Data Structures and Algorithm Analysis

11

Dr. Syed Asim Jalal
Department of Computer Science
University of Peshawar

Analysis of Algorithm

- Measuring algorithms in terms of the amount of computational resources that the algorithms requires
  - Running time: How much time is taken to complete the algorithm execution?
  - Storage requirement: How much memory is required to execute the program?
- In this course we mostly study *Running Time*.

Random-Access Machine (RAM) model

- For algorithm analysis, we shall assume a generic one-processor, *random-access machine (RAM)* model of computation as our implementation technology and understand that our algorithms will be implemented as computer programs on this machine.
  - It is a standard computational model of generic processor
  - It gives platform-independence: i.e. the analysis will not depend on any specific architecture

RAM Model Characteristics

1. Instructions are executed sequential, one-after another
2. Entire memory is equally expensive to access
   - All memory locations accessible in the same constant time. (nearest or furthest)
3. No concurrent operations
4. All basic instructions take unit time
5. Constant word size

RAM basic operations

- Assignment operation
- Basic arithmetic operation
  - +, -, *, /, MOD, floor, ceiling
- Comparison or Boolean operations.
- Branching Instructions, Subroutine Instructions

- No complicated Instructions in RAM
  - Such SORT, SEARCH etc

Running Time

- Different Algorithms for the same problem can have different “Running Times”
- Different inputs of the same size may result in different running times
- Running Time Analysis is based on count of the number of the primitive operations to be executed
- Algorithm analysis does not give us the exact running time in seconds. As the execution time duration is dependent on the machine to be used. We only model Time in terms of input size.

Running Time Representation

- We represent running time as a function of the input size ‘N’ as T(N)
  - $O(3N-7)$, $O(2\ N^2)$, $O(N^3 + N^2)$
- ‘N’ represents input data size
  - It represents different features in different problems, image size, data size, text size etc.
- We compare “running time” of different algorithms in terms of these Running Time Representations.

How is any algorithm analyzed?

- Algorithms can be analyzed in two main ways.
  - Experimental analysis
  - Theoretical Analysis

Experimental analysis

- Method
  - Write a program implementing the algorithm
  - Run the program with inputs of varying size and composition
  - Get an accurate measure of the actual running time
  - Plot the results

- Short comings
  - Implementing the algorithm may be difficult and Time consuming
  - Results may not be indicative of the running time on All Inputs not included in the experiment.
  - In order to compare two algorithms, the same hardware and software environments must be used

Theoretical Analysis

- Method
  - Uses description of the algorithm instead of an implementation
  - Characterizes running time as a mathematical function of the input size n.
  - Takes into account all possible inputs
    - ✓ Best and Worst cases
  - Allows us to evaluate an algorithm independent of the hardware/software environment (uses RAM model)
  - Uses Mathematics

Worst-Case and Best-Case Analysis

- ***Worst-Case Analysis***
  - The maximum amount of time that an algorithm require to solve a problem of size n.
  - This gives an upper bound for the time complexity of an algorithm.
  - Normally, we try to find worst-case behavior of an algorithm.

- ***Best-Case Analysis***
  - The minimum amount of time that an algorithm require to solve a problem of size n.
  - The best case behavior may be very different from worst or average case so it is not very useful to consider this all the time.

- ***Average-Case Analysis***
  - The average amount of time that an algorithm require to solve a problem of size n.
  - Sometimes, it is difficult to find the average-case behavior of an algorithm.
  - We have to look at all possible data organizations of a given size n, and their distribution probabilities.
  - ***Worst-case analysis is more common than average-case analysis.***

Running Time T(N)

- Running time expression is the number of primitive operations (steps) executed.
  - We assume RAM model for this purpose and mostly get a mathematical expression.
  - For example, Running Time = 8n+9

$$\sum_{\text{all statements}} (\text{cost of statement}) \cdot (\text{number of times statement is executed})$$

Running Time Analysis: *Example 1*

Calculate running time, counting basic operations

Input: integer n

Output: Sum of all numbers up to n

```
int sum( int n )
{
        int partialSum;

        partialSum = 0;
        for (int i = 1; i <= n; i++)
                partialSum += i * i * i;
        return partialSum;
}
```

`int partialSum;`	no time
`partialSum = 0;`	1
`for (int i = 1; i <= n; i++)`	1 + (N+1) + N
`partialSum += i * i * i;`	N * 4
`return partialSum;`	1

