Introduction

Abhiram Ranade

January 7, 2016

Administrative issues

Attendance is compulsory:

- If you miss a lecture it is your responsibility to find out what happened.
- Important announcements may be made on moodle, but this cannot be guaranteed.
- But no DX grade will be given.

Grading Scheme:

- 4 quizzes, each worth 10 %.
- Homeworks, totally worth 10 %. Best $n-1$ out of n.
  Discussion OK, writing must be independent.
- Endsem, worth 50 %.

Textbooks: Any of

- CLRS, Kleinberg-Tardos, Dasgupta-Papadimitriou-Vazirani.

Course goals

Analysis of algorithms: Given an algorithm to solve a problem, determine how good it is.

- Good = Fast. Or requiring small amount of memory
- Fast = ? Need to define carefully.

Techniques for designing good algorithms

Techniques for showing that for some problems it is difficult to design good algorithms

Outline for today

- Some important prerequisite skills
- The framework for algorithm analysis

Some important prerequisite skills

On reading an algorithm design problem be able to clearly state:

- What is the input
- What is the output
- What conditions the output must satisfy in order to be correct.
- Define suitable notation for all this.
- Construct examples of input and output

Understand iteration, recursion, basic data structures (CS 213).

Use the language of sequences, graphs, partial orders as needed (CS 207)

Know the distinction between problem and instance:

- "The shortest path problem" and also "The problem of finding the shortest path from IIT to TIFR"?
- Instance = Specific values that the input can take.
- Problem = Relation from Instances to desired output values.

More prerequisite skills

Be able to describe an algorithm without having to write a program.

- "j = smallest such that A[j] > 100." rather than loop.
- Separate data structure aspects from algorithmic aspect. "Find the smallest element in the set A" + "Store A as a priority queue" rather than give code for the data structure manipulation.
- Describe what you compute very clearly, how you do it will then be easier to understand.
- If your algorithm needs a function, clearly specify its arguments, and what it computes.
- Clearly define the inputs/outputs to your algorithm. Give good names.
- Do not hesitate in defining notation. Do not say "For the smallest index having ... do ... and then for the previously computed index do ...". If the same object is being referred to again, please give it a name.

A framework for algorithm analysis:

"The time taken by an algorithm": Should we write a program and measure its time? Time will depend upon:

- Programming language, the compiler optimization level used.
- Clock speed and architectural parameters of the machine.
- The specific instance given as input.

An observation: Time will usually be larger for larger inputs.

- Sorting will take longer for larger number of keys.
- DFS/BFS will require more time on larger graphs.

Convention 1: We will specify the time taken by an algorithm as a function of the *size* of the input.

Official definition: size = Number of bits needed to specify input.

Common usage: Any parameter from which you can calculate the size of the input in bits. e.g. number of keys for sorting.

The time taken by an algorithm (contd)

The time will be different for different instances of the same size.

- Insertion sort: time is much less if the instance is already sorted.
- GCD: Only one recursive call suffices if x divides y in instance (x, y).

Convention 2: Time required by an algorithm for size $n \equiv$ Maximum among the times taken for all instances of size n.

"Worst case analysis"

- Seems pessimistic, but anything else is very hard to evaluate.
- At least we can give solid guarantees. "Time will be at most ...".
- Often: worst case time is a good representative of all times.
- Sometimes worst case is indeed too pessimistic. For such cases, we may consider "average case".

Simplified Mathematical model of execution:

RAM : "Random Access Machine". processor + memory.

- T[Arithmetic operations including comparisons] = 1.
- T[Copying data to/from memory (assignment)] = 1.
- After evaluating the condition, an if statement must decide what to execute next. T[control transfer] = 1.
- After evaluating the condition, a while statement must decide whether to terminate the loop. T[control transfer] = 1.
- T[Function calls] = 1 for control transfer + time to copy the parameter references.
- T[Return] = 1 for control transfer + time to copy result if any.
- Other statements: you should be able to guess.

"Unit time if to amount of data involved = constant"
Constant = does not depend upon problem size

Critique of the RAM model

Very simplistic. e.g. On most real computers,
T[loading data from memory] $\approx$ 10 * T[addition].

Key idea: We use the RAM model for simplicity, but while drawing conclusions, keep its drawbacks in mind.

Analyzing algorithms using the RAM: 1

Suppose we determined that the worst case time for some algorithm A to be $= 5n^2 + 16n + 29$. $n =$problem (instance) size.

What can we conclude using this for a more accurate model, say one in which copy to memory takes 10 steps rather than 1?

Observation: The coefficients of the different terms will change, but not the functional dependence on n.

Conclusion: Even if the times for the different operations are different, the time taken by A must have the form $an^2 + bn + c$, where a, b, c are constants.

Will it help us to know that A takes time $an^2 + bn + c$ without knowing a, b, c?

Asymptotics

Suppose algorithm A takes time $an^2 + bn + c$ and algorithm A' takes time $a'n + b'$. Which one should we say is better?

Assumption: We use computers to solve large problems, i.e. only consider the case $n \to \infty$.

$$T(A) = an^2 + bn + c \geq an^2$$
$$T(A') = a'n + b' \leq (a' + b')n$$
$$n > (a' + b')/a \quad \Rightarrow \quad T(A) \geq an^2 > (a' + b')n \geq T(A')$$

Thus for values of n of our interest Algorithm A' is better.
This is true no matter what the values of a, b, c, a', b' are.

Key idea: A quadratic function will eventually become bigger than a linear function.
More generally, a function with a bigger leading term will eventually rise above one with a smaller term.

Analysis framework: conclusion

For any algorithm we will report the leading term (without its coefficient) of the function denoting the time required by the algorithm on the worst instance of size *n*.

