COP 3502 2/21/22

Class Wed: Substitute
3/14 - Brandon will sub
Quiz 2 → Will post sols, after tonight

Upto Q1 - Dyn Mem Alloc
~~Up~~ (Q1, Q2] - basic data structs (linked lists)
recursion
abstract data structs (stacks, queues)

(Q2, Q3] - Algorithm Analysis (Before SB)
Sorting (Finish > S.B)
Advanced Data Struct

High Level goal of algorithm analysis:

To analyze potential solution ideas to estimate (a) how much time they'll take to run
(b) how much memory they'll use.

1st Tool: Big-Oh notation

When I ~~saw~~ say something like $O(n^2)$, this refers to any function that is roughly equal to cn^2 for some const. c as n grows large

All of these functions are $O(n^2)$:

$$\left.\begin{array}{l} 2n^2 \\ \frac{n(n+1)}{2} \\ 3n^2 - \frac{n}{\lg n} + 7 \end{array}\right]$$

Dominating term of function is $c \cdot n^2$ for some constant.

To find the Big-Oh of a function, evaluate each term added, find "biggest" term ignore multiplicative constants.

$$\underbrace{1}_{\text{small}} \leq \underbrace{\lg n}_{\text{small}} < \underbrace{n^k}_{\substack{k > 0 \\ \text{poly}}} < 2^n < 3^n \ldots < n! \ldots < n^n$$

$\lg n^2$
$= 2 \lg n$

$$\boxed{O(\lg n^2) = O(\lg n)}$$

$$\log_4 n = \frac{\log_2 n}{\log_2 4} = \frac{1}{2} \log_2 n$$

$$n < n \lg n < \underbrace{n^2}_{n \cdot n}$$

$$n < n \lg n < n\sqrt{n} < n^2$$

1. Given a function in a variable "reduce" it to its Big-Oh equivalent.
2. Given an algorithm determine its Big-Oh run-time
3. Use #2 to make actual predictions about runtime

How I'll test

Two types of Problems

1. Given run-time of an algorithm, make predictions about runtimes.
2. Given pseudocode, determine its Big-Oh run time.

Variables/Inaccuracies in Predicting Run Times

Computer Architectures Different

Implementation Differences.

Small Statements take diff amts of time.

Why we use Big-Oh?

Too many small differences to make exact predictions. Rather best we can do is within some constant multiplicative factor

$\Longrightarrow$ Calculate a Big-Oh # of steps in terms of the input instead of an exact # steps.

An algorithm to sort n numbers runs in $O(n^2)$ time. On an input of 10,000 numbers, it took 1 second. How long would it be expected to take to sort 30,000 numbers?

Let $T(n)$ be run-time of alg. where n = input size

Then, $T(n) = cn^2$ for some const c.

$$T(10^4) = c(10^4)^2 = 1 \text{ sec}$$

$$c = \frac{1}{10^8} \text{ sec}$$

$$T(3\times10^4) = \frac{1}{10^8} \times (3\times10^4)^2 \text{ sec}$$

$$= \frac{1}{10^8} \times 9\times10^8 \text{ sec} = \boxed{9 \text{ sec}}$$

Algorithm to sort n ints ~~takes~~ runs in $O(n\lg n)$

For $n=2^{20}$ algorithm takes 80 ms. How long should it take to sort $n=2^{25}$ values?

$$T(2^{20}) = c\cdot 2^{20}\lg 2^{20} = 80 \text{ ms}$$

$$\boxed{T(n) = cn\cdot\lg n}$$

$$c = \frac{80\text{ms}}{2^{20}\cdot 20\lg 2} = \frac{4}{2^{20}\lg 2}\text{ ms}$$

$$T(2^{25}) = \frac{4\text{ms}}{\cancel{2^{20}}\lg 2} \times \cancel{2}^{25-20} \times \lg 2^{25} = \frac{4\text{ms}\cdot 2^5 \cdot 25\cancel{\lg 2}}{\cancel{\lg 2}}$$

$$= (100\text{ms})\times 32 = 3200 \text{ ms}$$

$$= \boxed{3.2 \text{ sec}}$$

Run time of code segments

(1)
```
for (int i=0; i<n; i++) {
    // Simple Stmt
}
```
$O(n)$

(2)
```
for (int i=0; i<n; i++) {
    for (int j=0; j<n; j++) {
        // Simple Stmts
    }
}
```
$O(n^2)$

i	j
0	0, 1, 2, ..., n-1
1	0, 1, 2, ... n-1
⋮	
n-1	0, 1, 2, ... n-1

(3)
```
for (int i=0; i<n; i++) {
    for (int j=0; j<m; i++) {
        // Simple Stmt
    }
}
```
$O(nm)$

→ Run time can be based on more than 1 parameter.

(4)

```
int low = 0, high = n-1;
while (low <= high) {
    int mid = (low + high)/2;
    if (value < array[mid])
        high = mid - 1;
    else if (value > array[mid])
        low = mid + 1;
    else
        return 1;
}
return 0;
```

$O(\lg n)$

Diff high-low gets divided by 2 each time. Let the loop run k times, then

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

any repeated dividing of n until 1 OR repeated doubling of 1 until n is $O(\lg n)$.

(5)

```
int sum = 0;
for (int i = 0; i<n; i++)
    for (int j = i+1; j<n; j++)
        if (array[i] > array[j])
            sum++;
```

i	j	
0	1,2,3,.. n-1	(n-1) +
1	2,3,4,.. n-1	(n-2) +
2	3,4,5... n-1	(n-3) +
⋮		+
n-1	~~0~~	(0)

$$\sum_{i=0}^{n-1} i = \frac{(n-1)(n)}{2}$$

$$= \frac{1}{2}n^2 - \frac{1}{2}n$$

$$= O(n^2)$$

(6)

```
int i, j = 0;
for (i = 0; i<n; i++) {
    while (j<n && array[i][j] == 1)
        j++;
}
return j;
```

at most n times

max # times n in total

run time = n + n = $\boxed{O(n)}$