Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Computational Complexity

Data Structure and Algorithms

Slides credit: Ms. Saba Anwar, CUI Lahore

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

outline

- Analysis of Algorithms
  - Time complexity
    - Experimental Analysis
    - Big-O Analysis
- Space complexity

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

What is the COST of an algorithm?

Algorithm analysis is all about how efficient an algorithm is?

Like binary search is more efficient than linear search but how?

Efficiency is measured in two terms:

1. Time Complexity or Running Time:

   How much time it takes to complete

2. Space Complexity:

   How much space it occupies while running

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Time Complexity

Most algorithms transform input into output.

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

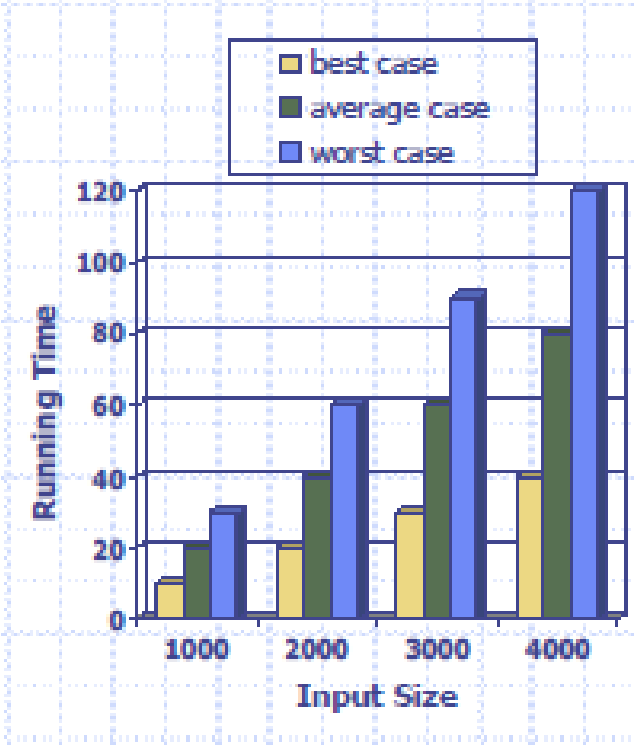
Average case time is often difficult to determine.

We focus on the worst case running time.

- Easier to analyze
- Crucial to applications such as games, finance and robotics.
  - what would happen if an autopilot algorithm ran drastically
  - slower for some unforeseen, untested inputs

How to measure?

- Experimental analysis
- Big-O Analysis



Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Experimental Analysis

Implement the algorithm

Run it on a machine with different input sizes

Measure the actual running times and plot the results

```
1  /** Uses repeated concatenation to compose a String with n copies of character c. */
2  public static String repeat1(char c, int n) {
3    String answer = "";
4    for (int j=0; j < n; j++)
5      answer += c;
6    return answer;
7  }
8
9  /** Uses StringBuilder to compose a String with n copies of character c. */
10 public static String repeat2(char c, int n) {
11   StringBuilder sb = new StringBuilder( );
12   for (int j=0; j < n; j++)
13     sb.append(c);
14   return sb.toString( );
15 }
```

Code Fragment 4.2: Two algorithms for composing a string of repeated characters.

n	repeat1 (in ms)	repeat2 (in ms)
50,000	2,884	1
100,000	7,437	1
200,000	39,158	2
400,000	170,173	3
800,000	690,836	7
1,600,000	2,874,968	13
3,200,000	12,809,631	28
6,400,000	59,594,275	58
12,800,000	265,696,421	135

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Experimental Analysis

Problems:

- An algorithm must be fully implemented to study its running time
- Results will vary depending upon hardware of machine, CPU, Memory
  - Even same system can take different time for same input, as CPU is shared with other processes.
- Running times of two algorithms are difficult to directly compare unless the experiments are performed in the same hardware and software environments.
- One program may be better implemented than the other
- Experiments can be done only on a limited set of test inputs; hence, they leave out the running times of inputs not included in the experiment (and these inputs may be important).

A Better Approach?

- Count the steps rather then time

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Time Complexity Analysis

The running time of an algorithms is an estimated total number of operations that it will perform for a given size of input. Number of operations executed by algorithm is defined as a function T of n. where n is input size.

