

Algorithms

CSE 2415

Md. Hasan

April 2024

Introduction

Algorithm is a sequence of steps / step-by-step procedure to solve a problem.

Properties of Algorithm:

- Specific input
- Specific output
- Definiteness
- Finiteness
- Effectiveness

Time and Space Complexity

Examples:

Algorithm:	space complexity			Space complexity		
	cost	repeat	total	cost	repeat	total
int i, j;	1	1	1	i= 4	1	4
for(i = 0; i < n; i++){	1+1+1	1+(n+1)+n	2n+2	j= 4	1	4
for(j = 0; j < n; j++)	1+1+1	(1+(n+1)+n)+n	2n^2+2n	n= 4	1	4
printf(" %d ", i+j);	1	n^2	n^2			
}						
	F(n)= 3n^2 + 3n + 1			S(n)= 12		
	TC -> O(n^2)			SC -> O(1)		

Algorithm:

```
int i, j, n, A[i][j], B[i][j], C[i][j];
for(i = 0; i < n; i++){
    for(j = 0; j < n; j++)
        C[i][j] = A[i][j] + B[i][j];
}
```

space complexity

cost	repeat	total
1	1	1
1+1+1	1+(n+1)+n	2n+2
1+1+1	n+n(n+1)+n ²	2n ² +2n
1	n ²	n ²

F(n)= 3n² + 4n + 3
TC -> O(n²)

Space complexity

cost	repeat	total
i=4	1	4
j=4	1	4
n=4	1	4
A[] []	4*n*n	4n ²
B[] []	4*n*n	4n ²
C[] []	4*n*n	4n ²

S(n)= 12n² + 12
SC -> O(n²)

Algorithm:

```
int i, n;
for(i = 0; i < n; i++)
    printf(" %d ", 2*i);
```

space complexity

cost	repeat	total
1	1	1
1+1+1	1+(n/2)+1+(n/2)	n+2
1	n/2	n/2

F(n)= (3n/2) + 3
TC -> O(n)

Space complexity

cost	repeat	total
i=4	1	4
n=4	1	4

S(n)= 8
SC -> O(1)

Algorithm:

```
int p=0, i, n;
for(i = 1; i <= p; i++)
    p+=1;
```

Step analysis:

i	p
0	0+1
1	0+1+2
2	0+1+2+3
3	0+1+2+3+4
.	.
.	.
.	.
k	0+1+2+3+4+...+k = $k*(k+1)/2$

assume , $p > n$ where step number is k and $p = k*(k+1)/2$

$$\begin{aligned} k(k+1)/2 &= n \\ \Rightarrow k(k+1) &= 2n \\ \Rightarrow k^2 + k - 2n &= 0 \\ \Rightarrow k^2 &= 2n \quad [\text{removed } k \text{ as } k^2 > k] \\ \Rightarrow k &= \sqrt{2n} \\ \Rightarrow k &= \sqrt{2} * \sqrt{n} \end{aligned}$$

thus $O(\sqrt{n})$

```
int i, n;
for(i =1; i<n; i*=2)
    printf("%d", i);
```

Step Analysis:

steps	$2*i$	
1	1	
2	2	
.	.	
.	.	
.	.	
k	2^k	[actual value is $2^{(k-1)}$]

assume, code stopped at step k where , $i = 2^k$
 $2^k > n$
 $\Rightarrow 2^k = n$
 $\Rightarrow \log 2^k = \log n$
 $\Rightarrow k = \log n$ [$\log 2^x = x$, here base is always 2 cause of binary]
 $\rightarrow O(\log n)$

```
int i, n;
for(i = n; i > 1 ; i/=2)
    printf("%d", i);
```

Step Analysis:

steps	$i/2$
1	n
2	$n/2$
.	.
.	.
.	.
k	$n/2^k$

$n/2^k < 1$
 $\Rightarrow 2^k < n$
 $\Rightarrow 2^k = n$
 $\Rightarrow \log 2^k = \log n$
 $\Rightarrow k = \log n$

```
int p=0, i;
for(i =0; i<n; i*=2)
    p++;          -----> O(log n)
for(i=p; i>1; i/=2)    '----> p = log n
    printf("%d", i);    '----> O(log log n)
```

Time Complexity Cheat Sheet

Constant TC	- $O(1)$	-> for(i=0; i<k; i++)
	- $O(\sqrt{n})$	-> for(i=0; p<n; i++)
Linear TC	- $O(n)$	-> for(i=1; i<n; i++)
		-> for(i=0; i<n; i+=2)
		-> for(i=n; i>n; i-=5)
Logarithm TC	- $O(\log n)$	-> for(i=0; i<n; i*=2)
		-> for(i=n; i>1; i/=2)
	- $O(\log_a n)$	-> for(i=1; i<n; i*=a)
Polynomial TC	- $O(n^2)$	-> for(--) for(--)
	- $O(n^3)$	-> for(--) for(--) for(--)
Exponential TC	- $O(2^n), O(n^n)$	-> Any recursive function

