# Programming Foundation With Pseudocode

Lesson 4: Algorithm Analysis and Design

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

# Lesson Objectives

➢ **To understand the following concepts**

    – Algorithm Analysis and efficiency

    – Measuring Unit for Algorithm

    – Order of Growth

        • Asymptotic notations

    – Best/Worst/Average case

    – Examples

    – Space Efficiency

# Algorithm Analysis and efficiency

- **Why Algorithm Analysis?**
  - A problem can have solution/multiple solutions.
  - To establish if a given algorithm uses a reasonable amount of resources to solve a problem, an Analysis of algorithm is required.
  - Ex: Google search algorithm.
- **There are two kinds of algorithm efficiency,**
  - Time efficiency
  - Space efficiency
- **Time efficiency indicates how fast an algorithm runs.**
- **Space efficiency indicates how much extra memory the algorithm needs.**

# Measuring Unit for Algorithm

➤ **How an Algorithm can be measured?**
  ➤ Running time of the implemented algorithm is the solution?

➤ **Drawbacks are,**
  ➤ It becomes machine dependent.
  ➤ Program dependent
  ➤ Compiler dependent etc.

➤ **As we are measuring Algorithm's efficiency, it is better to have a metric which is independent of all the factors like mentioned above.**

# Measuring Unit for Algorithm

- ➢ **Basic operation is the unit used for algorithm efficiency.**

- ➢ **What is Basic operation?**
    - ➢ It the operation contributing the most to the total running time of the algorithm. Generally, statements in the innermost loops.

- ➢ **Algorithm efficiency: Number of times the basic operation is executed for input size, n.**

- ➢ **Time taken by program implementing this algorithm is, $T(n)=C_{op} * C(n)$**
    - ➢ T(n): running time as function of n.
    - ➢ Cop: running time of single basic operation.
    - ➢ C(n): number of basic operations as a function of n.

- ➢ **The number of times basic operation is executed depends on input size, n.**

# Order of Growth

**Order of Growth:**

> ➢ To analyze an algorithm, we are interested in order of growth, i.e. how the running time increases when the input size increases.

> ➢ When we want to compare two algorithms with respect to their behavior for large input sizes, a useful measure is the so called, order of growth.

**Asymptotic Notations:**

➢ **To compare and rank order of growth multiple notations are used like,**

> ➢ O ( Big oh)

> ➢ Ω (Big omega)

> ➢ Θ (Big theta)

➢ **Mostly Big oh is used to represent the time efficiency of algorithm.**

➢ **Example:**

> ➢ $O(n)$, $O(n^2)$, $O(n^3)$, $O(nlogn)$ etc.

# Order of Growth

**Examples:**

➤ **Ex 1: Finding sum of elements in an array.**

  ➤ Basic operation is addition. Number of times basic operation is executed is n times.

  ➤ Efficiency of this algorithm is O(n).

➤ **Ex 2: Algorithm to find the sum of elements stored in two dimensional matrix.**

  ➤ Basic Operation: Addition operation in the inner most loop.

  ➤ As this has two loop's which is scanning array twice, number of times basic operation is executed is $n^2$ times.

  ➤ Efficiency of this algorithm is $O(n^2)$.

# Order of Growth

> **Basic Efficiency classes : Running time for most of the algorithms falls under different efficiency classes.**

High time efficiency

| Class | Name |
|-------|------|
| 1 | constant |
| log n | logarithmic |
| n | Linear |
| nlogn | nlogn |
| $n^2$ | quadratic |
| $n^3$ | cubic |
| $2^n$ | Exponential |
| n! | factorial |

Low time efficiency

Table 4.1. Efficiency classes

| n | $\log_2 n$ | n | $n\log_2 n$ | $n^2$ | $n^3$ | $2^n$ | n! |
|---|-----------|---|-------------|-------|-------|-------|-----|
| **10** | 3.3 | 10 | $3.3*10^1$ | $10^2$ | $10^3$ | $10^3$ | $3.6*10^6$ |
| **$10^2$** | 6.6 | $10^2$ | $6.6*10^2$ | $10^4$ | $10^6$ | $1.3*10^{30}$ | $9.3*10^{157}$ |
| **$10^3$** | 10 | $10^3$ | $1.0*10^4$ | $10^6$ | $10^9$ | | |
| **$10^4$** | 13 | $10^4$ | $1.3*10^5$ | $10^8$ | $10^{12}$ | | |

Table 4.2. If n is the input size, following table will show the values of several functions which helps in understanding algorithm analysis.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Best/Worst/Average Cases

➢ **Running time not only depends on the input size, n , but also depends on specific parameter of a particular input.**

➢ **Time taken by algorithm is not same for all the inputs.**

➢ **It also depends on other input parameter. For such algorithm we find 3 types of efficiencies,**

– Best case efficiency

– Worst case efficiency

– Average case efficiency

# Searching Techniques

➢ **Searching is looking for an element in a set of elements.**

➢ **For Example**

    – Searching a word in a dictionary which consists of sorted words.