Complexity function:

T(N) = 1 + 1 + (N+1) + N + N*(4) + 1 = 6N + 4

So our running time estimate is order of N i.e. **O(N).**

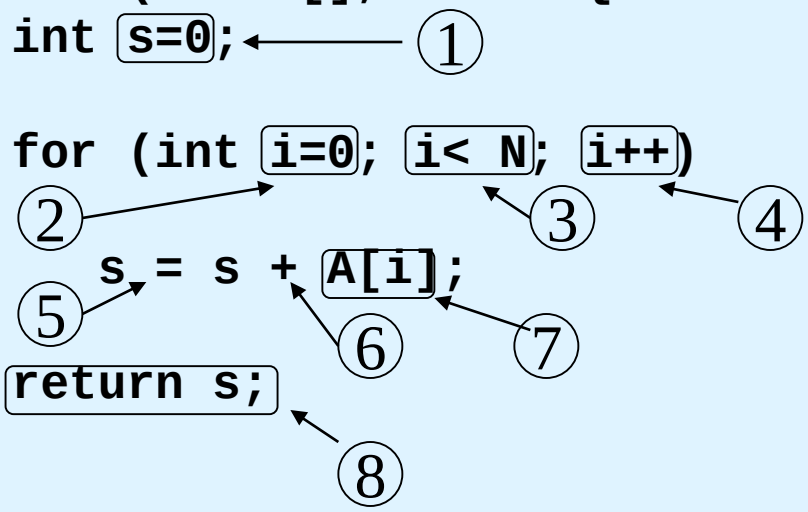
```
// Input: int A[N], array of N integers
// Output: Sum of all numbers in array A

int Sum(int A[], int N {
    int s=0;

    for (int i=0; i< N; i++)

        s = s + A[i];

    return s;
}
```



1,2,8: executed only one time.

3,4,5,6,7: Once per each iteration of for loop, N iteration

Total: 5N + 3

The *complexity function* of the algorithm is : *f(N) = 5N +3*

Order of Growth is: N, Linear

How 5N+3 Grows with input size???

Estimated running time for different values of N:

N = 10	=> 53 steps
N = 100	=> 503 steps
N = 1,000	=> 5003 steps
N = 1,000,000	=> 5,000,003 steps

As N grows, the number of steps grow in *linear* proportion to
N for this `Sum` function.

This represents the very basics of Algorithm analysis.

Example 3:

Algorithm *arrayMax*(*A*, *n*)	# operations
currentMax ← ***A***[0]	2(1)
for i ← 1 **to** i < n **do**	1(1) + 1(***n***)
if A[i] > currentMax **then**	2(***n*** – 1)
currentMax ← A[i]	2(***n*** – 1)
i ← i+1	2(***n*** – 1)
return currentMax	1(1)
Total	7***n*** – 2

Running Time …in a simple way

- To simplify the process, the running time is calculated on the basis of every statement instead of primitive operations involved in each expression.
  - Because every statement *i* is executed in a constant time C_i each time
  - The impact of number of operations in each statement is therefore constant is negligible.

Example: Simple Loop

	Cost	Times
`i = 1;`	c1	1
`sum = 0;`	c2	1
`while (i <= n) {`	c3	n+1
`  i = i + 1;`	c4	n
`  sum = sum + i;`	c5	n
`}`		

Total Cost = c1 + c2 + (n+1)*c3 + n*c4 + n*c5

= c1+c2+c3n+c3+c4n+c5n = **C6 + C7n**

➔ The time required for this algorithm is proportional to n

Example: Simple If-Statement

	Cost	Times
`if (n < 0)`	c1	1
`  val = -n`	c2	1
`else`		
`  val = n;`	c3	1

Total Cost = c1 + max(c2,c3)

Example: Nested Loop

	Cost	Times
`i=1;`	c1	1
`sum = 0;`	c2	1
`while (i <= n) {`	c3	n+1
`  j=1;`	c4	n
`  while (j <= n) {`	c5	n*(n+1)
`  sum = sum + i;`	c6	n*n
`  j = j + 1;`	c7	n*n
`  }`		
`  i = i +1;`	c8	n
`}`		

Total Cost = c1 + c2 + (n+1)*c3 + n*c4 + n*(n+1)*c5+n*n*c6+n*n*c7+n*c8

➔ The time required for this algorithm is proportional to n^2 or Order of n^2

Order of Growth: (Growth Rate)

- It is an abstraction to ease the comparison of algorithms. In growth rate we measure an algorithm's time requirement as a function of the *problem size 'n'*.