T(n) is called time complexity function, let say for a particular algorithm T(n) is following:

$T(n)=3n+1$

It means this algorithm will take 3n+1 steps for given value of n.

Advantages:

Is performed by studying a high-level description of the algorithm without need for implementation.

Allows us to evaluate the relative efficiency of any two algorithms in a way that is independent of the hardware and software environment.

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Time Complexity Analysis

Algorithm is composed of set of primitive/basic operations/steps like:

- Variable assignment
- Math (+, -, *, /, %)
- Comparisons (==, >, <=, ...)
- Calling a method
- Returning from a method
- Array allocation
- Creating a new object

- Even though each primitive operation takes different time but implicit assumption in this approach is that the running times of different primitive operations will be fairly similar.

  Thus, T(n) will be will be proportional to the actual running time of that algorithm.

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Calculating T(n)

To estimate T(n), we simply need to know the total number of steps it takes to complete.

1. sum= 0
2. For i = 0 to n-1
3. sum+= i
4. i=i+1
5. End For
6. print sum

T(n)=3n+4

How we counted the steps?

How many operations are performed?

assignment, condition, return

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Single Loop

1. sum= 0 — 1 assignment step
2. For i = 1 to n — 1 assignment step i=1, n+1 conditional step (i.e. i<n
3. sum+= i — n addition steps
4. i=i+1 — n increment step
5. End For
6. print sum — 1 print step

$T(n)=3n+4$

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Nested Loops

1. sum=0 — 1 step
2. for i=1 to n step 1 — i=1→1 to n→ n+1 Step 1 increment→ n
3. for j=1 to k step 1 — j=1→1*n to n→ n*k+1 Step 1 increment→ n*k
4. sum++ — 1 step (n*k time)
5. Print sum — 1 step

$T(n,k)= 3nk+3n+4$

There can be more then one parameters that decide T. in this case T is defined as function of all those parameters.

*The total number of times a statement inside inner loop executes = outer loop times * inner loop times*

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Time Complexity Analysis

T(n) can vary for different scenarios for same value of n, there can be three cases:

Worst case

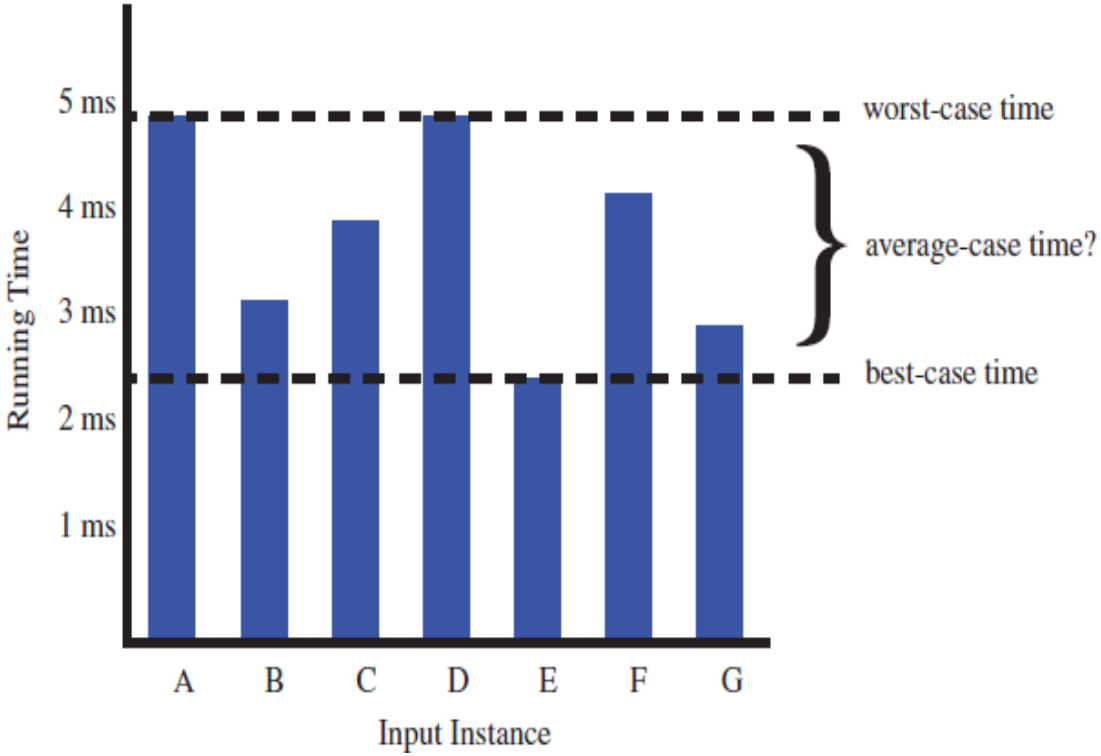
Maximum number of steps an algorithm takes on any input of size n.