    – Searching for Employee Details With Employee Number in an Employee Directory

➢ **Searching comprises of following algorithms**

    – Sequential Search or Linear Search

    – Binary search

# Sequential Search

➢ **Sequential search is also called as "Linear Search "**

– It is the simplest searching technique if number of elements are less

– It is useful when data is unsorted

– It operates by checking every element of a list one at a time in sequence until a match is found

• Best case: find the value at first position

• Worst case: find the value at last position

• Average case: find the value at the middle

# Sequential Search - Example

➢ **Example on Sequential search:**

– Consider an array as shown below:

- 14  15   23 10   7  9

– To search whether the number 23 exists in the array or not:

- Start comparing from the first element one by one till we find the number or we reach the last element in the array
- We should stop searching once we get the number at 2nd position and return the position

– To search element 50

- Start comparing from the first element one by one till we find the number or we reach the last element in the array. If we have reached the end of the array give a message saying element not found

# Sequential Search - Example

➢ **For Sequential search,**

➢ **Best Case: $C_{best}(n) = 1$**

    ➢ When key is found on the first comparison.

➢ **Worst Case:  $C_{worst}(n) = n$**

    ➢ When key is found on the last comparison or not found in the list.

➢ **Average Case: $C_{average}(n) = (n+1)/2$**

    ➢ When key is somewhere in between the list.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Binary Search - Features

➢ **Binary Search is useful for searching sorted data**

➢ **It is faster than sequential search**

➢ **It reduces the span of searching the value**

➢ **Steps involved in Binary Search:**

  – Compare the value at middle, otherwise divide the data into two parts at the middle

  – If the value to be searched is less than (<) the value at middle, then search in the fist half otherwise in the next half

➢ **Best case - The value is at the middle position**

➢ **Efficiency is**

  – Best Case: $C_{Best}(n)=1$

  – Worst Case: $C_{Worst}(n)=O(\log n)$

  – Average Case: $C_{Avg}(n)=O(\log n)$

# Binary Search - Example

➤ **Example on Binary search:**

    – Consider an array as shown below:

        • 7 9 14 18 22 33 55

    – To search whether the number 14 exists in the array or not:

        • Find the middle value: As number of elements available in the array is 7 , choose 4$^{th}$ element as middle value(18). Choose values 7, 9, 14 as subset 1. Consider values 22, 33, 55 as subset 2.

        • Start comparing search element with the middle element and stop searching if middle element is equal to search element.

        • If search element is not equal to middle element, identify whether search element is less than middlevalue. If search element<middlevalue, then start comparing search value with subset1 by following from step1.

        • If search element is not equal to middle element, identify whether search element is greater than middlevalue. If search element>middlevalue, then start comparing search value with subset2 by following from step1.

        • If element not matched, then display message as "Element not found".

# Comparison - Example

➢ **How do you look up a telephone number in the directory?**

  – You look at the name, say "Pramod Patil", open the directory at random, and compare the first character of the first name on that page

  – If the first character is less the "P", you continue the search in the second half

  – If the first character is greater than "P", you continue the search in the first half

➢ **If the first character is "P", you continue the search using the second character until you find the name you started with**

➢ **What is the pre-requisite, to succeed with this kind of search?**

  – Answer:  The directory has to be sorted in an ascending order of names!

# Comparison - Example

➢ **Now, hypothesize that you have got hold of a phone number.  You need to find out who it belongs to!**

➢ **Assuming you have only the directory to refer to, how will you locate the owner of this number?**

–  Answer: You need to do a "Sequential Search" on the Phone Numbers!!

➢ **Suppose that a data set has N items:**

–  How many comparisons are required on an average by using "Sequential Search"?

➢ **"Sequential Search" requires N/2 comparisons on an average**

➢ **Suppose a data set has N items:**

–  How many comparisons would be required for a "Binary Search"?

➢  **"Binary Search" requires log2(N) comparisons**

# Comparison – Binary search

➢ **Note that, while using Binary search:**

– For N = 1000, you require maximum 10 comparisons

– For N = 1 million, you require maximum 20 comparisons

– The condition you have to fulfill is that you need to keep the data sorted on the relevant field

➢ **Develop the Algorithm for Binary search as pseudo code**

# Discussion

➤ **In the following scenarios, which algorithm will you use for searching:**

– You want to purchase a birthday gift of photo frame for your friend from a collection of photo frames in the local photo store.

– You have a large data set that is in unsorted order in front of you. You need to complete N searches on this data set. Does it make more sense to use linear search, or to sort it and use binary search?

