Wednesday, 12 August 2020 8:00 AM

Algorithms

A finite sequence of well-defined instructions to solve an problem. Typically problem is [class of problems] or [specific problem].

→ Algorithm describes the ordering that will be taken on the input problem.

→ Infinitely correct algorithms will fail the same algorithmic problem.

→ good algorithm !!

Anything that is optimal is good.

what is optimal ?

Efficiency

[→ running time
→ space used]

→ How should we measure the running time of the algorithm ??

① Experimental Analysis

→ write a program that implements the algorithm

→ run the program after varying inputs.

Drawbacks

① Hardware & software dependent

② It is necessary to implement the test.

② Asymptotic analysis

Goal: to simplify analysis of running time by getting rid of details, which may be affected by specific hardware & software evaluation & environment.

(O(3n) - 0 notation)

It is a simplified analysis of program efficiency

Q1: complexity in terms of input size (n)

Q2: machine independent [abstract the efficiency of the algorithm from the machine they run on]

Q3: count computer steps.

(O(3n)) can be used to analyze their time & space.

Types of measurements

Three are 3 complexities of an algorithm

① worst case
② best case
③ average case

→ when we talk about O(3n) notation we talk about worst case.

Pseudocode

① (constant time algorithm)

```
x = 6 + (15 * 20);
```

→ independent of size N.

① 15 → 30
② 6 → 240
③ x → 246

O(1) ⇔ O(2)

② (linear time algorithm)

```
x = 5 + (15 * 20);
y = 4 - x;
print (x-y);
```

→ O(1)
→ O(n)
→ O(1)

O(2) + O(3) + O(1) ⇔ O(n) ⇔ O(1) ⇔ O(3)

③

```
for x in range (0,n):
    print x;
```

O(n)

TC → O(n)

n = 5	n = 9
0	0
1	1
2	2
3	3
4	4
	5
	6
	7
	8

value

→ linear complexity

④

```
y = 5 + (15 * 20);
for x in range (0,n):
    print x;
```

→ O(1)
→ O(n)
→ O(1)

TC → O(1) + O(n)
→ O(n)
⇒ >> large

⑤

```
for x in range (0,n):
    for y in range (0,n):
        print (x + y);
```

quadratic complexity

n = 4

0 + 0	1 + 0	2 + 0	3 + 0
0 + 1	1 + 1	2 + 1	3 + 1
0 + 2	1 + 2	2 + 2	3 + 2
0 + 3	1 + 3	2 + 3	3 + 3

n = 4 → print → 16 values
n = 5 → print → 25 values

TC → O(n²)

⑥

```
x = 5 + (15 * 20);
for x in range (0,n):
    print (x);
for x in range (0,n):
    print ("x + y");
```

→ O(1)
→ O(n)
→ O(n)

TC → O(1) + O(n) + O(n)
→ O(n)
⇒ >> large

→ remove lower order stuff

⑦

```
if x > 0:
    print x
else if x < 0:
    print (-x)
else:
    print ("zero")
```

→ O(n²)

O(1) → constant time →

⑧

→ c = n;
→ n = 0;
→ while (c > 0) {
    n = n + 1;
    c = c / 2;
}

log₂ steps

n = 8
c = 8/2 = 4
c = 4/2 = 2
c = 2/2 = 1
c = 1/2 = 0

n = 8 + 1 = 9

TC → O(log n)

→ Proper Exponentiation.com

9 Print

n → n²
calc → n²

x = 3, n = 9
x^n = 3^9 = 81

by ①

$x^n \rightarrow x$

$3^9 \rightarrow 3$

$3^{9-1} * 3 * 3$

$3^{9-2} * 3 * 3 * 3$

3^9

$3 * 3 * 3 * 3$

4 steps

O(n)

ans = 1;
for (i=0; i<n; i++) {
    ans = ans * x;
}

O(n)

by ②

→ even power

$3^4 * 3^4$

$3^2 * 3^2$

$3^1 * 3^1$

$3^1 * 3^1 * 3$

$3^2 * 3^2 * 3$

$3^4 * 3^4 * 3$

x^n

calc → $x^{n/2} * x^{n/2}$ (log n) steps

even → ans = ans * ans

odd → ans = ans * ans * x

$3^4 + 3^4 \rightarrow 3^8$

$3^2 * 3^2 = 3$

$3^2 * 3^2 * 3$

$3^2 * 3^2 * 3 = 3^5$

$3^2 * 3^2 * 3 = 3^5$

3^5

$3^5 * 3^5 =