Best case

Minimum number of steps an algorithm takes on any input of size n.

Average case

Average number of steps an algorithm takes on any input of size n.



Worst-case analysis is much easier than average-case analysis, as it requires only the ability to identify the worst-case input, which is often simple

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Growth Order?

$T(n)=3n+4$ $T(n)=5n$ $T(n)=n^2+100$

Which one is efficient?

If growth order of one algorithm is slower than other algorithm for very larger n, we say that algorithm with slower growth order is more efficient than algorithm with faster growth order.

This process of approximating growth order of T(n) is called ASYMPTOTIC COMPLEXITY or ASYMPTOTIC BEHAVIOR of algorithm.

Asymptotic complexity studies the efficiency of an algorithm as the input size becomes large

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Growth Order?

LINEAR_SEARCH(A[],n,v)

1 n+1 n

1. For i = 0 ; i< n; i++
2. if(A[i]==v) n
3. return i 1
4. End For
5. return -1 1

Best case: 4

V at first location

- Worst case:T(n)=3n+4

1 return will execute

How much faster T(n) will grow for larger values of n?

n	T(n)
100	3*100+4
10,000	3*10,000+4
100,000	3*100,000+4

As n becomes large, the term with n changes more than constant term 4

we can ignore constants for larger values of n

Because their contribution to T remains same

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Growth Order?

Similarly see the growth rate of $T(n)= 5n^{2+}2n+5$ n3

n	T(n)
100	5*100*100+2*100+5
10,000	5*10,000*10,000+2*10,000+5
100,000	5*100,000*100,000+2*100,000+5

As n grows large, term with largest order becomes dominant

- It means lower order terms can be neglected for larger values of n to find growth order
  - Because their contribution to T grows at slower rate when compared to the terms with higher order when n grows.
- Coefficients can also be neglected
  - They become irrelevant when comparing two functions with different order

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Asymptotic Analysis

Asymptotic analysis is very useful for comparing algorithm efficiency. It defines the growth order of algorithm rather than exact number of steps T(n)

As steps are not universally defined, so T(n) is not very accurate approach to measure time complexity. Moreover, every machine can take different time for given steps.

So Asymptotic analysis categorize the algorithms on basis of their growth order

$T(n)= 3n+4 \rightarrow O(n)$ $T(n)=n^2+5n+100 \rightarrow O(n^2)$ $T(n)= 7n+1000 \rightarrow O(n)$

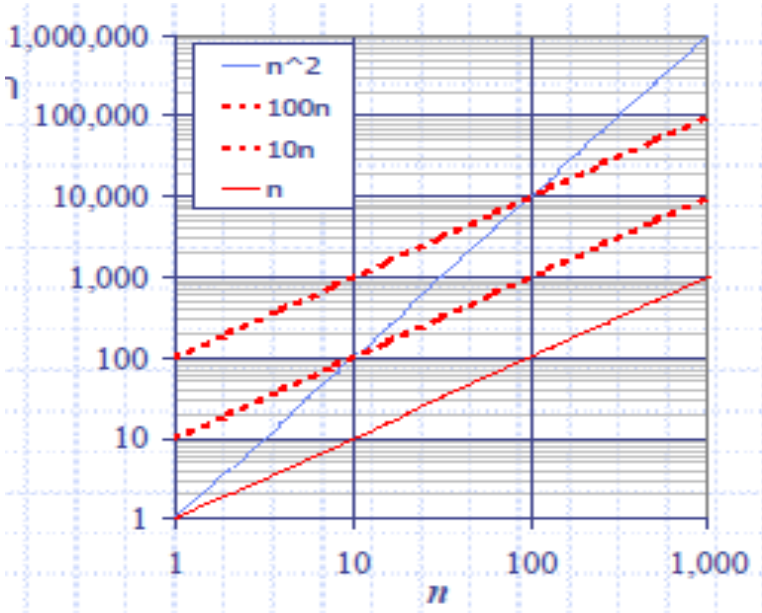
Algorithm with complexity Order n^2 is slower than algorithm with Order n

