بسم الله الرحمن الرحيم

BISMILLAH ARRAHMAN ARRAHEEM

CS302
Design and Analysis of Algorithms

Lecture 2:
Complexity Analysis

Dr. Fahad Sherwani

(Assistant Professor – FAST NUCES)

fahad.sherwani@nu.edu.pk

Calculating Complexity of Algorithm

Finding Sum of all elements of an array:

Algorithm : FindSum (A, n)
Input : An array 'A' of 'n' integers
Output : Sum of all integers in 'A'

STEPS

1- Sum = A[0]
2- for i = 1 to i = n-1
3- Sum = Sum + A[i]
4- return Sum

step 2 can also be written as:-

$$\text{for } (\underbrace{i=1}_{2a};\ \underbrace{i<=n-1}_{2b},\ \underbrace{i++}_{2c})$$

Calculating Complexity of Algorithm

Statement#	Operations	Iterations	Subtotal
1	2	1	$2 \times 1 = 2$
2a	1	1	$1 \times 1 = 1$
2b	1	n	$1 \times n = n$
2c	2	$n-1$	$2 \times (n-1) = 2n-2$
3	3	$n-1$	$3 \times (n-1) = 3n-3$
4	1	1	$1 \times 1 = 1$

$$\text{Complexity} = 2 + 1 + n + 2n - 2 + 3n - 3 + 1$$
$$= 6n - 1$$

Algorithm:

```
sum=0
for i=1 to i=n
        for j=1 to j=n
                sum++
return sum
```

Iteration#1 [for outer loop]

```
sum=0
for i=1 to i=n
        for j=1 to j=n
                sum++
return sum
```

- Suppose n=9
- Iteration#1 [for outer loop]
  - Sum =0
  - Outer loop variable ‘i’ will be initialised with 1
  - Outer loop’s condition will be tested and outcome will be true since 1<=9
  - Inner loop variable ‘j’ will be initialised with 1 and outcome will be true since 1<=9
  - Inner loop will continue to execute till ‘j’ becomes 10 (because of increment by factor of 1) and condition will be false.
  - After exit from inner loop ,the outer loop will be executed

Iteration#2 [for outer loop]

```
sum=0
for i=1 to i=n
        for j=1 to j=n
                sum++
return sum
```

- Outer loop variable 'i' will be incremented by factor of i=2
- Outer loop's condition will be tested and outcome will be true since 2<=9
- Inner loop variable 'j' will be initialised with 1 and outcome will be true since 1<=9
- Inner loop will continue to execute till 'j' becomes 10 (because of increment by factor of 1) and condition will be false.
- After exit from inner loop ,the outer loop will be executed.

Iteration#9 [for outer loop]

- Outer loop variable ‘i’ will be incremented by factor of 1 so eventually i=9
- Outer loop’s condition will be tested and outcome will be true since 9<=9
- Inner loop variable ‘j’ will be initialised with 1 and outcome will be true since 1<=9
- Inner loop will continue to execute till ‘j’ becomes 10 (because of increment by factor of 1) and condition will be false.
- After exit from inner loop ,the outer loop will be executed

Iteration#10 [for outer loop]

- Outer loop variable 'i' will be incremented by factor of 1 so eventually i=10
- Outer loop's condition will be tested and outcome will be false since 10<=9
- Inner loop will not be executed.
- The compiler or interpreter directly jumps to the return statement which will be executed once.

Complexity Analysis

```
1  sum=0
2  for i=1 to i=n
3-         for j=1 to j=n
4-                      sum++
5- return sum
```

Statement#	Operations	Iterations	Sub-total
1	1	1	1
2a	1	1	1
2b	1	n+1	n+1
2c	2	n	2n
3a	1	n*1	n
3b	1	n*(n+1)	n^2+n
3c	2	n(n)	$2n^2$
4	2	n*n	$2n^2$
5	1	1	1
			$5n^2+5n+4$

Big O Notation

- Describes the limiting behaviour of the function when the argument tends towards a particular value or infinity , usually in terms of a simpler function.
- Big O notation gives the upper bound which will be determined by the most dominant term.

