the algorithm is calculated by the following expression

$$\frac{1}{3} \cdot \sum_{i=1}^{n} \left(\frac{2}{3}\right)^{i-1} \cdot i + \left(\frac{2}{3}\right)^{n} \cdot n = 3$$

Best, Worst, and Average-case analyses

Example 2: Given n real numbers $a_1; a_2; \ldots; a_n$ which belong to the set $\{0; 1; 2\}$. Point out a index i such that $a_i = 0$.



Problem

Find the smallest distance between two points of given n points in plane.[5]

[5]Page 51, Algorithm design, Jon Kleinberg & Eva Tardos

Brute-force approach - an $O(n^2)$ algorithm

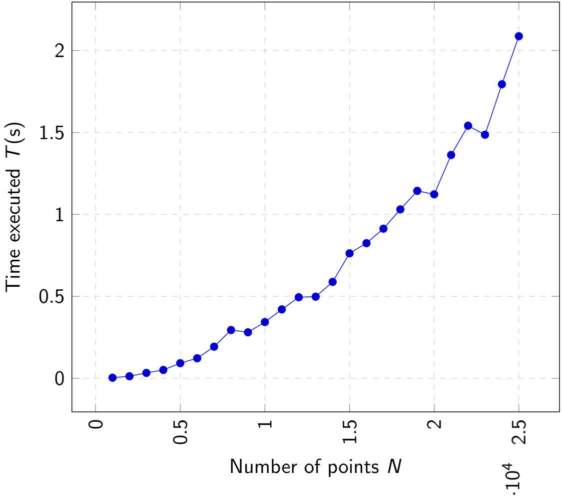


Figure 3: Chart showing the relation between number of points and time executed of the brute-force algorithm

Brute-force approach - an $O(n^2)$ algorithm

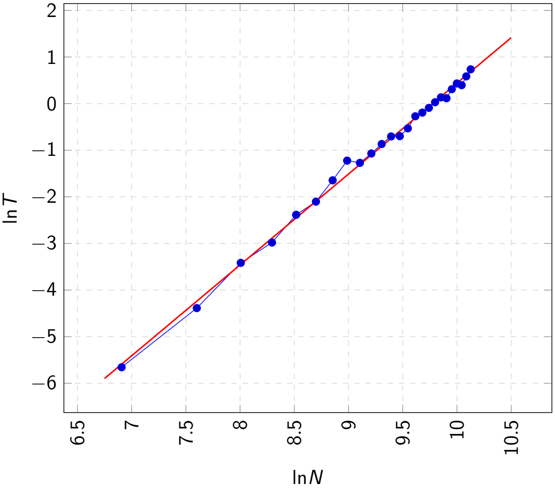


Figure 4: ln-ln chart

Brute-force approach - an $O(n^2)$ algorithm

By using least square method, we can guess the following relation between N and T

$$\ln T = 1.95 \ln N - 19.06$$
$$T = 5.28 \cdot 10^{-9} N^{1.95}$$

Additions

Non-deterministic algorithms

Given a particular input, a non-deterministic algorithm is an algorithm which does not always produce the same output after passing through the same sequence of states.

Algorithms[6]	**Upper bound of running time**	**Certainty**
Brute-force algorithm	$O(\sqrt{n})$	100%
AKS test (2002)	$O((\log n)^{6+\epsilon})$	100%
Miller-Robin test	$O(k \cdot (\log n)^3)$	4^{-k} chance misjudge a composite number

Table 5: Primality test

[6] Wikipedia

Quantum algorithms

Algorithms to factor an integer N	Time complexity
Shor's quantum algorithm	$O((\log N)^2(\log\log N)(\log\log\log N)$
General number field sieve	$O(\epsilon^{1.9(\log\ N)^{1/3}(\log\ \log\ N)^{2/3}})$

Table 6: Primality test

Reference

- Intro Problem Solving in Computer Science, `http://courses.cs.vt.edu/cs2104/Fall12/notes/T16_Algorithms.pdf`
- Skiena, S. S. (2008). The algorithm design manual (2nd ed.). Springer.
- Kleinberg, J., & Tardos, E. (2006). The algorithm design. Pearson, Addison Wesley.