EECS 3101 A: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101A`
Also on Moodle

Careful Running Time Analysis of Algorithms

- Use the “RAM” model
- Follows the example in the text on page 26
- Precise counting of the computational cost (e.g. running time) of each line of pseudocode

Analysis of FindMax

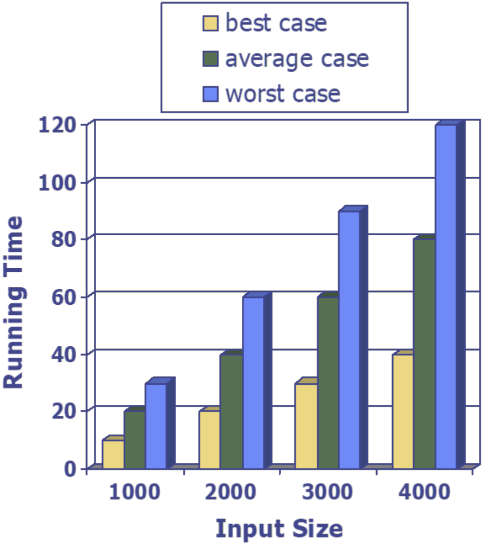
FIND-MAX(A)

```
1  max = A[1]
2  for j = 2 to A.length
3          if max < A[j]
4          max = A[j]
5  return max
```

line	Cost	Times
1	c_1	1
2	c_2	n
3	c_3	$n-1$
4	c_4	$0 \le k \le n-1$
5	c_5	1

Best Case: $k = 0$
Worst Case: $k = n-1$
Average Case: ?

Best/Worst/Average Case Analysis



Best/Worst/Average Case Analysis - 2

The running time of an algorithm typically grows with the input size.

- Best Case: Not very informative
- Average Case: Often very useful, but hard to determine
- Worst Case: Easier to analyze. Crucial in applications like
  - Games
  - Finance
  - Robotics

Analysis of FindMax - Continued

```
Find-max(A)
1  max = A[1]
2  for j = 2 to A.length
3        if max < A[j]
4        max = A[j]
5  return max
```

line	Cost	Times
1	c_1	1
2	c_2	n
3	c_3	$n-1$
4	c_4	$0 \leq k \leq n-1$
5	c_5	1

Running time (worst-case): $c_1 + c_5 - c_3 - c_4 + (c_2 + c_3 + c_4)n$
Running time (best-case): $c_1 + c_5 - c_3 + (c_2 + c_3)n$

Simplifying Running Times

Note that the worst-case time of $c_1 + c_5 - c_3 - c_4 + (c_2 + c_3 + c_4)n$ is

- Complex
- Not useful as the c_i's are machine dependent

A simpler expression: $C + Dn$ [still complex].

Want to say this is Linear, i.e., $\approx n$

Q: How/why can we throw away the coefficient D and the lower order term C?

Simplifying Running Times - Rationale

- Discarding lower order terms: We are interested in large n – cleaner theory, usually realistic.
- Discarding coefficients (multiplicative constants): the coefficients are machine dependent

Caveat: remember these assumptions when interpreting results! We will not get:

- Exact run times
- Comparison for small instances
- Small differences in performance

Analysis of FindMax - Summary

- Last expression: $C + Dn$ written as $\Theta(n)$
- Also called linear time
- Question: Can we do better?

Later:
Lower Bounds: We will show that for any algorithm for this problem, for each $n > 0$, there exists an input that make the algorithm take $\Omega(n)$ time

Another problem

The i^{th} prefix average of an array X is the average of the first $i+1$ elements of X:

$$A[i] = (X[0] + X[1] + \ldots + X[i])/(i+1)$$

We will look at 2 implementations.

A Slower Algorithm

```
/** Returns an array a such that, for all j, a[j] equals the average of x[0], ..., x[j]. *
public static double[ ] prefixAverage1(double[ ] x) {
  int n = x.length;
  double[ ] a = new double[n];                     // filled with zeros by default
  for (int j=0; j < n; j++) {
    double total = 0;                              // begin computing x[0] + ... + x[
    for (int i=0; i <= j; i++)
      total += x[i];
    a[j] = total / (j+1);                          // record the average
  }
  return a;
}
```

Good example for determining the running time

Analysis

- Outer loop iterates for $j = 0, \ldots, n-1$
- Inner loop iterates for $i = 0, \ldots, j$
- The loop body takes $\Theta(1)$ steps

Analysis - 2

The easiest way to sum the running time is

$$
\begin{aligned}
T(n) &= \sum_{j=0}^{n-1} \sum_{i=0}^{j} 1 \\
&= \sum_{j=0}^{n-1} (j+1) \\
&= \sum_{j=1}^{n} j \\
&= n(n+1)/2
\end{aligned}
$$

So $T(n) \in \Theta(n^2)$

A Faster Algorithm

```
/** Returns an array a such that, for all j, a[j] equals the average of x[0], ..., x[j]. *
public static double[ ] prefixAverage2(double[ ] x) {
  int n = x.length;
  double[ ] a = new double[n];              // filled with zeros by default
  double total = 0;                         // compute prefix sum as x[0] + x[1] + .
  for (int j=0; j < n; j++) {
    total += x[j];                          // update prefix sum to include x[j]
    a[j] = total / (j+1);                   // compute average based on current sur
  }
  return a;
}
```

Analysis: Linear time $\Theta(n)$

More practice

PoA:

- 280
- 283
- 264