- *For instance, we say that for the problem size n*
  - Algorithm A requires **$5*n^2$** time units
  - Algorithm B requires **100*n** time units.

- The **most important factor to know** is how quickly the algorithm's time requirement grows as a function of the problem size.
  - Algorithm A requires time proportional to **n^2**.
  - Algorithm B requires time proportional to **n**

- An algorithm's proportional time requirement is known as ***Growth Rate***. We can compare the efficiency of two algorithms by comparing their **Growth Rates** only.

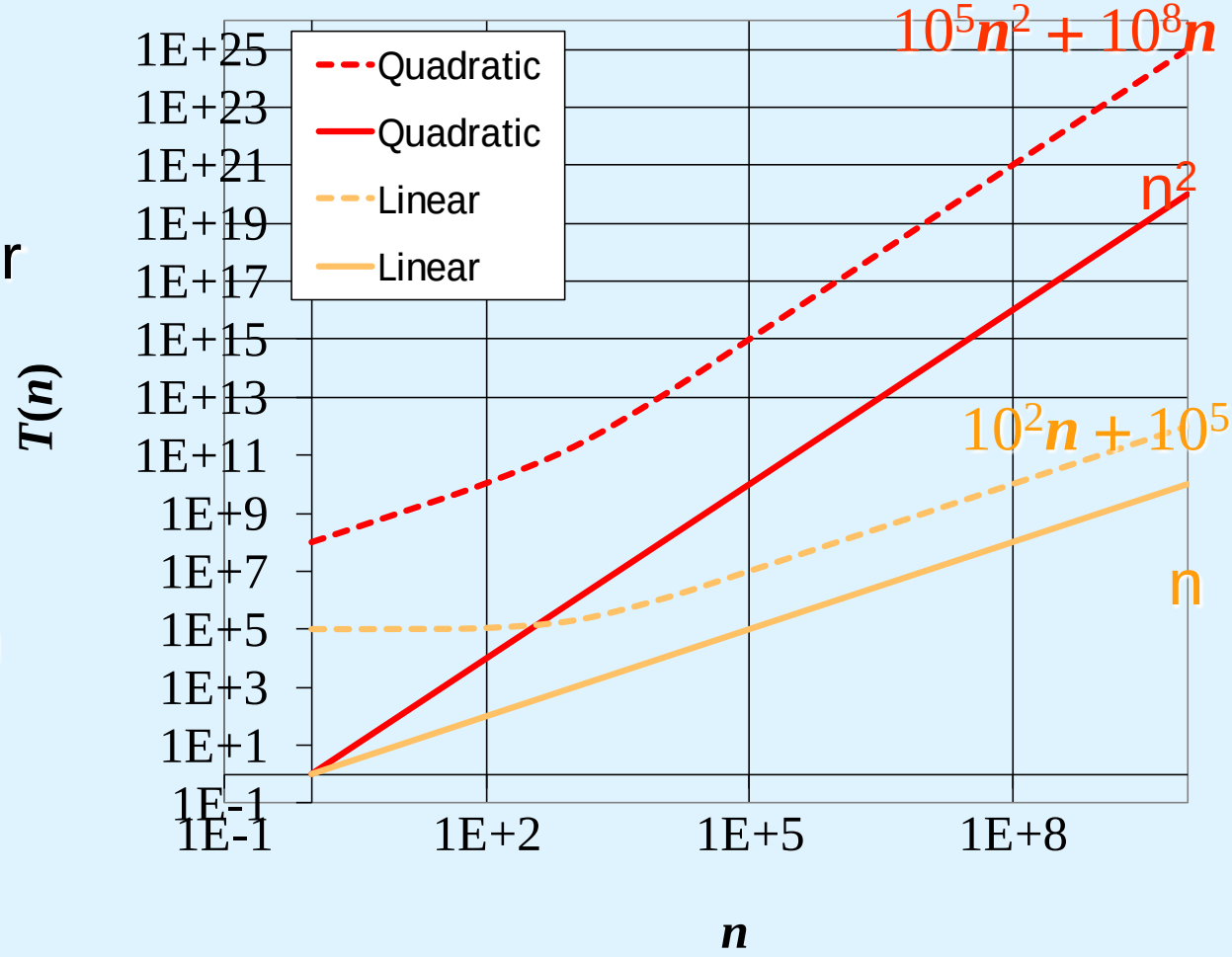
  - To find growth rate we will look only at the leading term of the running time.
  - We get the growth rate from T(N) by
    - dropping lower-order terms.
    - By ignoring the constant coefficient in the leading term.
    - So $10^5n^2 + 10^8n$ becomes n^2

