Lecture 24: Algorithmic Complexity

Aims:

- To compute and compare algorithm complexities;
- To discuss polynomial functions.

24.1. Algorithm Complexity Examples

24.1.1. The Prefix Averages Problem

- The problem we'll look at in this section is computing the *prefix averages* of an array of numbers. Given an array $a[1 \dots n]$ of n numbers, we want to compute another array $b[1 \dots n]$ also of length n such that $b[i]$ contains the average of $a[0] \dots a[i]$ (for $0 \leq i \leq n$).

Problem 24.1.

Parameters: *An array $a[1 \dots n]$ of integers.*

Returns: *An array $b[1 \dots n]$ of doubles such that $b[i]$ is the average of $a[1] \dots a[i]$.*

Example

	1	2	3	4
a	3	1	2	6
b	3.0	2.0	2.0	3.0

This is a problem that arises in, e.g., financial software. E.g. given the year-by-year returns of an investment, you might want to plot the average returns over the lifetime of the investment.

24.1.2. Simple Algorithm

- Here is the simplest algorithm.

Algorithm: PREFIXAVERAGES1(a)

```
create array b[1 ... n]
for i := 1 upto n
{    sum := 0.0;
     for j := 1 upto i
     {    sum := sum + a[j];
     }
     b[i] := sum/i;
}
return b;
```

The first line is in English. Beware of such lines! They may hide huge quantities of work. In this case, it's fairly benign: all algorithms that solve this problem will have to contain this piece of work, and it doesn't hide any additions and divisions.

- **Class Exercise**

  Give a formula for $t(n)$, in terms of n, that defines its worst-case time complexity, counting only the additions and divisions used to compute the averages.

24.1.3. A Faster Algorithm

- Consider consecutive averages $b[i-1]$ and $b[i]$. They are similar:

$$
\begin{aligned}
b[i-1] &= (a[1]+a[2]+\ldots+a[i-1]) && /(i-1) \\
b[i] &= (a[1]+a[2]+\ldots+a[i-1]+a[i]) && /i
\end{aligned}
$$

If we keep track of the total so far, we can easily compute the next average:

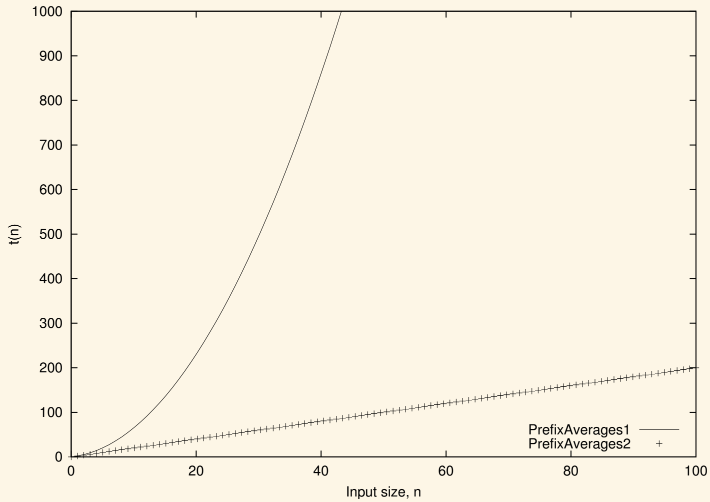
- add the next number to the total
- divide

Algorithm: $\textsc{prefixAverages2}(a)$

```
create array b[1...n]
sum := 0.0;
for i := 1 upto n
{    sum := sum + a[i];
     b[i] := sum/i;
}
return b;
```

- **Class Exercise**

  Give a formula for $t(n)$, in terms of n, that defines its worst-case time complexity, again counting only the additions and divisions.


t(n)
Input size, n
PrefixAverages1
PrefixAverages2
0
100
200
300
400
500
600
700
800
900
1000
20
40
60
80

24.2. Polynomials Functions

- In algebra, a *polynomial function*, or *polynomial*, is a function of the form:

$$f(n) =_{\text{def}} a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$$

  - k is a nonnegative integer;
  - for us, $a_0, \ldots, a_k$ will be integers, called the *coefficients* of f;
  - the highest occurring power of n (i.e. k if a_k is not zero) is called the *degree* of f;
  - its coefficient, a_k, is called the *leading coefficient*;
  - each summand, of the form $a_i x^i$, is called a *term*.

