Running Time and Elementary Operations

Richard Hua

Semester 2, 2021

Program input

We wish to execute the same program on multiple inputs
Probably a bad idea to hard-code the input in the source file
Always prompting the user for manual input is annoying
Need something else

Standard streams

Standard streams are interconnected input and output communication channels used by computer programs. There are three basic I/O connections:

- Standard input (stdin)
- Standard output (stdout)
- Standard error (stderr)

Standard streams

When a command is executed on shell, the streams are connected to the terminal on which the shell is running by default. And so you see the output and/or any error messages of the program on screen.

The streams can also be redirected via system commands and/or pipeline.

This is how the automarker will run your submitted program for your assignment, i.e. your program will need to read the specified input from the standard input stream and send the output to the standard output stream.

Example

```java
import java.util.Scanner;
public class example{
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                while(sc.hasNext()){
                        System.out.println(sc.nextLine());
                }
        }
}
```

Simple Java program to read all lines from the standard input stream and write them as they are to the standard output stream.

Example

```
1 import sys
2
3 for line in sys.stdin.readlines():
4     print(line.strip())
```

Python program that does exactly the same thing.

Example

```
1 Great fleas Have Lesser Fleas,
2 upon Their backs to Bite'em,
3 And Lesser Fleas Have Lesser fleas,
4 And So, Ad infinitum.
5 and Those great Fleas, Themselves, In turn
6 Have Greater Fleas To go On;
7 while Those Again have Greater still,
8 And greater Still, And So on.    -- Augustus De Morgan
```

Poem by De Morgan about fleas and recursions. We have it in a text file called poem.txt

Example

```
sc-cs-316976:standard_streams_example rwan074$ cd ~/Desktop/teaching/SE284/standard_streams_e
sc-cs-316976:standard_streams_example rwan074$ ls
example.java    example.py      poem.txt
sc-cs-316976:standard_streams_example rwan074$
```

Example

```
sc-cs-316976:standard_streams_example rwan074$ javac example.java
```

Example

```
sc-cs-316976:standard_streams_example rwan074$ java example < poem.txt
Great fleas Have Lesser Fleas,
upon Their backs to Bite'em,
And Lesser Fleas Have Lesser fleas,
And So, Ad infinitum.
and Those great Fleas, Themselves, In turn
Have Greater Fleas To go On;
while Those Again have Greater still,
And greater Still, And So on.    -- Augustus De Morgan
```

Example

```
sc-cs-316976:standard_streams_example rwan074$ java example < poem.txt > output
sc-cs-316976:standard_streams_example rwan074$ ls
example.class   example.java    example.py     _output          poem.txt
```

Sets

Definition

A **set** is an unordered collection of distinct object, called **elements** of the set.

We write $a \in X$ to mean that a is an element of X.
Some important common sets are:
$\mathbb{N} = \{0, 1, 2, \cdots\}$, the set of natural numbers,
$\mathbb{R}$, the set of real numbers, and
$\emptyset = \{\}$, the empty set having no elements.
Notation: List elements, e.g. $X = \{2, 3, 5, 7, 11\}$ or use set-builder notation $X = \{x \in \mathbb{N} \mid x \text{ is prime and } x < 12\}$.

Common set operations

- $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$, the **intersection** of A and B.
- $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$, the **union** of A and B.
- $|A|$ is the number of elements of A, the **cardinality** of A.

Exercise

Exercise

What is the cardinality of $A = \{1, 1, 5\}$*? Of* $B = \{x^2 \mid x \in \{-1, 1, 2\}\}$*? What is* $A \cap B$ *and* $A \cup B$*?*

Note that we don't allow duplicates in sets by definition so $A = \{1, 5\}$ and $B = \{1, 4\}$.

Functions

Definition

A **function** is a mapping f from a set X (the **domain**) to a set Y (the **codomain**) such that every $x \in X$ maps to a unique $y \in Y$. We denote this by $f : X \to Y$.

