Concepts of Algorithms
CS-211
Chapter 1
Algorithm Analysis

Dr. Fethi Fkih
College of Computer
Qassim University

General Information

- Instructor:

Dr. Fethi Fkih

- E-mail:

f.fki@qu.edu.sa

- Text Books:

Introduction to Algorithm Second Edition

by

Thomas H. Cormen

Charles E. Leiserson

Ronald L. Rivest

Clifford Stein

Algorithm

- An ***algorithm*** is a set of instructions to be followed to solve a problem.
  - There can be more than one solution (more than one algorithm) to solve a given problem.
  - An algorithm can be implemented using different programming languages on different platforms.
- An algorithm must be correct. It should correctly solve the problem.
  - e.g. For sorting, this means even if

    (1) the input is already sorted, or

    (2) it contains repeated elements.
- Once we have a correct algorithm for a problem, we have to determine the efficiency of that algorithm.

Algorithmic Performance

There are *two aspects* of algorithmic performance:

- Time
  - Instructions take time.
  - How fast does the algorithm perform?
  - What affects its runtime?
- Space
  - Data structures take space
  - What kind of data structures can be used?
  - How does choice of data structure affect the runtime?

➢ We will focus on time:

– How to estimate the time required for an algorithm

– How to reduce the time required

Analysis of Algorithms

- ***Analysis of Algorithms*** is the area of computer science that provides tools to analyze the efficiency of different methods of solutions.
- How do we compare the time efficiency of two algorithms that solve the same problem?

  Native Approach: implement these algorithms in a programming language (C++), and run them to compare their time requirements. Comparing the programs (instead of algorithms) has difficulties.

  - *How are the algorithms coded?*
    - Comparing running times means comparing the implementations.
    - We should not compare implementations, because they are sensitive to programming style that may cloud the issue of which algorithm is inherently more efficient.

Analysis of Algorithms

- ***Analysis of Algorithms*** is the area of computer science that provides tools to analyze the efficiency of different methods of solutions.
- How do we compare the time efficiency of two algorithms that solve the same problem?

  Native Approach: implement these algorithms in a programming language (C++), and run them to compare their time requirements. Comparing the programs (instead of algorithms) has difficulties.

  - *What computer should we use?*
    - We should compare the efficiency of the algorithms independently of a particular computer.

Analysis of Algorithms

- ***Analysis of Algorithms*** is the area of computer science that provides tools to analyze the efficiency of different methods of solutions.
- How do we compare the time efficiency of two algorithms that solve the same problem?

  Native Approach: implement these algorithms in a programming language (C++), and run them to compare their time requirements. Comparing the programs (instead of algorithms) has difficulties.

  - *What data should the program use?*
    - Any analysis must be independent of specific data.

Analysis of Algorithms

- When we analyze algorithms, we should employ mathematical techniques that analyze algorithms independently of *specific implementations, computers, or data.*

- To analyze algorithms:
  - First, we start to count the number of significant operations in a particular solution to assess its efficiency.
  - Then, we will express the efficiency of algorithms using growth functions.

The Execution Time of Algorithms

- Each operation in an algorithm (or a program) has a cost.

  ➔ Each operation takes a certain of time.

`count = count + 1;` ➔ take a certain amount of time, but it is constant

A sequence of operations:

```
count = count + 1;
sum = sum + count;
```

Cost: c_1

Cost: c_2

➔ Total Cost = $c_1 + c_2$

The Execution Time of Algorithms (cont.)

Example: Simple If-Statement

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

Total Cost <= c1 + max(c2,c3)

The Execution Time of Algorithms (cont.)

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

= (c3+c4+c5)*n + (c1+c2+c3)

= a*n + b

(a = c3+c4+c5, b = c1+c2+c3)

➔ The time required for this algorithm is proportional to **n**

The Execution Time of Algorithms (cont.)

