Kazakh-British Technical University

Algorithms and Data Structures, Spring 2011

Lecture 1: Algorithm Analysis

1 Introduction to Algorithms

What is *algorithm*?
The following description of this term we may observe in *OxfordDictionary (Lingvo)*:

> *– a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer;*
> *Origin:*
> *late 17th cent. (denoting the Arabic or decimal notation of numbers): variant (influenced by Greek arithmos 'number') of Middle English algorism, via Old French from medieval Latin algorismus. The Arabic source, al-Kwarizmi 'the man of Kwarizm' (now Khiva), was a name given to the 9th-cent. mathematician Abu Ja'far Muhammad ibn Musa, author of widely translated works on algebra and arithmetic*

Of course, algorihtms are not restricted only by computer calculations. We perform algorithms in a real life every day when we walk along street or go to university.

Algorithms play a crucial role in Computer Science. Together with rigorous mathematical background there exists something special, which we may cautiously call 'algorithmic area'. It includes at least algorithms development, analysis and implementation; and informally, terms like 'algorithmic thinking'.

On previous courses you already invited with several algorithms. Notice that we work with problems which have some input values and after running of the algorithm we receive some output values as solution.

2 Algorithm Analysis

When we develop algorithms at least we have to think about the following:

- Correctness (that is, we surely get the solution or final state and it is exactly what we need);
- Efficiency (resources: time, memory);

Requirement of the first reason is obvious, everything should work correctly. But why we need the second reason? It arises due to limited resources of our computers.
Let us consider the following example.

Suppose that computer can perform 10^8 standard operations per second [1].
Using this computer we solve the following problems:

1. read from input one billion (10^9) integers;
2. sort these numbers.

First problem will take $10^9/10^8 = 10$ seconds. What about the second? It depends on way of solution. During classes we will learn several algorithms of sorting and looking forward notice that:
Bubble sort algorithm will take about $(10^9)^2$ operations and it gives

$$10^{18}/10^8 = 10^{10} \text{ seconds} \approx 115740 \text{ days} \approx 317 \text{ years!}$$

Working time is so long! Fortunately, there are another algorithms for the same task.
For instance, Quick sort algorithm will take about $10^9 \log 10^9$ operations and in time equivalent

$$10^9 \log 10^9/10^8 = 10 \log 10^9 \approx 300 \text{ seconds.}$$

This example elucidate on the problem of time efficiency in algorithm development. And this happens not only in sorting of data sets, but in another even 'simple' problems.

Let us consider a standard number theoretic question:

Check whether a given number is prime.

One possible implementation of this problem looks as follows:

```
bool isPrime(int p){
  for(int i=2; i <= sqrt(p); i++)
    if (p % i == 0)
      return false;
  return true;
}
```

How many operations this code produce? We have 3 operations in each iteration of loop

```
for(int i=2; i <= sqrt(p); i++)
```

(one for computing `sqrt(p)`; one for check `i <= sqrt(p)`; one for operation `i++`); And the loop itself produces `sqrt(p)` operations. Together with above 3 operations we have one 2 more operations inside the loop:

```
if (p % i == 0)
  return false;
```

So the code

[1] It is almost the same time as our ordinary computer produce

```
bool isPrime(int p){
  for(int i=2; i <= sqrt(p); i++) // 3*sqrt{p} operations
    if (p % i == 0) // 2 operations
      return false; // 1 operation
  return true; // 1 operation
}
```

will finally produce in a worst case at most $2 \times 3 \times \sqrt{p} + 1 = 6\sqrt{p} + 1$ operations.
Now, let us analyze this bound. If number p is less than 10^9 the algorithm will work very fast, namely about 60000 operations which is less than 1 second.