Important properties a function may (or may not) have:
A function $f : X \to Y$ is **one-to-one** if whenever $x \neq x'$, we have $f(x) \neq f(x')$.
A function $f : X \to Y$ is **onto** if for every $y \in Y$, there exists some $x \in X$ such that $f(x) = y$.
A function that is both one-to-one and onto is called a **bijection**.
And only bijections have inverses, i.e. we can obtain $f^{-1} : Y \to X$.

Examples

- Power functions $f(x) = x$, $f(x) = x^2$, $f(x) = x^3$, etc.
- Exponential functions $f(x) = 2^x$, $f(x) = (1.5)^x$, etc.
- Logarithm (inverse of exponential) is defined by $\log_a(y) = x$ if $y = a^x$ where a is called the **base** of the log function. Its domain is $\{x \in \mathbb{R} \mid x > 0\}$. We sometimes just write $\log_a y$, omitting the parentheses if there is no confusion. We sometimes write $\ln = \log_e$ and $\lg = \log_2$ for notational convenience.
- **Ceiling** rounds up to the nearest integer, e.g. $\lceil 3.7 \rceil = 4$.
- **Floor** rounds down to the nearest integer, e.g. $\lfloor 3.7 \rfloor = 3$.

Sums

Definition

A **sequence** is a function $f : \mathbb{N} \to \mathbb{R}$.

Notation: $f_0, f_1, f_2, \cdots$ where $f_i = f(i)$.
Sum notation: $f_m + f_{m+1} + \cdots + f_n = \sum_{i=m}^{n} f_i$.
Useful formulas

- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.
- $\sum_{i=m}^{n} a^i = \frac{a^{n+1} - a^m}{a-1}$.

Exercise

Simplify $\sum_{i=1}^{2n} 2^i$.

$2^{2n+1} - 2 = 2(2^n)^2 - 2$.

Limits

Example

$\lim_{x\to\infty} x^2 = \infty$, $\lim_{x\to\infty} 1/x = 0$.

Slightly trickier example. What is $\lim_{n\to\infty} \frac{e^x}{x^2}$? Naive answer, $\lim_{n\to\infty} \frac{e^x}{x^2} = \frac{\lim_{x\to\infty} e^x}{\lim_{x\to\infty} x^2} = \frac{\infty}{\infty} = 1$. It's incorrect because the top and bottom functions could be approaching infinity at drastically different rate.

These limits of *indeterminate* forms can be calculated using L'Hospital's Rule.

$\lim_{x\to c} \frac{f(x)}{g(x)} = \lim_{x\to c} \frac{f'(x)}{g'(x)}$ provided that $\lim_{x\to c} \frac{f'(x)}{g'(x)}$ exists.

L'Hospital's rule example

$\lim_{x\to\infty} \frac{e^x}{x^2} = \lim_{x\to\infty} \frac{e^x}{2x}$. And it's still indeterminate ☹.
When in doubt, do it again. $\lim_{x\to\infty} \frac{e^x}{2x} = \lim_{x\to\infty} \frac{e^x}{2} = \infty$ ☺.

Fibonacci sequence

The famous Fibonacci sequence is recursively defined by

$$F(n) = \begin{cases} n & \text{if } n = 0 \text{ or } n = 1 \\ F(n-1) + F(n-2) & \text{if } n \geqslant 2. \end{cases}$$

Fibonacci sequence

The recursive nature of the Fibonacci sequence immediately suggests a recursive algorithm.

function SLOWFIB(integer n)
    if $n < 0$ then return 0
    else if $n = 0$ or $n = 1$ then return n
    else return SLOWFIB(n-1) + SLOWFIB(n-2)

The above function is obviously correct, but it does a lot of repeated computations. We can do a lot better.