Recursion:

Substitution method:

- > It solves all types of recursive problems
- > It always gives right answers
- > Takes more time than Master method

Master method:

- > It solves Specific types of recursive problems
- > Format: $T(n) = aT(n/b) + f(n)$; where $a \geq 1$ and $b > 1$

Different cases:

1. $f(n) < n \log_b a$;
 $T(n) = (\theta) (n \log_b a)$
 2. $f(n) = n \log_b a$;
 $T(n) = (\theta) (n \log_b a \cdot \log n)$
 3. $f(n) > n \log_b a$;
 $T(n) = (\theta) (f(n))$
-

Examples: $EQ^n :$

$$T(n) = \begin{cases} t(n-1) + n & , n > 1 \\ 1 & , n \leq 1 \end{cases}$$

 $Sol^n :$ We have,

$$T(n) = t(n-1) + n \quad \text{---(1)} \tag{1}$$

Replace n by $n-1$ in equation (1):

$$T(n-1) = t(n-2) + (n-1) \quad \text{---(2)} \tag{2}$$

Putting equation (2) in equation (1) we get,

$$T(n) = t(n-2) + (n-1) + n \quad \text{---(3)} \tag{3}$$

Similarly, replacing n by $n-2$ in equation (1) we get,

$$T(n-2) = t(n-3) + (n-2) \quad \text{---(4)} \tag{4}$$

Putting equation (4) in equation (3) we get,

$$T(n) = t(n-3) + (n-2) + (n-1) + n \quad \text{---(5)} \tag{5}$$

So, the general equation may be,

$$T(n) = t(1) + 2 + 3 + \dots + (n-2) + (n-1) + n \quad \text{---(6)} \tag{6}$$

Given $T(1) = 1$ we get from equation (6),

$$\begin{aligned} T(n) &= t(1) + 2 + 3 + \dots + (n-2) + (n-1) + n \\ &= 1 + 2 + 3 + \dots + (n-2) + (n-1) + n \\ &= \frac{n(n+1)}{2} \\ &= \frac{n^2}{2} + \frac{n}{2} \end{aligned}$$

Here, n^2 is the time complexity. Thus, the answer is $O(n^2)$.

$EQ^n :$

$$T(n) = \begin{cases} t(n/2) + c & , n > 1 \\ 1 & , n \leq 1 \end{cases}$$

$Sol^n :$ We have,

$$T(n) = t\left(\frac{n}{2}\right) + c \quad \text{---(1)} \quad (7)$$

Replace n by $n/2$ in equation (1):

$$T\left(\frac{n}{2}\right) = t\left(\frac{n}{4}\right) + c \quad \text{---(2)} \quad (8)$$

Putting equation (2) in equation (1) we get,

$$T(n) = t\left(\frac{n}{4}\right) + c + c \quad \text{---(3)} \quad (9)$$

Similarly, replacing n by $n/4$ in equation (1) we get,

$$T\left(\frac{n}{4}\right) = t\left(\frac{n}{8}\right) + c \quad \text{---(4)} \quad (10)$$

Putting equation (4) in equation (3) we get,

$$T(n) = t\left(\frac{n}{8}\right) + c + c + c \quad \text{---(5)} \quad (11)$$

So, the general equation may be,

$$T(n) = t\left(\frac{n}{2^k}\right) + kc \quad \text{---(6)} \quad (12)$$

Assume that in the k -th step, $\frac{n}{2^k} = 1$

$$\begin{aligned} \Rightarrow \frac{n}{2^k} &= 1 \\ \Rightarrow n &= 2^k \\ \Rightarrow \log n &= \log 2^k \\ \Rightarrow k &= \log n \end{aligned}$$

Using the value of k in equation (6), we get,

$$T(n) = t(1) + kc = 1 + (\log n) \cdot c \quad (13)$$

Here, $t(1)$ is the base case complexity. Thus, the answer is $O(\log n)$.

$EQ^n :$

$$T(n) = \begin{cases} 2t(n/2) + n & , n > 1 \\ 1 & , n = 1 \end{cases}$$

$Sol^n :$

We have,

$$T(n) = 2t\left(\frac{n}{2}\right) + n \quad \text{---(1)} \quad (14)$$

Replace n by $n/2$ in equation (1):

$$T\left(\frac{n}{2}\right) = 2t\left(\frac{n}{4}\right) + \frac{n}{2} \quad \text{---(2)} \quad (15)$$

Putting equation (2) in equation (1) we get,

$$T(n) = 2\left(2t\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \quad (16)$$

$$T(n) = 4t\left(\frac{n}{4}\right) + 2n \quad \text{---(3)} \quad (17)$$

Similarly, replacing n by $n/4$ in equation (1) we get,

$$T\left(\frac{n}{4}\right) = 2t\left(\frac{n}{8}\right) + \frac{n}{4} \quad \text{---(4)} \quad (18)$$