This simple problem today has very interesting and useful applications in cryptography, information security. Real life applications need to check for primality numbers with about 200 digits (number $\approx 10^{200}$). In this case our algorithm will take

$$\approx 6 \times 10^{100} = 6 \times 10^{92} \text{ seconds } = 10^{91} \text{ minutes}$$
$$> 10^{89} \text{ hours } > 10^{87} \text{ days } > 10^{84} \text{ years } >> \text{ Universe life!!!}$$

And for this problem there also exist more fast algorithms than ours. But our goal is to see how the time of performance is important in algorithm development.

Note that together with resource like time there exists another one: memory.

We also use several *data structures*. A data structure is a way to store and organize data in order to facilitate access and modifications. No single data structure works well for all purposes, and so it is important to know the strengths and limitations of several of them. [1]

3 Functions of Algorithm Analysis

To estimate the efficiency of algorithms we will use asymptotic notation. Very often it is what we understand by the term *algorithmic complexity*.

3.1 Big Oh notation

Definition 1. Let $f(n)$ and $g(n)$ be two real-valued functions with positive integer domains. Then we write

$$f(n) = O(g(n)) \text{ or } f = O(g)$$

if there exist positive constants C and N_0 such that

$$f(n) \leq Cg(n)$$

for all positive integers $n \geq N_0$.

From this definition we can see that O-notaion is an upper bound asymptotics of the function. Sometimes if we check whether functions f and g are satisfy $f = O(g)$ it is better to consider the limit

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} =^{?} const.$$

If this limit is equal to constant then $f = O(g)$ otherwise if limite is equal to ∞ then $f \neq O(g)$.

So, all problems are divided by classes with the same O-notation.

Exercise 1. Prove the following formulas:
(a) $n = O(n)$; (b) $2n = O(n)$; (c) $5n+7 = O(2n)$; (d) $n^2/2 = O(n^2)$; (e) $\log n = O((\log n)^2)$; (f) $n^2 + n + 1 = O(n^2)$; (g) $n^2 + n \log n = O(n^3)$.

Exercise 2. Is it true that:
(a) $\sqrt{n} = O(n)$; (b) $n^2 = O(n)$; (c) $\log n = O(n)$; (d) $n \log n = O(n^2)$; (e) $n^2 = O(\log n)$; (f) $2^n = O(n^{100})$; (g) $2^n = O(3^n)$.

Remark that together with O-notation there exists other notations like Θ and Ω.
O-notation shows an upper bound asymptotic; Ω-notation shows a lower bound; and Θ-notaion shows the asymptotic between lower and upper bounds.

3.2 Problem cases: How to analyze?

Further we provide only answers in terms of Big-Oh notation. Try to explain it is true.

Problem 1. Reading from input n integers.

```
int a[100];
int n;
for(int i = 0; i < n; i++)
  cin >> a[i];
```

Complexity is $O(n)$.

Problem 2. Reading from input matrix $n \times n$.

```
int a[100][100];
int n;
for(int i = 0; i < n; i++)
  for(int j = 0; j < n; j++)
    cin >> a[i][j];
```

Complexity is $O(n^2)$.

Problem 3. Bubble sort for n numbers.

```
int a[100];
int n;
void bubble_sort(){
  for(int i = 0; i < n; i++)
    for(int j = i + 1; j < n; j++)
      if (a[i] > a[j]){
        int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
      }
}
```

Complexity is $O(n^2)$.

Problem 4. Calculate maximal power of 2 which divide given number n. For example, 2^3 divide 24 and maximal power of 2 in this case is equal to 3.

```
int n;
int ans = 0;
while (n % 2 == 0){
  ans++;
  n /= 2;
}
cout << ans;
```

Complexity is $O(\log n)$.

Problem 5. Check for primality revisited.

```
bool isPrime(int p){
  for(int i=2; i <= sqrt(p); i++)
    if (p % i == 0)
      return false;
  return true;
}
```

Complexity is $O(\sqrt{n})$.

References

[1] Thomas H. Cormen, Charles E. Leiserson. *Introduction to algorithms – 2-nd edition.* – USA : MIT Press, 2001. – 1180p.