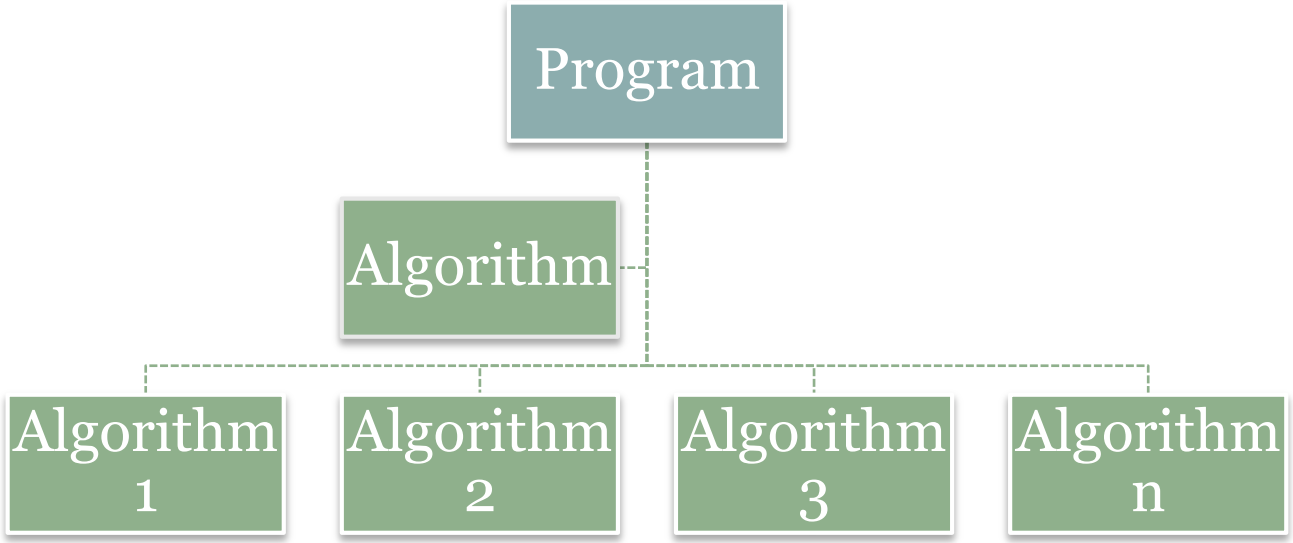
Complexity of Algorithms

BY CHAITALI PATIL

Assistant Professor,Dept. of Computer Engg.
K.K.Wagh Institute of Engineering Education and Research, Nashik

Analysis of algorithms

- How to decide which algorithm is best?



Analysis of algorithms

- Analysis involves measuring the performance of an algorithm. Performance is measured in terms of the following parameters:

1. Programmer's time complexity — Very rarely taken into account as it is to be paid for once
2. Time complexity — The amount of time taken by an algorithm to perform the intended task
3. Space complexity — The amount of memory needed to perform the task.

Algorithms are measured in terms of time and space complexity

Complexity of Algorithms

- Algorithms are measured in terms of time and space complexity.
- The time complexity of an algorithm is a measure of how much time is required to execute an algorithm for a given number of inputs and is measured by its rate of growth relative to standard functions.

Space Complexity

- Space complexity measurement, which is the space requirement of an algorithm, can be performed at two different times:

1. Compile time 2. Run time

Compile time space complexity is defined as the storage requirement of a program at compile time.

Space complexity = Space needed at compile time

This includes memory requirement before execution starts.

Run-time Space Complexity

- If the program is recursive or uses dynamic variables or dynamic data structures, then there is a need to determine space complexity at run-time.
- Memory requirement is the summation of the program space, data space, and stack space.

- Program space: This is the memory occupied by the program itself.
- Data space: This is the memory occupied by data members such as constants and variables.
- Stack space : This is the stack memory needed to save the function's run-time environment while another function is called. This cannot be accurately estimated since it depends on the run-time call stack

Time Complexity

- Time complexity T (P) is the time taken by a program P, that is, the sum of its compile and execution times, it is system-dependent.
- To determine the number of steps needed by a program to solve a particular problem instance in one of the following two ways
  - Introduce a new variable, count, into the program.
  - Manually compute the number of times each statement will be executed(frequency count.)

Frequency Count

- Int a=10;
- Print a

- Frequency count ?
- Total frequency is 2

Int a=0;

For i (0 to n)

A=a+1

Print a

- Frequency count ?
- Total frequency is 2n+3

We get frequency count in polynomial but we are interested in magnitude of the algorithm, i.e. the statement that have greatest frequency count

Best, Worst, and Average Cases

- The best case complexity of an algorithm is the function defined by the minimum number of steps taken on any instance of size n.
- The worst case complexity of an algorithm is the function defined by the maximum number of steps taken on any instance of size n.
- The average case complexity of an algorithm is the function defined by an average number of steps taken on any instance of size n.
- Each of these complexities defines a numerical function — time versus size.

