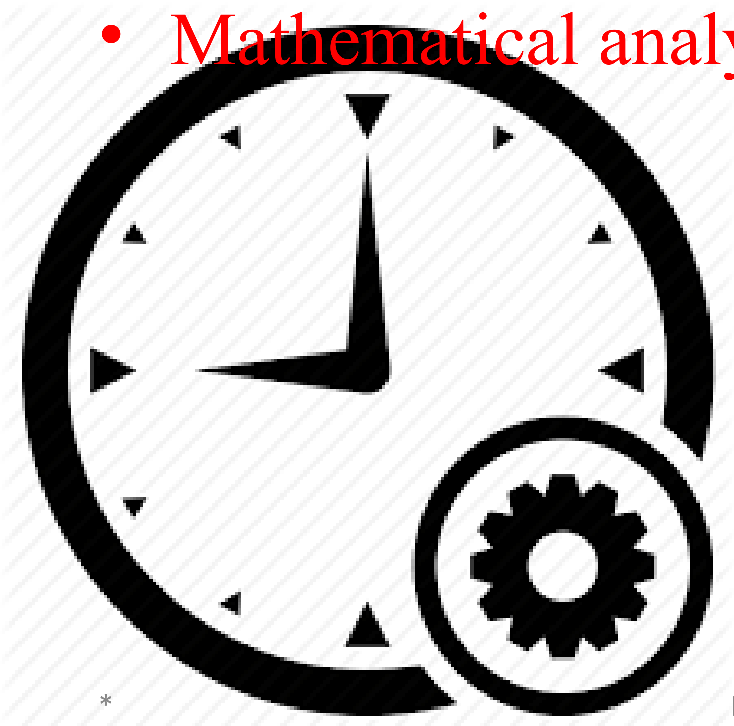
Fundamentals of the Analysis of Algorithm Efficiency

- Analysis Framework
- Asymptotic Notations and its properties
- Mathematical analysis of Non - Recursive algorithms
- Mathematical analysis of Recursive algorithms

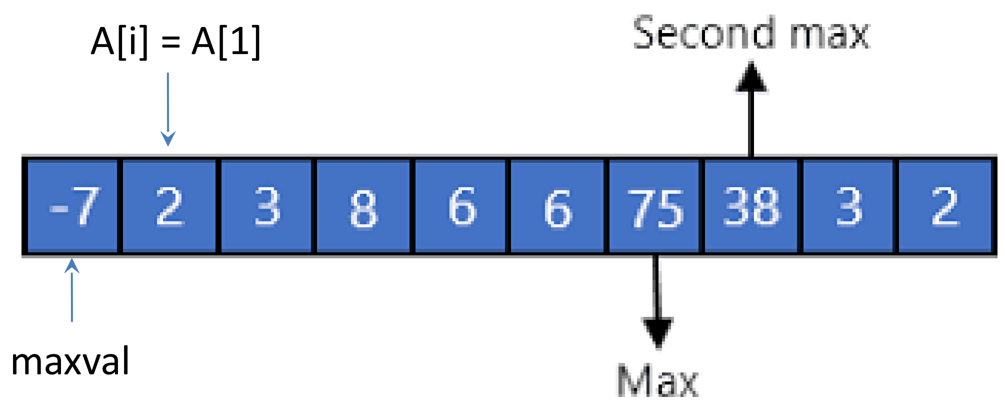


Mathematical analysis of Non - Recursive algorithms

- Analysis framework – systematic – analyze the time efficiency of non-recursive algorithm
- Example 1: Finding the largest value in a list of n numbers

ALGORITHM *MaxElement*($A[0..n-1]$)

//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
  if $A[i] > maxval$
    $maxval \leftarrow A[i]$
return $maxval$



Example 1: Finding the largest value in a list of n numbers

$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

1	What is the problem size	n
2	What is the basic operation	Comparison in for loop
3	Count of basic operation	$C(n) = \sum_{i=1}^{n-1} 1 = n\text{-}1 \; \varepsilon \; \boldsymbol{\Theta(n)}$
4	Depends on what efficiency? Worst/best/average	

General Plan for Analyzing the Time Efficiency of Non recursive Algorithms

1. Decide on a parameter (or parameters) indicating **an input's size.**
2. Identify the algorithm's **basic operation.** (As a rule, it is located in the inner- most loop.)
3. Check whether the **number of times the basic operation is executed** depends only on the size of an input. If it also depends on some additional property, the **worst-case, average-case, and, if necessary, best-case efficiencies** have to be investigated separately.
4. **Set up a sum** expressing the number of times the algorithm's basic operation is executed.
5. Using **standard formulas** and rules of sum manipulation, either find a closed-form formula for the count or, at the very least, establish its order of growth.

