Algorithm Analysis

RM.Periakaruppan

Associate Professor

Dept. of Applied Mathematics and Computational Sciences

PSG College of Technology

Algorithm, Program and Data Structures

- An algorithm is the step-by-step instructions to solve the given problem.
- A program is an implementation of algorithm in a particular programming language.
- An appropriate choice of data structure is the key for designing an efficient algorithm.

Efficiency of algorithm

- Running time (Time Complexity)
- Memory (Space Complexity)

Time Complexity

- Empirical or Posteriori approach (dependent on machine, programming language etc)
- Theoretical or apriori approach(independent of hardware and software used)

Why Time Complexity is important?

- To analyze the running time of algorithm as the input size increases.
- For larger input size, it is important to analyze the running time of algorithm.

Understanding growth rate – Rice Grains and Chessboard

•

Growth rate of algorithm

- Algorithm1 for finding Prime Number
- bool isPrime1(int n)
  // Check from 2 to n-1
- for (int i = 2; i < n; i++)
- **if (n % i == 0)** **Total Number of divisions = ?**
- **return false;** (in case of prime number)
- return true;
- }
-

Growth rate of algorithm

- Algorithm1 for finding Prime Number
- bool isPrime1(int n)
  // Check from 2 to n-1
- for (int i = 2; i < n; i++)
- **if (n % i == 0)**
- **return false;**
- return true;
- }
-

Total Number of divisions = n-2

(in case of prime number)

Growth rate of algorithm

- Algorithm2 for finding Prime Number
- bool isPrime2(int n)
  // Check from 2 to sqroot(n)
- for (int i = 2; i <= sqroot(n); i++)
- **if (n % i == 0)** **Total Number of divisions = ?**
- **return false;**
- return true;
- }
-

Growth rate of algorithm

- Algorithm2 for finding Prime Number
- bool isPrime2(int n)
  // Check from 2 to sqroot(n)
- for (int i = 2; i < sqroot(n); i++)
- **if (n % i == 0)** **Total Number of divisions = sqroot(n)-1**
- **return false;**
- return true;
- }
-

Growth rate of algorithm

-

Input Size N	Time Taken by Alg I n-2	Time Taken by Alg2 sqroot(n)-1
11	9 ms	2 ms
101	99	9
1000003	10^6 ms = 10^3 sec = 16.6 minutes	10^3 ms = 1 sec
1000000009	10^10 ms = 10^7 sec = 115 days	10^5 ms = 100 sec =1.60 minutes

Frequency Count

- We can express the running time of algorithm as function of input size N.
- Example
- sum()   Frequency Count
- {
- S=0   1
- for(i=0;i<n;i++)   n + 1
- S= S+ A[i]   n
- return S   1
- }   Total   2n + 3 expressed as O(n)

Example 1

- for(i=1;i<=n;i++)

```
{
    statements;
}
```

Time complexity is

Example 1

- for(i=1;i<=n;i++)

```
{
    statements;
}
```

Time complexity is O(n)

Example 2

- for(i=n;i>=1;i--)

```
{
    statements;
}
```

Time complexity is

Example 2

- for(i=n;i>=1;i--)

```
{
    statements;
}
```

Time complexity is O(n)

Example 3

- for(i=1;i<=n;i=i+2)

```
{
    statements;
}
```

Number of times statements executed = ?

Time complexity is ?

Example 3

- for(i=1;i<=n;i=i+2)

```
{
    statements;
}
```

Number of times statements executed = n/2

Time complexity is O(n)

Example 4

- for(i=1;i<=n;i++)
- for(j=1;j<=n;j++)

```
{
    statements;
}
```

Time complexity is ?

Example 4

- for(i=1;i<=n;i++)
- for(j=1;j<=n;j++)

{

    statements;

}

Time complexity is O(n^2)

Example 5

- for(i=1;i<=n;i++)
- for(j=1;j<=i; j++)

```
{
    statements;
}
```

Time complexity is ?