If the leading term of one algorithm is smaller than the leading term of another, we will say that the former is better.

If the leading terms of two algorithms are the same, then we cannot distinguish between the algorithms.

For doing this analysis, we will use the RAM model. However, since we only want the leading term, our analysis will often be further simplified.

Good news: Our analysis does not depend upon the machine, programming language.

Bad news: Our analysis is coarse, and only asymptotic.

"Leading term" seems awkward. Better notation possible?

The theta (θ) notation

Informal: If the leading term of a function is t, then the function is said to be in class $\theta(t)$.

Definition: Suppose f, g are functions. Then f is said to belong to class $\theta(g)$ if there exist real constants $c_1, c_2, n_0 > 0$ such that for all $n \geq n_0$ we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$.

Example: Let $f(n) = 5n^2 + 16n + 29$, $g(n) = n^2$. Then $f \in \theta(g)$
Proof: $c_1 = 1$, $c_2 = 5 + 16 + 29$, $n_0 = 1$ suffice.

The notation does not have anything to do with running times of algorithms. We may, for example, write $\binom{n}{2} \in \theta(n^2)$.
Proof: $n_0 = 2$, $c_1 = 1/4$, $c_2 = 1$.

The notation is only used to say that we only know the leading term and nothing more in a function.

Common (ab)use of notation

What do we mean when we say "A rose is a flower"?

We mean: "A rose belongs to the set of flowers."

Common abuse: "is" is often used to mean "belongs to the set of".

Likewise, we will write $f = \theta(g)$ instead of the more accurate $f \in \theta(g)$.

Example: Multiplying $n \times n$ matrices

```
double a[n][n], b[n][n], c[n][n];
...
for(int i=0; i<n; i++)
  for(int j=0; j<n; j++){
    c[i][j] = 0;
    for(int k=0; k<n; k++)
       c[i][j] += a[i][k]*b[k][j];
    }
```

Each loop executes n times and hence the statement inside the innermost loop will execute n^3 times.
Statement in the innermost loop will use time cn^3 for some c.

Other statements will contribute $dn^2 + en + f$ for some d, e, f.

Thus $T(n) = cn^3 + dn^2 + en + f$.
Thus $T(n) = \theta(n^3)$. Same for all cases including worst

Example: Merging sequences A, B of length n

1. Set $C =$ empty sequence.
2. While both A, B have at least one element, move the smaller of the elements at the heads of A, B to the end of C.
3. If A contains elements, move them in order to the end of C.
4. If B contains elements, move them in order to the end of C.

For any execution:
Stmt 2 executes at least n times and at most $2n$ times.
Each execution of stmt 2 requires some d_2 steps
$d_2 n \leq$ Time spent in executions of stmt 2 $\leq d_2 \cdot 2n$.

Number of times stmts 3,4 execute: at most n times.
Time spent in execution of 3, 4 is $\leq d_3 n + d_4 n$.

$d_2 n \leq$ Total time $\leq n(2d_2 + d_3 + d_4)$.
Thus time for any execution $= \theta(n)$.

We proved both upper and lower bounds.

"Worst case time" $= \theta(n)$

Example: Multiplying n bit/digit numbers

Read bits/digits into an array, mimic primary school algorithm.

Time $= \theta(n^2)$ Homework.

$\theta(n^2)$ time also for finding remainder, quotient.

Algebra of θ

Suppose $h(n) = f(n) + g(n)$, and $f = \theta(n)$, and $g = \theta(n^2)$. Then $h(n) = \theta(n^2)$

Proof:

We know $\exists c_1, c_2, n_1, d_1, d_2, n_2$ such that

$c_1 n \leq f(n) \leq c_2 n$ for $n \geq n_1$

$d_1 n^2 \leq g(n) \leq d_2 n^2$ for $n \geq n_2$.

Let $n_3 = \max(n_1, n_2)$. Then for $n \geq n_3$ we have

$h(n) = f(n) + g(n) \geq c_1 n + d_1 n^2 \geq d_1 n^2$

$h(n) = f(n) + g(n) \leq c_2 n + d_2 n^2 \leq (c_2 + d_2) n^2$.

So $e_1 n^2 \leq h(n) \leq e_2 n^2$ for $n \leq n_3$, $e_1 = d_1, e_2 = c_2 + d_2$.

So $h(n) = \theta(n^2)$

We will handle this intuitively, $\theta \equiv$ most significant term

Simplify expressions inside $\theta(\cdot)$: $\theta(5n^3 + 7n) = \theta(n^3)$

Problems defined by two size parameters

It is more natural to ask: how much time does it take for merging a length m sequence and an n length sequence.

"Problem size" : the pair m,n.

We need to generalize θ when there are two sizes.

Definition: Let f, g be functions of two variables. Then $f \in \theta(g)$ if there exist real numbers $c_1, c_2, n_0, m_0 > 0$ such that $0 \leq c_1 g(m,n) \leq f(m,n) \leq c_2 g(m,n)$, provided $m \geq m_0$ and $n \geq n_0$.

Exercise: Estimate the time taken for multiplying a m digit number by a n digit number. Show that it is in $\theta(mn)$.

Exercise: Same for division, i.e. computing quotient and remainder.

Exercise: Suppose that for integer multiplication as the size parameter we use the total number of bits in the input. Show that the running time is $\theta(n^2)$.

Summary

- "Time taken by an algorithm" = "Time taken by an algorithm in the worst case as a function of problem size"
- Assume program runs on RAM, but pay attention only to the most significant term in the analysis.
- General goal: Try to express time taken by an algorithm as $\theta(\ldots)$.
- θ includes upper bound as well as lower bound.