COP 3502 10/2/23

Completed Summations

Run Time Predictions:

Can't be perfect

lots of differences btw machines

hardware, different executables

Instead making exact predictions we just want to be "in the right ballpark"

Can I predict # simple instructions to within a constant multiplicative factor?

Yes

A function $f(n) = O(g(n))$ <u>iff</u> "if and only if"

for all $n > n_0$ there exists a constant c,

suchthat $f(n) \le c \cdot g(n)$.

$$3n^2 - n = O(n^2)$$

because $\cancel{} - n \le 3 \cdot n^2$, for all $n > 0$.

$3n^2$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \le c, \text{ for some const } c.$$

Intuitive way: $f(n) = \boxed{3n^3} - 2n^2 + 7n + 18$

grab "biggest" term, drop constants $f(n) = O(n^3)$

Functions slowest to fastest: $O(1), O(\lg \lg n), O(\lg n), O(\lg^2 n), O(\sqrt{n})$
$O(n), O(n \lg n), O(n \lg^2 n), O(n^2), O(n^3),$
$O(n^k), O(2^n), O(n!), O(n^n)$
$k \geq 3$

Big-Oh is upper bound technically

$n^2 = O(n^{100})$ and $n \lg n = O(2^n)$

these are true but not very helpful.

$f(n) = \Theta(g(n))$ c_1, c_2

for all $n > n_0$ there exists a constants > 0

Such that $f(n) \leq c_1 g(n)$ and $f(n) \geq c_2 g(n)$,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = c, \quad c > 0.$$

2 types of problems

1. Run-time prediction exercise
2. Given a code segment or description of an algorithm determine its big-oh run-time and/or use of memory.

An algorithm with input size n runs in $O(n^2)$ time. On an input with $n = 30{,}000$ the algorithm runs in 100 ms. How long in seconds will it take on an input of size 120,000?

Let $T(n)$ be the amt of time the alg takes w/ input size n,

$$T(n) = cn^2; \quad \text{(Note: Simplification)}$$

$$T(30000) = c\,(30000)^2 = 100\text{ ms}$$

$$c = \frac{100\text{ms}}{(30000)^2}$$

$$T(120000) = \frac{100\text{ms}}{(30000)^2} \times (120000)^2 \qquad a^2b^2 = (ab)^2$$

$$= 100\text{ms} \times \left(\frac{120000}{30000}\right)^2$$

$$= 100\text{ms}\ (4)^2$$

$$= 1600\text{ms}$$

$$= \boxed{1.6\text{ sec}}$$

Algorithm A processes data on an $n \times m$ grid in $O(n \lg m)$. When $n = 1000$ and $m = 2^{15}$, the algorithm takes 30 ms. How long will it take on a grid of size $n = 750$ and $m = 2^{20}$?

$$T(n, m) = c \cdot n \lg m$$

$$T(1000, 2^{15}) = c \cdot 1000 \lg 2^{15} = 30\,\text{ms}$$

$$= c = \frac{30\,\text{ms}}{1000 \cdot 15 \lg 2}$$

$$T(750, 2^{20}) = \frac{30\,\text{ms}}{1000 \cdot 15 \lg 2} \times 750 \lg 2^{20}$$

$$= \frac{\overset{2}{\cancel{30}}\,\text{ms} \times \overset{3}{\cancel{750}} \times \overset{5}{\cancel{20}} \times \cancel{\lg 2}}{\underset{4}{\cancel{1000}} \cdot \cancel{15} \cancel{\lg 2}} \quad \cancel{\text{ms}}$$

$$= \boxed{30\,\text{ms}}$$

Analyzing Code Segments

```
int sum=0;
for (int i=0 ; i<n; i++) {
    sum++; //simple stmt
}
```

$O(n)$

```
for (int i=0 ; i<strlen(s); i++)
    printf("%c", s[i]);
```

this function loops to end of the string $O(|s|)$

$O(|s|^2)$

If n is length of s then, $O(n^2)$.

```
int len = strlen(s);
for (int i=0; i<len; i++)
    printf("%c", s[i]);
```

$O(n)$

$O(n)$ $\boxed{O(n)}$

```
for (int i=0; i<n; i++) {
    for (int j=0; j<m; j++){
        //Const work no Δ i,j.
    }
}
```

$O(nm)$

```
for (int i = 0; i < n; i++ ) {
  for (int j = 0; j < i; j++) {
     // const work no Δ i,j
  }
}
```

0, 1, 2, ...

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

$$\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=0}^{n-1} i$$

$O(n^2)$

```
i = 0, j = 0
while (i < n) {
   while (j < n && g[i][j] == 1)
      j++;
   i++;
}
```

At most j can be ~~HAT~~ incremented n times

i can be incremented n times

$$\text{Run-time} \leq 2 \cdot n = O(n)$$