- The growth rate is not affected for large n by constant factors and lower-order terms

Examples:

Running times of $10^2n + 10^5$ is a linear function (n)

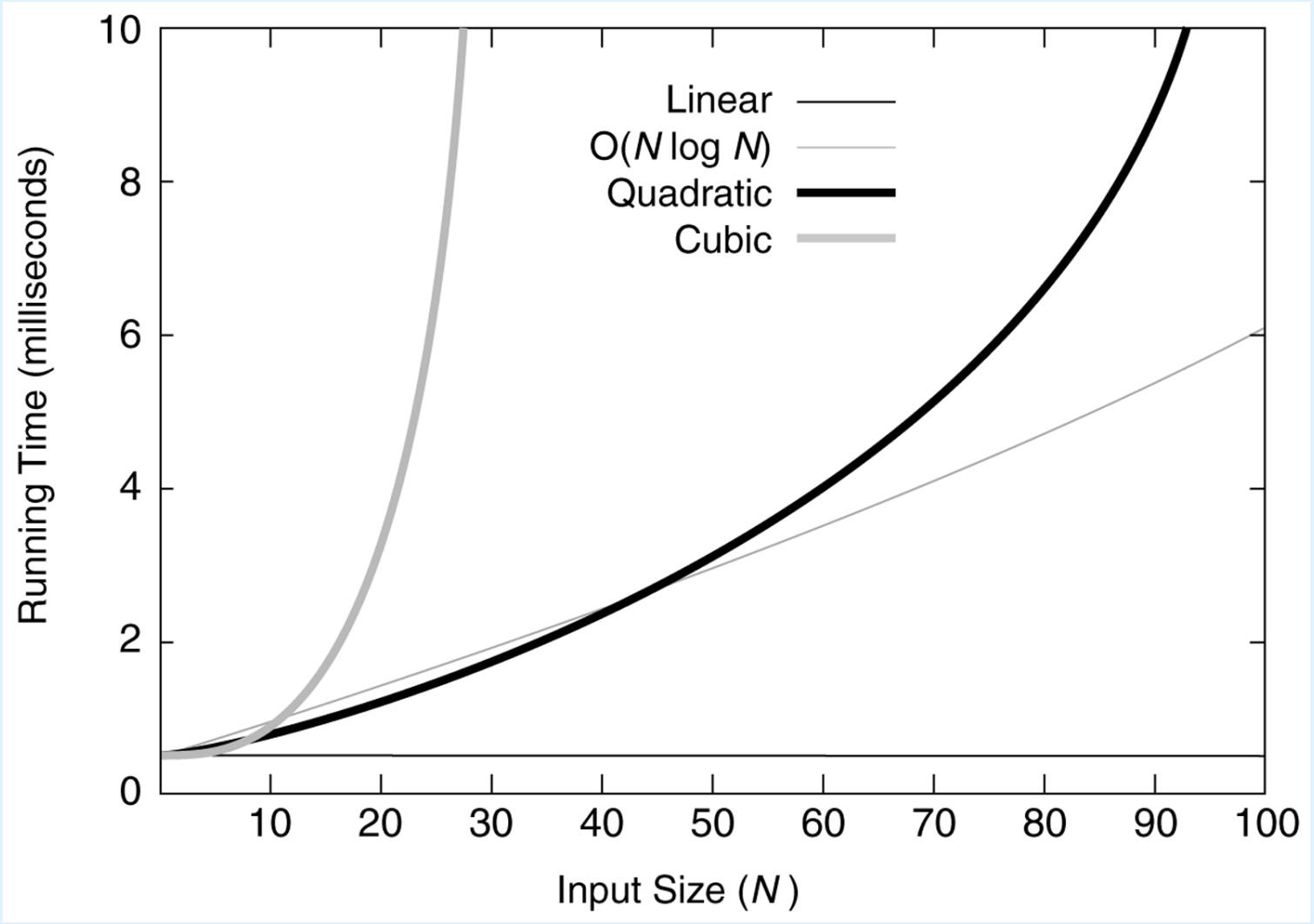
$10^5n^2 + 10^8n$ is a quadratic function (n^2)