O stands for Order, and is referred as Big-O

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Big-O Rules

1. If $f(n)$ is a polynomial of degree d, then $f(n)$ is $O(n^d)$.
   Drop lower-order terms
   Drop constant factors
   Example: $T(n)=5n^2+n \rightarrow O(n^2)$
2. If $f(n)=c$ where c is any constant, then $f(n)$ is $O(1)$



Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Major Growth Orders

The big-O notation gives an upper bound on the growth rate of a function

We can use the big-O notation to rank functions according to their growth rate.

Seven functions are ordered by increasing growth rate in the sequence below, that is, if a function f (n) precedes a function g(n) in the sequence, then f(n) is O(g(n)):

constant	*logarithm*	*linear*	*n-log-n*	*quadratic*	*cubic*	*exponential*
1	$\log n$	n	$n \log n$	n^2	n^3	a^n

Table 4.1: Classes of functions. Here we assume that $a > 1$ is a constant.

Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

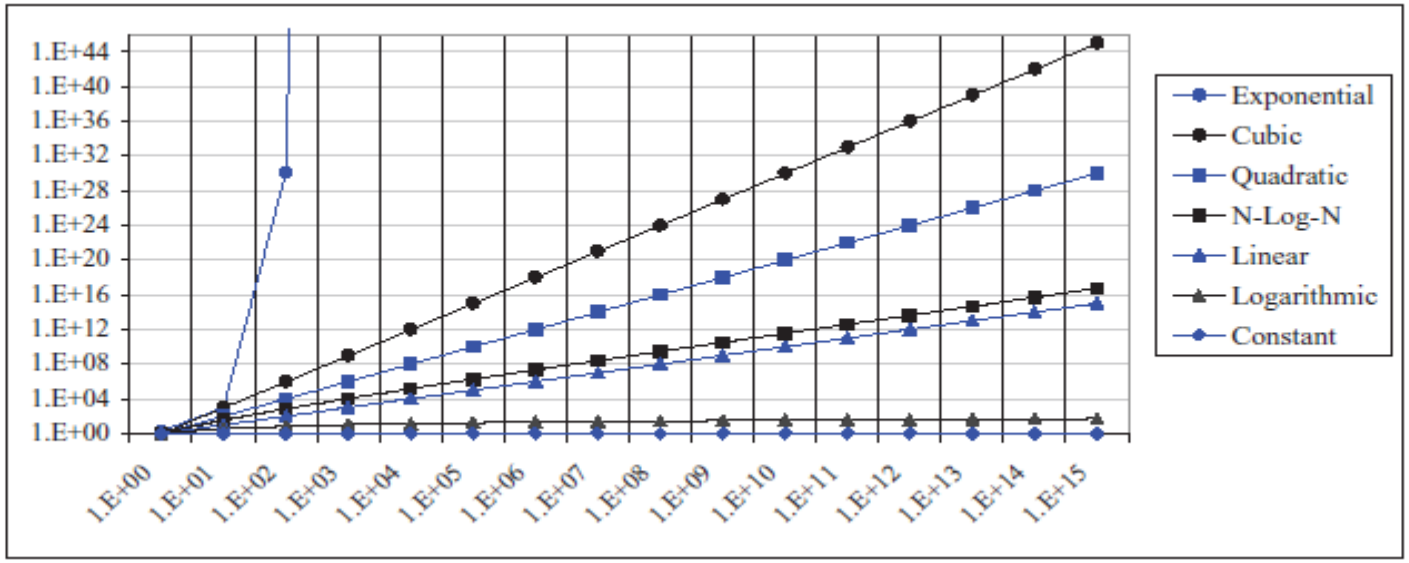
Growth Rate Comparison

Growth rate comparison:

Difference is more visible

at larger values of n

n	$\log n$	n	$n \log n$	n^2	n^3	2^n
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	4,096	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}



Edited with the trial version of
Foxit Advanced PDF Editor
To remove this notice, visit:
www.foxitsoftware.com/shopping

Summary

In this lecture, we have discussed:

- Complexities of algorithms in terms of time and space
- Time complexity function T(n)
- Asymptotic analysis and Big-O