- Polynomials of
  - degree 0 are called *constant functions*, e.g.:

$$f(n) =_{\text{def}} 3$$

  - degree 1 are called *linear functions*, e.g.:

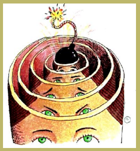
$$f(n) =_{\text{def}} 2n + 3$$

  - degree 2 are called *quadratic functions*, e.g.:

$$f(n) =_{\text{def}} 7n^2 + 2n + 3$$

  - degree 3 are called *cubic functions*, e.g.:

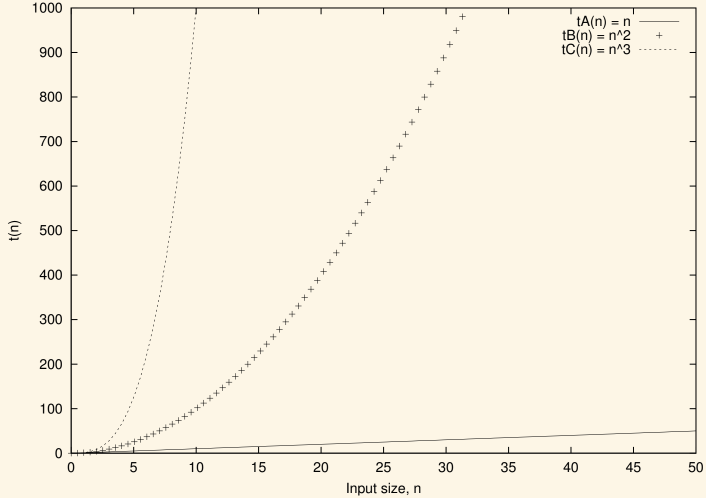
$$f(n) =_{\text{def}} n^3 + 3$$



- A *root* of a polynomial $f(n)$ is a number r such that $f(r) = 0$
- To determine the roots of polynomials, i.e. to 'solve algebraic equations':
  - approximate them using, e.g., Newton's method
  - use a formula, where known, e.g. the quadratic formula for quadratic equation $an^2 + bn + c = 0$

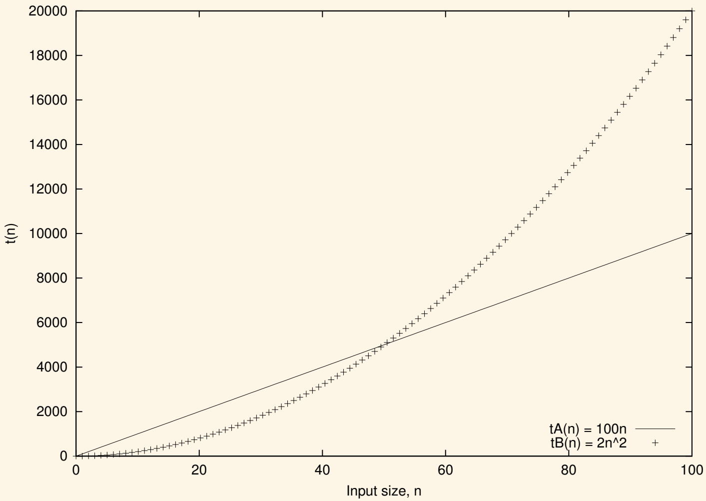
$$n = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Here are the graphs of a few very simple polynomials.



- In general, of course, coefficients will not always be one or zero, as they were in the previous graph. So let's see what happens when we have some slightly more interest-

ing polynomials. Suppose algorithm A's worst-case time complexity $t_A(n) =_{\text{def}} 100n$, and algorithm B's worst-case time complexity $t_B(n) =_{\text{def}} 2n^2$.



The lines cross at $n = 50$. So when inputs of are of size less than 50, algorithm B is the faster; when input sizes exceed 50, algorithm A is the faster. What's more, from that point on, the larger the input, the bigger the advantage A has over B. If $n = 100$, A is twice as fast as B; if $n = 1000$, A is 20 times as fast.

- Those of you who have a bit of algebra under your belts can compare algorithms without graphing them.
- To find out where the graphs for A and B cross, solve

$$100n = 2n^2$$

Rearranging:

$$2n^2 - 100n = 0$$

To solve $an^2 + bn + c$, we can use

$$n = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In this case,

$$n = \frac{-(-100) \pm \sqrt{-100^2 - 4 \times 2 \times 0}}{2 \times 2}$$

So $n = 0$ and $n = 50$ are the roots of this equation.

- We can find out the maximum input size that can be handled within a certain time period.
  - To find out how much work A can do in 1000 milliseconds (1 second), assuming each operation takes 1 millisecond, solve

    $$100n = 1000$$

    So $n = 10$.
  - To find out how much B can do in 1000 milliseconds, solve

    $$2n^2 = 1000$$

    You would use the quadratic formula again from above

    $$n = \frac{-0 \pm \sqrt{0^2 - 4 \times 2 \times -1000}}{2 \times 2}$$

    This has only one positive solution, $n = 22.36$. So we can solve instances of B up to size 22 in 1000 milliseconds.

Time available (seconds)	Maximum problem size solvable with A	Maximum problem size solvable with B
1	10	22
10	100	70
100	1000	223
1000	10000	707

- Obviously, the algebra gets harder if the functions that describe the complexities of the algorithms are not as simple as these ones.

Acknowledgements

The prefix averages problem and its two algorithms come from [GT02]. Some of the discussion of polynomial functions is based on [AU92].

Clip Art (of head with bomb) licensed from the Clip Art Gallery on DiscoverySchool.com.

References

[AU92] A. V. Aho and J. D. Ullman. *Foundations of Computer Science*. W.H. Freeman, 1992.

[GT02] M. T. Goodrich and R. Tamassia. *Algorithm Design: Foundations, Analysis, and Internet Examples*. Wiley, 2002.