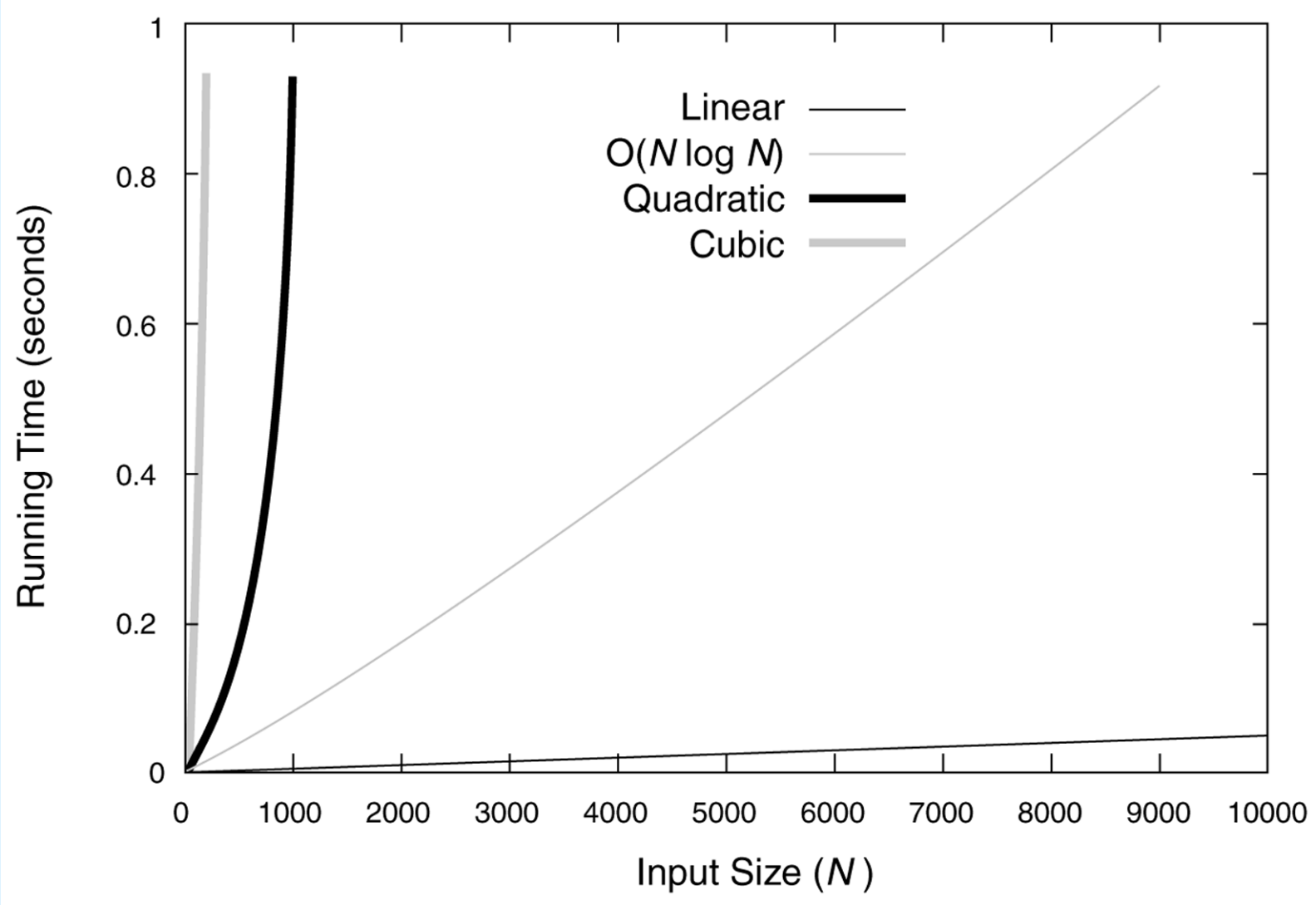
Common Growth Rates

Function	Growth Rate Name
C	Constant
log N	Logarithmic
$\log^2 N$	Log-squared
N	Linear
N log N	Logarithmic
N^2	Quadratic
N^3	Cubic
2^N	Exponential

Running times for small inputs



Running times for moderate inputs



A Comparison of Growth-Rate Functions