Fibonacci sequence

Iterative algorithm to compute $F(n)$.

function FASTFIB(integer n)
    if $n < 0$ then return 0
    else if $n = 0$ or $n = 1$ then return n
    else
        $a = 1$
        $b = 0$
        for i in range(2 to n) do
            $t = a$
            $a = a + b$
            $b = t$
    return a

Fibonacci sequence

How many additions, function calls, boolean operations, etc are needed by the FASTFIB algorithm to compute $F(n)$?
Take the number of additions for example, the main loop (if we ignore the number of additions needed by the for-loop itself) does 1 addition per iteration and so a total of $n-2$ addition operations are executed.
What about SLOWFIB?
At least $F(n)$ additions.

Fibonacci sequence

Question

Say a particular program that we have coded based on the FASTFIB algorithm takes 1 second to compute $F(10)$. How long will it take the same program to compute $F(100)$?

Answer: I don't know ☹.

But it's pretty safe to assume it will probably take longer as n gets larger.

Running time measurement

The running time of the algorithm depends on input. What's the issue here?
Using clock-time directly is probably not a good idea.
We want an input-independent measurement.

Input

The example we have here is computing the nth value of a sequence $F(n)$. Since all integers are unique, it wouldn't be too much trouble. But say we have something more complicated (e.g. sorting a list of 100 integers, say in range from 1 to 100), if we use the input (the list) itself as a parameter to our time measurement method, we would potentially need to specify for which input we get which time etc.. and there are 100^{100} possible input to specify. There are only about 10^{24} stars and 6×10^{79} atoms in the observable universe...

Input size

The running time heavily depends on the size of the input. Some examples on input size:

Example

- The length of the list we are trying to sort.
- Number of vertices and/or edges in a graph.
- The number of bits we need to represent an integer n.

You need at least $\lfloor \lg n \rfloor + 1$ bits by the way.
Since the input and input size are related in a very straightforward way (in the $F(n)$ case), we can use n itself as a measurement. But be careful how you measure input sizes.

Elementary operations

Most modern computers and languages build complex programs from ordinary arithmetic and logical operations such as standard unary and binary arithmetic operations (negation, addition, subtraction, multiplication, division, modulo operation, or assignment), Boolean operations, binary comparisons ("equals", "less than", or "greater than"), branching operations, and so on. It is quite natural to use these basic computer instructions as algorithmic operations, which we will call **elementary operations**.

Definition

An **elementary operation** is any operation whose execution time does not depend on the size of the input.

Elementary operations

Definition

The **running time** (or computing time) of an algorithm is the number of its elementary operations (informal).

Question

What are some examples of elementary operations?

Example

- Adding two integers?
- Writing one bit to memory?
- Reading one bit from memory?

Elementary operations

Note that the our concept of an elementary operation here is a very loose idea and different sources may have a different perspective on this matter.
For example, addition of two 64-bit integers should definitely count as elementary, since it can be done in a fixed time. But for some applications, such as cryptography, we must deal with much larger integers (some RSA applications nowadays uses numbers with more than 2000 bits), which must be represented in another way. Addition of such "big" integers takes a time roughly proportional to the size of the integers, so it is not reasonable to consider it as elementary. From now on we shall ignore such problems. For most of the examples in this course. However, they must be considered in some situations, and this should be borne in mind.

Elementary operations

We could choose a rigid model say the Random Access Machine (RAM) and work on this model only in this course. Let's have a look at its elementary operations:

- Adding 1,
- writing a bit,
- reading a bit,
- etc

It will a nightmare to count the number of these operations even in a simple algorithm.

Running time

What we have so far: There are three main characteristics of an algorithm designed to solve a given problem.
Domain of definition: The set of legal inputs.
Correctness: The algorithm gives correct output for each legal input. This depends on the problem we are trying to solve and can be tricky to prove.
Resource usage: We focus on time efficiency here using our measure of running time.

Definition

We define a notion of **input size** on the data. This is a non-negative integer; for example, the number of records in a database to be sorted.

Just because we didn't talk about them...

Brenan Keller
@brenankeller

A QA engineer walks into a bar.
Orders a beer. Orders 0 beers.
Orders 9999999999 beers.
Orders a lizard. Orders -1 beers.
Orders a ueicbksjdhd.

First real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.

1:21 PM · 30 Nov 18