Example Linear Search

Key 16

Key 58

Key 100

Key 03

Asymptotic Notations

- Asymptotic complexity helps us to quantify the performance of the algorithms.
- Big (O), omega (Ω), and theta (Θ) are the asymptotic notations used in this algorithmic analysis
- Big O formally represents the upper bound of the algorithm's time complexity as it suggests the maximum value or upper limit of the time taken by an algorithm to execute.(Longest amount of time, never more than)
- Omega (Ω) formally represents the lower bound of algorithms running time.(smallest amount of time, always greater than)
- Theta (Θ) used generally when best case and worst case requires same time

Find out the total frequency Count

Program 1

```
x := x + 1
```

Program 2

```
FOR i := 1 to n
DO
        x := x + 1
END
```

Find out the total frequency Count

Program 1

```
x := x + 1
```

– Frequency count is 1

Program 2

```
FOR i := 1 to n DO
        x := x + 1
END
```

–Total Frequency count is 2n-1

- Time complexity is O(n)
- Order of magnitude is n
- Efficiency f(n)=n

Frequency count

n

n-1

Total= n+n-1

=2n-1

Find Time Complexity

Program 3

```
FOR i := 0 to n
  DO
  FOR j := 0 to n
    DO
      x := x + 1
    END
  END
```

Find Time Complexity

Program 3	Frequency Count
FOR i := 0 to n	(n+1)
DO	
FOR j := 0 to n	n.(n+1)
DO	
x := x + 1	n.n
END	----------
END	$2n^2 + n + 2$

Time complexity is $O(n^2)$

Find Time Complexity

Program 4

```
FOR k := 0 to n
        FOR i := 0 to n
                FOR j := 0 to n
                        x := x + 1
                END
        END
END
```

Find Time Complexity

Program 4

```
FOR k := 0 to n
        FOR i := 0 to n
                FOR j := 0 to n
                        x := x + 1
                END
        END
END
```

Time complexity is $O(n^3)$

Frequency Count

$(n+1)$

$n.(n+1)$

$n.n.(n+1)$

$n.n.n$

$2n^3 + 2n^2 + 2n + 1$

Find Time Complexity

Program 5

```
Read n,m
Let r=1
For i = 1 to m do
R=r*n
Print r
stop
```

Find Time Complexity

Program 5	Frequency Count
Read n,m	1
Let r=1	1
For i = 1 to m do	m
R=r*n	m-1
Print r	1
stop	----------
	2m+2

Time complexity is O(m)

Find Time Complexity

Program 6

```
Do // consider i=1
{
        a++
        If(i==5)
                Break
        i++
}while(i<=n)
```

Find Time Complexity

Program 6

```
Do // consider i=1
{
        a++
        If(i==5)
                Break
        i++
}while(i<=n)
```

Frequency Count

5
5
1
5
5

21

Constant Time complexity

Complexity

- O(1) Constant (computing time)
- O(n) Linear (computing time)
- $O(n^2)$ Quadratic (computing time)
- $O(n^3)$ Cubic (computing time)
- $O(2^n)$ Exponential (computing time)
- O(log n) is faster than O(n) for sufficiently large n
- O(n log n) is faster than O(n2) for sufficiently large n

Find efficiency

Program 7

```
i=1
For (i <=500)
        i= i+1
        print I
end
```

Find efficiency

Program 7

```
i=1
For (i <=500)
        i= i+1
        print I
end
```

Frequency Count

```
1
501
500
500
--------------
1502
```

Time complexity O(n), efficiency f(n)=n

Find efficiency

Program 8

```
i=1
For (i <=500)
        i= i+2
        print i
end
```

Find efficiency

Program 8

```
i=1
For (i <=500)
        i= i+2
        print i
end
```

Frequency Count

```
1
501
250
250
--------------
1002
```

Time complexity O(n/2), efficiency f(n)=n/2

Find efficiency

Program 9

```
i=1
For (i <=250)
        i= i*2
        print i
end
```

Find efficiency

Program 9

```
i=1
For (i <=250)
        i= i*2
        print i
end
```

Frequency Count

1

9 (i=1,2,4,8,16,32,64,128,254)

8

8

26

$2^3=8$ $\log_2 8=3$

Time complexity $O(\log_2 n)$, efficiency $f(n)=\log_2 n$

Find efficiency

Program 10

```
i=250
For (i >=1)
        i= i/2
        print i
end
```

Find efficiency

Program 10

```
i=250
For (i >=1)
        i= i/2
        print i
end
```

Frequency Count

1

9

8

8

26

$2^3=8$ $\log_2 8=3$

Time complexity $O(\log_2 n)$, efficiency $f(n)=\log_2 n$

How to choose best algorithm ?

- $\text{Log } n < n < n \log n < n^2 < n^3 < \ldots < n^k < 2^n < n!$

Thank You