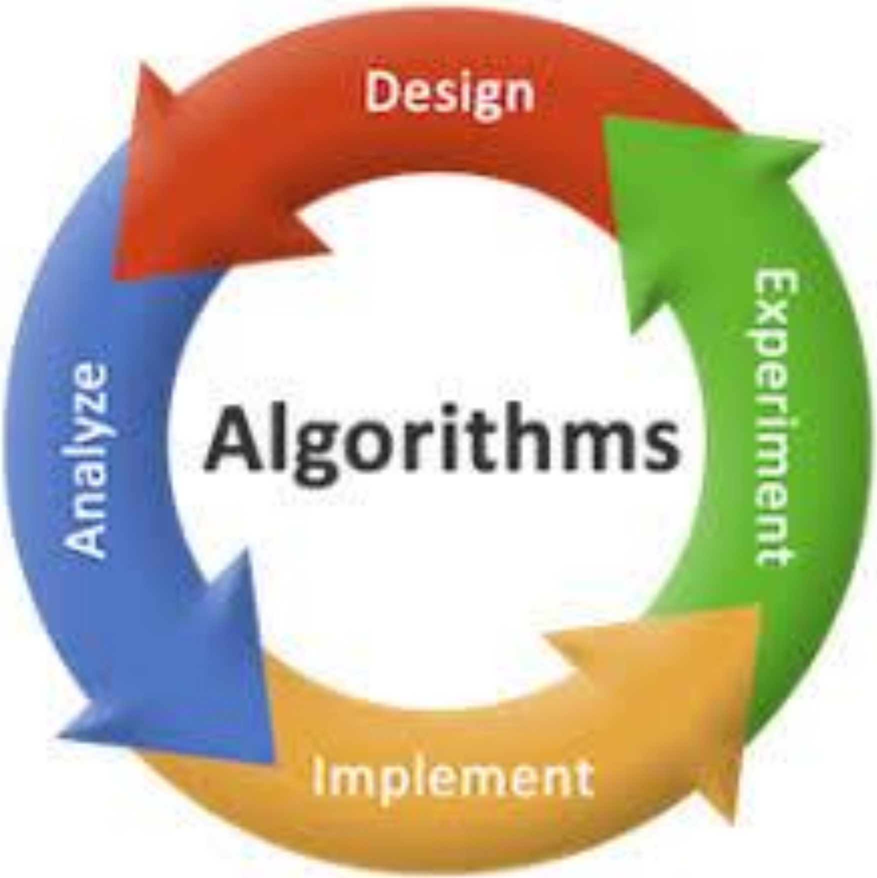

Design & Analysis of Algorithms

Lecture 5

Analysis of Algorithm Efficiency

Analysis Framework

Two kinds of efficiency:

- **Time efficiency**, also called **time complexity**, indicates how fast an algorithm in question runs.
- **Space efficiency**, also called **space complexity**, refers to the amount of memory units required by the algorithm in addition to the space needed for its input and output.

Analysis Framework

Measuring an Input’s Size

- Almost all algorithms run longer on larger inputs.

  For example

  - longer to sort larger arrays, multiply larger matrices, ...

- Investigating an algorithm’s efficiency as a function of some parameter ***n*** indicating the algorithm’s input size

Analysis Framework

Measuring an Input's Size

Exception-Example

- For evaluating a polynomial $p(x) = a_n x^n + \ldots + a_0$ of degree *n,* it will be the polynomial's degree or the number of its coefficients, which is larger by 1 than its degree.

Analysis Framework

Measuring an Input's Size

Other Examples

- Searching among **n** elements
- Sorting **n** elements
- computing the product of two **n × n** matrices
- Primality testing of a number **n** [In such situations, it is preferable to measure size by the number b of bits in the n's binary representation: $b = \lfloor \log_2 n \rfloor + 1$]

Analysis Framework

Orders of Growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Analysis Framework

Worst-Case, Best-Case, and Average-Case Efficiencies

- The **worst-case** efficiency of an algorithm is its efficiency for the worst-case input of size **n**, which is an input (or inputs) of size n for which the algorithm runs the longest among all possible inputs of that size.

Analysis Framework

Worst-Case, Best-Case, and Average-Case Efficiencies

- The **best-case** efficiency of an algorithm is its efficiency for the best-case input of size **n**, which is an input (or inputs) of size n for which the algorithm runs the fastest among all possible inputs of that size.

Analysis Framework

Worst-Case, Best-Case, and Average-Case Efficiencies

- However, that neither the **worst-case** analysis nor its **best-case** counterpart yields the necessary information about an algorithm's behavior on a "typical" or "random" input.
- This is the information that the **average-case** efficiency seeks to provide.

Analysis Framework

Worst-Case, Best-Case, and Average-Case Efficiencies

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$

```
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: The index of the first element in A that matches K
//          or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```

References

- **Chapter 2**: Anany Levitin, “Introduction to the Design and Analysis of Algorithms”, Pearson Education, Third Edition, 2017

- **Chapter 2**: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, “Introduction to Algorithms”, MIT Press/PHI Learning Private Limited, Third Edition, 2012.

Homework

- 1. a. Consider the definition-based algorithm for adding two $n \times n$ matrices. What is its basic operation? How many times is it performed as a function of the matrix order n? As a function of the total number of elements in the input matrices?

  b. Answer the same questions for the definition-based algorithm for matrix multiplication.

- 2. Consider a variation of sequential search that scans a list to return the number of occurrences of a given search key in the list. Does its efficiency differ from the efficiency of classic sequential search?