Putting equation (4) in equation (3) we get,

$$T(n) = 4\left(2t\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n \quad (19)$$

$$T(n) = 8t\left(\frac{n}{8}\right) + 3n \quad \text{---(5)} \quad (20)$$

So, the general equation may be,

$$T(n) = 2^k t\left(\frac{n}{2^k}\right) + kn \quad \text{---(6)} \quad (21)$$

Given $T(1) = 1$,

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$\text{we get , } n = 2^k \quad \text{---(a)}$$

$$\log n = \log 2^k$$

$$\text{or, } k = \log n \quad \text{---(b)}$$

Since $n = 2^k$, we have $k = \log n$

$$\text{thus, } T(n) = nT\left(\frac{n}{n}\right) + n \log n \quad [\text{using equations (a) and (b)}]$$

$$T(n) = n + n \log n$$

$$\text{thus, we get } TC = O(n \log n)$$

Master Method Examples:

EQ^n :

$$T(n) = \begin{cases} 2t\left(\frac{n}{2}\right) + n & , n > 1 \\ 1 & , n = 1 \end{cases}$$

Sol^n :

$$a = 2, \quad b = 2, \quad f(n) = n \\ n^{\log_b a} = n^{\log_2 2} = n^1 = n$$

$$f(n) = n^{\log_b a}$$

$$TC = \theta(n^{\log_2 2} \cdot \log n) \\ = \theta(n \cdot \log n)$$

EQ^n :

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + c & , n > 1 \\ 1 & , n \leq 1 \end{cases}$$

Sol^n :

$$a = 1, \quad b = 2, \quad f(n) = c \\ n^{\log_b a} = n^{\log_2 1} = n^0 = 1$$

$$f(n) = n^{\log_b a}$$

$$TC = \theta(n^{\log_2 1} \cdot \log n) \\ = \theta(1 \cdot \log n) \\ = \theta(\log n)$$

EQ^n :

$$T(n) = \begin{cases} T(n-1) + n & , n > 1 \\ 1 & , n \leq 1 \end{cases}$$

Sol^n :

$$a = 1, \quad b = 1, \quad f(n) = n$$

Since, b is not greater than a, we cant use master method to find the complexity.

Divide And Conquer Method:

3 step to solve level of problem:

1. Divide:

Divide the problem into problems into a number of subproblems that are similar to the problem.

2. Conquer:

Conquer the subproblems by solving them recursively. If the subproblem's size is small enough, just solve it in straight forward manner.

3. Combine:

Combine the solutions of the subproblems to get the solution of the original problem.

Example:

For MergeSort it works as follows:

1. Divide the n -element array into two $n/2$ elements to Sort.
 2. Sort the two subsequences recursively using MergeSort.
 3. Merge the two sorted subsequences.
-

Time Complexity function Comparison:

Big O cheat sheet

TC class :

$$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \dots < 2^n < 3^n < \dots < n^n$$

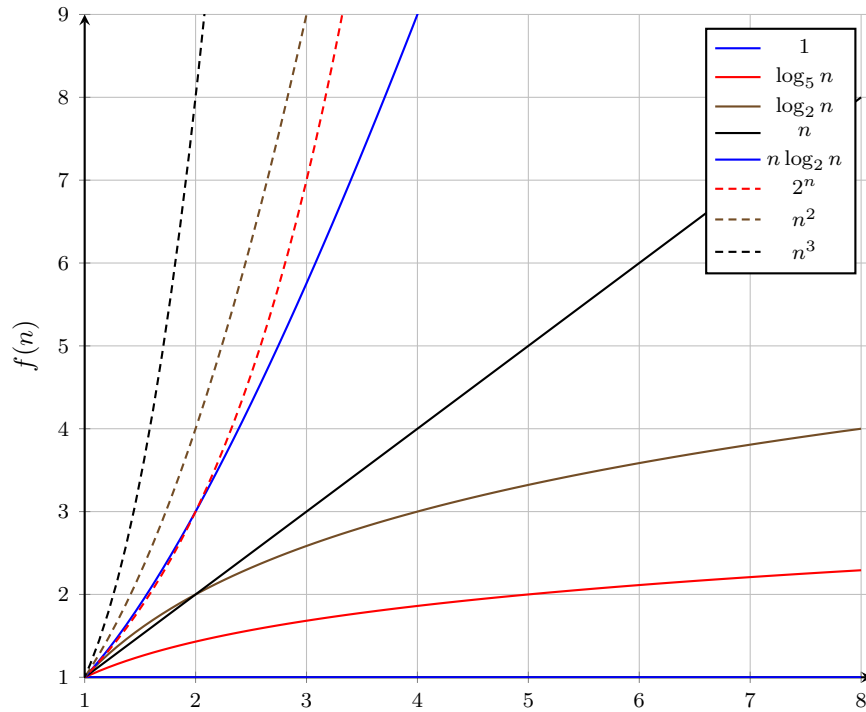


Figure 1: Function Complexity