Example: Simple Loop

	Cost	Times
`i = 1;`	c1	1
`sum = 0;`	c2	1
`while (i <= n) {`	c3	log(n)+2
`  i = i*2;`	c4	log(n)+1
`  sum = sum + i;`	c5	log(n)+1
`}`		

Total Cost = c1 + c2 + (log(n)+2)*c3 + (log(n)+1)*c4 + (log(n)+1)* c5

= (c3+c4+c5)*log(n) + (c1+c2+2*c3+c4+c5)

= a*log(n) + b

➔ The time required for this algorithm is proportional to log(n)

The Execution Time of Algorithms (cont.)

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

= (c5+c6+c7)*n^2 + (c3+c4+c5+c8)*n + (c1+c2+c3)

|___a____| |_____b_____| |___c____|

= a*n^2 + b*n + c

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

The Execution Time of Algorithms (cont.)

Example: Nested Loop

```
                                        Cost        Times
i=n;                                    c1          1
sum = 0;                                c2          1
while (i>1) {                           c3          log(n)+1
    j=1;                                c4          log(n)
    while (j <= n) {                    c5          log(n)*(n+1)
        sum = sum + i;                  c6          log(n)*n
        j = j + 1;                      c7          log(n)*n
    }
    i = i/2;                            c8          log(n)
}
```

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

= (c5+c6+c7)*nlog(n) + (c3+c4+c5+c8)*log(n) + (c1+c2+c3)

|___a____| |_____b_____| |___c____|

= a*nlog(n) + b*log(n) + c ➔ O(nlog(n))

The Execution Time of Algorithms (cont.)

Example: Nested Loop

```
                                        Cost        Times
i=1;                                    c1          1
sum = 0;                                c2          1
while (i < n) {                         c3          log(n)+1
    j=1;                                c4          log(n)
    while (j <= n) {                    c5          log(n)*(n+1)
        sum = sum + i;                  c6          log(n)*n
        j = j + 1;                      c7          log(n)*n
    }
    i = i*2;                            c8          log(n)
}
```

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

= (c5+c6+c7)*nlog(n) + (c3+c4+c5+c8)*log(n) + (c1+c2+c3)

|___a____| |_____b_____| |___c____|

= a*nlog(n) + b*log(n) + c ➔ O(nlog(n))

General Rules for Estimation

- **Loops**: The running time of a loop is at most the running time of the statements inside of that loop times the number of iterations.
- **Nested Loops**: Running time of a nested loop containing a statement in the inner most loop is the running time of statement multiplied by the product of the sized of all loops.
- **Consecutive Statements:** Just add the running times of those consecutive statements.
- **If/Else**: Never more than the running time of the test plus the larger of running times of S1 and S2.

The Execution Time of Algorithms (cont.)

	Cost	Times
`for (i=1; i<=n; i++)`	c1	n+1
`for (j=1; j<=i; j++)`	c2	$\sum_{j=1}^{n}(j+1)$
`x=x+1;`	c3	$\sum_{j=1}^{n} j$

Total Cost = $c1*(n+1) + c2*\left(\sum_{j=1}^{n}(j+1)\right) + c3*\left(\sum_{j=1}^{n} j\right)$

$= a*n^2 + b*n + c$

➔ The time required for this algorithm is proportional to $\mathbf{(n^2)}$

Some Rules

- For(i=Vf; i<=Vi; i++){

  instractions /*Vf-Vi+1*/

}

- 1+2+.....+(n-1)=n(n-1)/2

- 1+2+.....+n=n(n+1)/2

The Execution Time of Algorithms (cont.)

	Cost	Times
`for (i=1; i<=n; i++)`	c1	n+1
`  for (j=1; j<=m; j++)`	c2	n*(m+1)
`    for (k=1; k<s; k++)`	c3	n*m*s
`      x=x+1;`	c4	n*m*(s-1)

Total Cost = c1+ n*(C1+C2) + n*m*(C2-C4) + n*m*s* (C3+C4)

= a*n*m*s + b*n*m + c*n + d

➔ The time required for this algorithm is proportional to **(n*m*s)**

What to Analyze

- An algorithm can require different times to solve different problems of the same size.
  - Eg. Searching an item in a list of n elements using sequential search. ➔ Cost: 1,2,...,n
- ***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 of an algorithm is NOT so useful.
- ***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 of these organizations.
  - ***Worst-case analysis is more common than average-case analysis.***

END