Example 5

- for(i=1;i<=n;i++)
- for(j=1;j<=i; j++)

```
{
    statements;
}
```

Time complexity is O(n^2)

(ie. When i=1, j=1

i=2 , j=1+2

i=n, j=1+2+...n

No of times j executes=n(n+1)/2 approximated as n^2)

Example 6

- p=0
- for(i=1;p<=n;i++)

```
{
    p=p+i
}
```

Time complexity is ?

Example 6

- p=0
- for(i=1;p<=n;i++)

```
{
   p=p+i
}
```

Time complexity is O(sqrt(n))

Example 6

i	p
1	0+1=1
2	1+2 = 3
3	1+2+3=5
.	
k	1+2+3...+k = k(k+1)/2

- Loop will terminate, when p>n
- k(k+1)/2 >n or k^2 > n => k = sqrt(n)

Example 7

```
• for(i=1;i<=n;i=i*2)
{
    Stmt;
}
```

Time complexity is ?

Example 7

- for(i=1;i<=n;i=i*2)

```
{
    Stmt;
}
```

Time complexity is O(log n)

Example 7

'i' takes values 1,2,2^2,....2^k

Loop will terminate, when i>n

2^k > n

(i.e) 2 ^ k =n

=> k = log n

- Time complexity is O(log n)

Example 8

- for(i=n;i>=1;i=i/2)

```
{
    Stmt;
}
```

Time complexity is ?

Example 8

- for(i=n;i>=1;i=i/2)

{

Stmt;

}

Time complexity is O(log n)

(i takes values n, n/2, n/(2^2),...n/(2^k)

Loop will terminate when i<1 (ie) n/(2^k) <1

2^k > n => k= log n)

Example 9

- for(i=0;i*i<n;i++)

{

    Stmt;

}

Time complexity is ?

Example 9

```
• for(i=0;i*i<n;i++)
{
    Stmt;
}
```

Time complexity is O(sqrt(n))

(loop will terminate when i^2 >=n

i = sqrt(n))

Example 10

- for(i=1;i<=n;i++)
- {
- for(j=1;j<=n;j++)

```
  {
 Stmt;
   }
}
for(k=1;k<=n; k++)
{
Stmt;
}
```

Time complexity is ?

Example 10

```
• for(i=1;i<=n;i++)
• {
• for(j=1;j<=n;j++)
     {
   Stmt;
      }
}
for(k=1;k<=n; k++)
{
Stmt;
}
```

Time complexity is O(n^2) (we have f(n)= n^2 + n which is n^2)

Example 11

- p=0
- for(i=1;i<=n;i=i*2)
- {
- p++;
- }
- for(j=1; j<=p;j=j*2)
- {
- Stmt;
- }

Time complexity is ?

Example 11

- p=0
- for(i=1;i<=n;i=i*2)
- p++;
- for(j=1; j<=p;j=j*2)
- Stmt;

Time complexity is O(log log n)

(In first loop, value of p=log n

The second loop runs for log p, substituting p, we have

log log n)

Example 12

```
for(i=0;i<n;i++)
{
for(j=1;j<n;j=j*2)
{
    Stmt;
}
}
```

Time complexity is ?

Example 12

```
for(i=0;i<n;i++)
{
for(j=1;j<n;j=j*2)
{
    Stmt;
}
}
```

Time complexity is O(nlog n)

Inner for loop runs for log n times and outer for loop runs for n times,

=> total number of times is n log n

Commonly used time complexities

- $O(1)$ – Constant
- $O(\log n)$ - logarithmic
- $O(n)$ – linear
- $O(n \log n)$ – linear logarithmic
- $O(n^2)$- Quadratic
- $O(n^3)$- Cubic
- $O(n^k)$ – Polynomial (in general)
- $O(2^n)$ – Exponential time complexity (in general $O(k^n)$)

Comparison of time complexities

- O(1)<O(logn)<O(n)<O(nlogn)<O(n^2)..<O(n^k)<O(2^n)<(3^n)..<O(k^n)