General rules

- Rule#1 (For loop):

The running time of ‘for’ loop is equal to the sum of running time of individual statements along with the running time of the ‘for’ loop

Algo:

1. Sum=0
2. for (i=1, i<=n, i++)
3. sum+=I
4. return sum

big O notation is O(n)

General rules

- Rule#2 (Nested 'For' loops):

The total running time of a statement inside a group of nested 'for' loop is equal to the running time of statement multiplied by the product of sizes of all 'for' loops

Algo:

1- sum=0

2- for i=1 to i=n # of iteration=n

3- for j=1 to j=n # of iteration=n

4- sum++

big O notation is $O(n^2)$

General rules

- Rule#3 (Consecutive Statements):

The running time of individual consecutive statements are added to calculate the running time of algorithm.

Algo:

```
1  sum=0                  # consecutive statement
2  for i=1 to i=n         # of iteration=n
3- sum++
```

big O notation is O(n)

General rules

Rule#4 (Conditional Statements):

The running time of an if/else statement is never more than the running time of test plus the larger of running time of S1 and S2.

Algo:

1. if (condition)
2. S1
3. else
4. S2

General rules

- Rule#5 :

An algorithm is O(log n) if it takes constant time to divide the problem size/data set by a fraction (which is usually half or ½).

The base of log is basically the number with which division of data set is performed.

Example for rule#5

```
int divide_sum (int n)
{
        int sum=0
        while (n>1)
         {
                sum+=n
                n/=2
          }
        return sum
}
```

Iteration	n=8
1	8
2	4
3	2
4	1

Topic: Array and its Operations

Introduction to Array

- Arrays are referred to as structured data types. An array is defined as **finite ordered collection of homogenous** data, stored in contiguous memory locations.
- The elements of array are accessed through index set containing 'n' consecutive numbers.
  - **finite** *means* data range must be defined.
  - **ordered** *means* data must be stored in continuous memory addresses.
  - **homogenous** *means* data must be of similar data type.

Introduction to Array

- In C language ,index of an array starts from 0 so if there are ‘n’ values then ‘n-1’ indexes will be used.
- Example:

Data: 2, 4, 6, 8 (n=4)

2	4	6	8
0	1	2	3

- Length or size of above array is 4.

Operations supported by Array

- Traversal
- Insertion
- Deletion
- Sorting
- Merging
- Search

Traversal Operation in an Array

Variables:

START: Initialised with starting index of array.

N: Number of elements in array

A:Variable for array

2	4	6	8
A[0]	A[1]	A[2]	A[3]

Algorithm:

1. START = 0
2. Repeat Step3 while (START<N)
3. Read A [START]
4. START = START + 1

Complexity of Traversal Operation in an Array

Algorithm:

1. START = 0
2. Repeat Step3 while (START<N)
3. Read A [START]
4. START = START + 1

2	4	6	8
A[0]	A[1]	A[2]	A[3]

Complexity Analysis

Statement#	Operations	Iterations	Sub-Total
1	1	1	1
2	1	N+1	N+1
3	1	N	N
4	2	N	2N

Summary

- Big O notation gives the upper bound or worst case scenario for a particular algorithm
- If an algorithm has the single loop then big O notation will be equal to the iterations of that loop
- If an algorithm has nested loops then big O notation will be equal to the number of iterations of outer loop multiplied by the number of iterations of inner loop.
- The big O notation of a constant will always be 1.
- In case of consecutive statements, addition is performed to find out the complexity.

Summary

- If an algorithm has conditional statement then complexity includes running time of conditional statement along with the maximum running time of nested code within if or else statement.
- The big O notation for an algorithm dividing the problem size by a constant factor will always be defined in terms of Log n.
- Array is the linear data structure in which data is stored in continuous memory location and can be accessed through indexes.