MA 515: Introduction to Algorithms & MA353 : Design and Analysis of Algorithms [3-0-0-6]

Lecture 2

http://www.iitg.ernet.in/psm/indexing_ma353/y09/index.html

Partha Sarathi Mandal

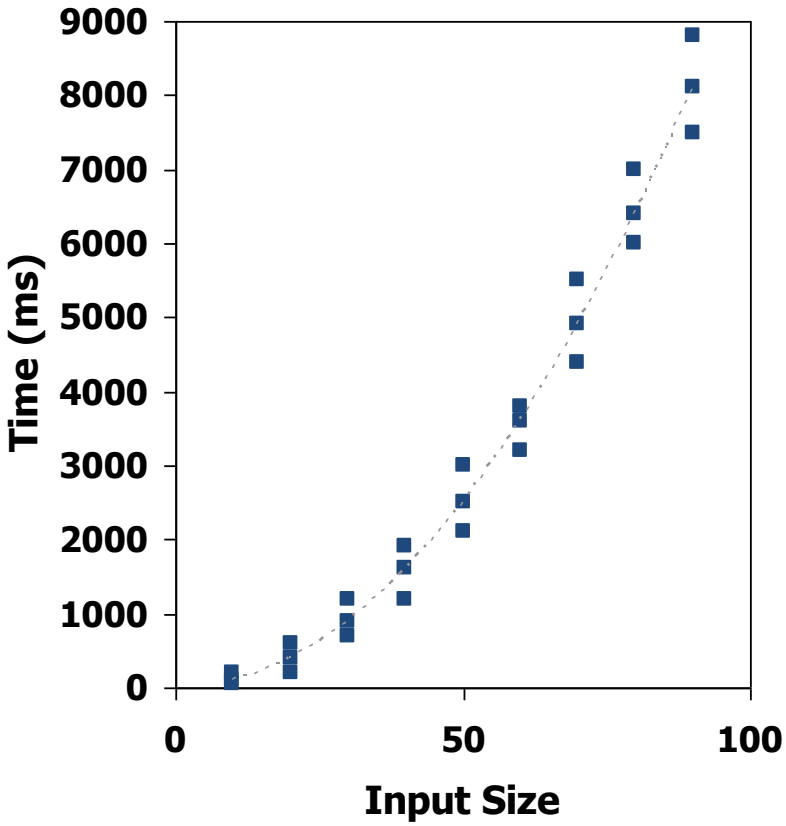
psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55
Class Room : 2101

Experimental Studies

- Write a program implementing the algorithm.
- Run the program with inputs of varying size and composition.
- Use a method like System.currentTimeMillis() / time_t to get an accurate measure of the actual running time.
- Plot the results.



Limitations of Experiments

- It is necessary to implement the algorithm, which may be difficult.
- Results may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments must be used.

Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation.
- Characterizes running time as a function of the input size, n.
- Takes into account all possible inputs.
- Allows us to evaluate the speed of a algorithm independent of the hardware/software environment.

Pseudocode

- High-level description of an algorithm.
- More structured than English prose.
- Less detailed than a program.
- Preferred notation for describing algorithms.
- Hides program design issues.

Example:
Find max element of an array:

Algorithm ***arrayMax(A, n)***
Input array *A of n integers*
Output maximum element of *A*

```
currentMax <- A[0]
for i  <- 1 to n - 1 do
   if A[i] > currentMax then
            currentMax <- A[i]
return currentMax
```

Pseudocode Details

- Control flow
  - **if … then … [else …]**
  - **while … do…**
  - **repeat… until…**
  - **for … do…**
- Indentation replaces braces
- Method declaration
- Method call
- Return value
- Expression
  - assignment
  - equality testing etc.

Example:
Find max element of an array:

Algorithm *arrayMax(A, n)*
Input array *A of n integers*
Output maximum element of *A*

```
currentMax <- A[0]
for i  <- 1 to n - 1 do
   if A[i] > currentMax then
            currentMax <- A[i]
return currentMax
```

The Random Access Machine (RAM) Model

- Machine-independent algorithm design depends upon a hypothetical computer called the *Random Access Machine* or RAM.
  - Each ``simple'' operation (+, *, -, =, if, call) takes exactly 1 time step.
  - Loops and subroutines are *not* considered simple operations. Instead, they are the composition of many single-step operations.
  - Each memory access takes exactly one time step, and we have as much memory as we need.
  - The RAM model takes no notice of whether an item is in cache or on the disk, which simplifies the analysis.

Primitive Operations

- Examples:
  - Evaluating an expression
  - Assigning a value to a variable
  - Indexing into an array
  - Calling a method
  - Returning from a method

- Basic computations performed by an algorithm
- Identifiable in pseudocode
- Largely independent from the programming language
- Exact definition not important
- Assumed to take a constant amount of time in the RAM model

Counting Primitive Operations

- By inspecting the pseudocode, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size.

Algorithm *arrayMax*(*A*, *n*)	# operations
currentMax ← *A*[0]	2
for (i =1; i<n; i++)	2*n*
(i=1 once, i<n n times, i++ (n-1) times)	
if *A*[*i*] > *currentMax* **then**	2(*n* – 1)
currentMax ← *A*[*i*]	2(*n* – 1)
return *currentMax*	1
Total	6*n* –1

Estimating Running Time

- Algorithm ***arrayMax*** executes 6n-1 primitive operations in the worst case.

Define:

- a = Time taken by the fastest primitive operation
- b = Time taken by the slowest primitive operation
- Let ***T(n)*** be worst-case time of ***arrayMax.***

  Then ***a (6n-1) <= T(n) <= b (6n-1)***
- Hence, the running time ***T(n)*** is bounded by two linear functions.

Growth Rate of Running Time

- Changing the hardware/ software environment
  - Affects ***T(n)*** by a constant factor, but
  - Does not alter the growth rate of ***T(n)***
- The linear growth rate of the running time ***T(n)*** is an intrinsic/basic property of algorithm ***arrayMax***

The Growth Rate of the Six Popular functions

n	log$n$	n	$n$log$n$	n^2	n^3	2^n
4	2	4	8	16	64	16
8	3	8	24	64	512	256
16	4	16	64	256	4,096	65,536
32	5	32	160	1,024	32,768	4,294,967,296
64	6	64	384	4,094	262,144	$1.84 * 10^{19}$
128	7	128	896	16,384	2,097,152	$3.40 * 10^{38}$
256	8	256	2,048	65,536	16,777,216	$1.15 * 10^{77}$
512	9	512	4,608	262,144	134,217,728	$1.34 * 10^{154}$
1024	10	1,024	10,240	1,048,576	1,073,741,824	$1.79 * 10^{308}$