Wednesday, 22 July 2021 8:24 PM

Algorithm

A set of [finite sequence] of well-defined instructions [computer] [step-by-step] to solve a [class of problem] or any problem in computation.

→ Algorithm describes the action and with in an finite no. the input problem.

→ Infinite may would algorithms exist for the same solve algorithmic problem.

⇒ good algorithm ??

Anything that is efficient is good.

what is efficient ??

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 with varying inputs

n = 2 sec.

10n → t = 2 sec.

Drawbacks

1. hardware & software dependent
2. It is necessary to implement and test.

② Asymptotic analysis

goal is simply analyze the running time by going "high level"

→ by ignoring implementation and hardware.

(eg. 0 notation)

It is simplified analysis of program efficiency.

1. complexity is known by input size ⓝ
2. machine independent [abstract one efficiency of an algorithm from true machine long run on]
3. better computer chips.

(eg. 5) can be used to analyze data base & space.

Type of measurements

There are a couple of ways to look at an algorithm efficiency

1. worst case
2. best case
3. average case

→ when we talk about eg. 0 notation, we talk about worst case.

Problems

① constant time algorithm

$x = 2 + (15 * 20)$;

→ independent of size ⓝ

1. $15 * 20$
2. $+ 30$
3. $= 302$

$O(1) \Rightarrow O(1)$

② $x = 2 + (15 * 20)$; → $O(1)$

print x;

$x = 302$

$O(1)$

$x = 2 + (15 * 20)$; → $O(1)$

print (x + y);

$x = 302$

$y = 15$

$O(1) \Rightarrow O(1)$

③

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

$O(n)$

$TC \rightarrow O(n)$

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

→ linear complexity

④

$y = 2 + (15 * 20)$; → $O(1)$

for x in range (0,n):

print y; → $O(n)$

$TC \rightarrow O(1) + O(n)$

$\rightarrow O(n)$

n → large

⑤

n = 3	n = 5
0,0	0,0
0,1	1,0
1,0	2,0
1,1	3,0
2,0	4,0
2,1	⋮
	⋮

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

$TC \rightarrow O(n^2)$

→ quadratic complexity

⑥

$x = 2 + (15 * 20)$; → $O(1)$

for x in range (0,n):

print (x); → $O(n)$

for x in range (0,n):

for y in range (0,n):

print (x + y); → $O(n^2)$

$TC \rightarrow O(1) + O(n) + O(n^2)$

$\rightarrow O(n^2)$

n → large 10^3

→ Number order terms

⑦

if n > 0:

print (5)

elif n < 0:

print (n)

else

print (n*n) → $O(1)$

⑧

Code	Output
1	1
2	2
3	3
4	4
5	5
⋮	⋮
hoo	hoo

n	k
1	1
2	2
3	3
4	4
5	5
⋮	⋮
n+1	n+1

$O(n)$

$O(\log n)$

⑨ Type checking error.

```
for (i,n = (int, int)) {
    for (j,k = (int, int)) {
        for (l,m = (int, int)) {
            print (i,j,k,l,m);
        }
    }
}
```

$O(n^3)$

$O(\log n)$

⑩

Phone

Pin →

n = 3

$n^3 = 3^3 = 27$

①

3^1

3^2

3^3

$3^4 = 3^3$

3^5

$3^6 = 3^3$

3^7

$3^8 = 3^3$

$3 * 3 * 3 = 3^3$

⑪ Given a string (s), count all palindromic substrings.

babad

b a b a d

substrings

b	a	b	a	d
bab	aba	bab	aba	
babab	abab			

output → 7

→ palindromes

→ nodes same, forward & backward

① minimum changes

n a w → n a w

② n a w a w

odd

even

n a w a w

a b b a

for a string length → n

max = ⌊ n/3 ⌋

⑬

a

a a

a

a a a

a a a a

a a a a a

a a a a a a