– Consider XYZ Bank, stores their Customer database in a big array, sorted alphabetically by Customer Id. The array contains 2.5 million elements. How many comparisons, at most, will it take to locate the data it is searching for?

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Example 1– MaxElement

1. ALGORITHM *MaxElement(List[0..n-1])*
2. //Determines largest element in Array
3. //Input: An array list[0..n-1] of real numbers
4. //Output: Value of largest element
5. currentmax ← list[0]
6. for index ← 0 to n-1 do
7.         if list[index] > currentmax
8.             currentmax ← list[index]
9. return currentmax

# Example 1 – MaxElement

➢ **Basic Operation:  Comparison statement in the loop (Line 7). There is only one basic operation in the given algorithm.**

   ➢ For each loop, derive the summation of lower bound to upper. 1 indicates that there is only one basic operation.

➢ **Set up the summation:**

$$C(n) = \sum_{index=1}^{n-1} 1$$

➢ **Solve the summation accordingly,**

$$C(n) = \sum_{index=1}^{n-1} 1 = n-1 \in O(n)$$

➢ **Efficiency of given algorithm is O(n).**

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Example 2 – Selection sort

➢ **Lets take following selection sort example and find efficiency of the given algorithm.**

1.  ALGORITHM *SelectionSort(List[0..n-1])*
2.  //Sorts a given array by selection sort
3.  //Input: An array list[0..n-1] of orderable elements
4.  //Output: Array list[0..n-1] sorted in ascending order
5.  for index← 0 to n-2 do
6.        min←index
7.                  for nextindex←index+1 to n-1 do
8.                     if list[nextindex] < list[min]
9.            min←nextindex
10.        swap list[index] and list[min]

# Example – Selection sort

- **Basic Operation: Comparison statement in innermost loop (Line 8). There is only one basic operation in the given algorithm.**
  - For each loop, derive the summation of lower bound to upper. 1 indicates that there is only one basic operation.

- **Set up the summation:**

$$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

- **Solve the summation accordingly,**

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i). = \frac{(n-1)n}{2}$$

- **Efficiency of selection sort is Θ(n²).**

- **Note: Constants are ignored while concluding the efficiency class. The highest order of growth is considered as the efficiency class.**

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Example – Selection sort

Notes Page

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Example – Selection sort

Notes Page

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Example – Selection sort

- **Basic Operation:  Comparison statement in innermost loop (Line 8). There is only one basic operation in the given algorithm.**
  - For each loop, derive the summation of lower bound to upper. 1 indicates that there is only one basic operation.
- **Set up the summation:**

$$= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

- **Solve the summation accordingly,**

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2}[(n-1)-(i+1)+1] = \sum_{i=0}^{n-2}(n-1-i). \ = \frac{(n-1)n}{2}$$

- **Efficiency of selection sort is $\Theta(n^2)$.**
- **Note: Constants are ignored while concluding the efficiency class. The highest order of growth is considered as the efficiency class.**

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

# Example

➢ **Logarithmic**

   ➢ It is a result of cutting problem's size by a constant factor on each iteration of the algorithm.

   ➢ Example: Binary search algorithm, Quick sort (Worst case), etc.

➢ **n logn**

   ➢ Many divide and conquer algorithms fall into this category.

   ➢ Example: Merge sort(average case), quick sort(average case), etc.

# Space Efficiency

➢ **An algorithm that is space-efficient uses the least amount of computer memory to solve the problem.**

➢ **Following measures must be taken care to use the memory space efficiently,**

  ➢ Use local variables.

  ➢ Make sure program should not create any dangling pointers.

  ➢ Allocate memory dynamically.

  ➢ Use register variable where ever possible.

  ➢ Reuse the variables where ever possible.

# Lab

➢ **Analyzing Algorithms Lab 4**

# Summary

➢ **In this lesson, you have learnt about:**

– Algorithm Analysis and efficiency

– Measuring Unit for Algorithm

– Order of Growth

– Best/Worst/Average case

– Asymptotic notations

– Examples

– Space Efficiency

# Review Question

➢ **Question 1: If efficiency of Algorithm X is O(n²) and Algorithm Y is O(n logn), Which is more efficient ?**

    – Algorithm X

    – Algorithm Y

    – Both are same

    – Depends on input type.

➢ **Question 2: The time factor when determining the efficiency of algorithm is measured by**

        a. Counting micro seconds

        b. counting the number of key operations

        c. counting the number of statements

        d. counting the kilobytes of algorithm

# Review Question

➢ **Question 3: Two main measures for the efficiency of an algorithm are**

> a. Processor and memory
>
> b. Complexity and capacity
>
> c. Time and space
>
> d. Data and space

➢ **Question 4: If an algorithm's running time depends on not only the input size n, but also depends on other parameter, then we need to find Best, Worst and Average case efficiencies.** True/False