Formula for Sum Manipulation

$$\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i, \qquad \textbf{(R1)}$$

$$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i, \qquad \textbf{(R2)}$$

two summation formulas

$$\sum_{i=l}^{u} 1 = u - l + 1 \quad \text{where } l \le u \text{ are some lower and upper integer limits,} \qquad \textbf{(S1)}$$

$$\sum_{i=0}^{n} i = \sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2). \qquad \textbf{(S2)}$$

Example 2: Element Uniqueness Problem

10	**20**	**30**	**40**	**30**	**50**	**60**
1st	2nd	3rd	4th	5th	6th	7th
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]

Here: n=7, n-1 = 6, n-2=5

ALGORITHM *UniqueElements*$(A[0..n-1])$

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
// and "false" otherwise

for $i \leftarrow 0$ **to** $n-2$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    if $A[i] = A[j]$ **return false**
return true

Example 2: Element Uniqueness Problem

1	What is the problem size	n
2	What is the basic operation	if statement (comparison)
3	Count of basic operation	$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$:
4	Depends on what efficiency? Worst/best/average ***Worst case*** – all the elements are different – all sequence of for loop ***Best case*** – 1st and 2nd element are same – comes out of loop	
5	Summation	

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2)$$

Example 3: Sum of n numbers

Program:

```
count = 0;
for (i=1;i<=n;i++)
      count=count+i;
return count;
```

Example:

n = 5. count =0

i=1 → count =0+1 = 1

i=2 → count = 1+2 = 3

i=3 → count = 3+3 = 6

i=4 → count = 6+4 = 10

i=5 → count = 10+ 5 = 15

Analysis of sum of n numbers:

1.Problem size?

2.Basic Operation ?

3.Count of basic operation ?

4.Worst / Best / Average case efficiency ?

Example 4: to find the no of binary digits in a binary representation of a positive decimal integer

2^5	2^4	2^3	2^2	2^1	2^0
32	16	8	4	2	1
					1
				1	0
				1	1
	1	0	0	0	0

Number	Binary representation
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
15	1111
16	10000

ALGORITHM *Binary*(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

$count \leftarrow 1$

while $n > 1$ **do**

$\quad count \leftarrow count + 1$

$\quad n \leftarrow \lfloor n/2 \rfloor$

return $count$

Iteration	n value	Count
Initial	2	1
1 st		2
	n= n/2 = 1	

Iteration	n value	Count
Initial	3	1
1 st		2
	n= n/2 = 1.5	

Iteration	n value	Count
Initial	4	1
1 st		2
	n= 4/2 = 2	
2nd		3
	n=2/2=1	

Iteration	n value	Count
Initial	8	1
1 st		2
	n= 8/2 = 4	
2nd		3
	n=4/2=2	
3rd		4
	n=2/2=1	

Example 4: Analysis

1	What is the problem size	n
2	What is the basic operation	Comparison in while loop
3	Count of basic operation	$\boldsymbol{C(n) = \sum_{i=1}^{\lg(n)+1} 1}$
4	Depends on what efficiency? Worst/best/average	

Example 5: Matrix Multiplication

ALGORITHM *MatrixMultiplication*($A[0..n-1, 0..n-1]$, $B[0..n-1, 0..n-1]$)

//Multiplies two square matrices of order n by the definition-based algorithm

//Input: Two $n \times n$ matrices A and B

//Output: Matrix $C = AB$

for $i \leftarrow 0$ **to** $n-1$ **do**

for $j \leftarrow 0$ **to** $n-1$ **do**

$C[i, j] \leftarrow 0.0$

for $k \leftarrow 0$ **to** $n-1$ **do**

$C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$

return C

Example 5: Matrix Multiplication Analysis

1	What is the problem size	*Order of matrix*
2	What is the basic operation	Multiplication and addition
3	Count of basic operation	$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$
4	Depends on what efficiency? Worst/best/average	
5	Running time T(n) = Cop C(n) = Cm M(n) + Ca A(n) = Cm n^3 + $Ca\ n^3$ = $(Cm+Cn)\ n\ 3$	

Write the program for the following output and do the analysis process

1

1 2

1 2 3

1 2 3 4

1 2 3 4 5