COP3502 10/3/23

1. Big-Oh, Big-Theta
2. Predicting Run-Times → Live Demo also
3. Analyzing Code Segments

→ $f(n) = O(g(n))$ iff There exist constants c and n_0 such that for all $n > n_0$

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

$f(n) = 3n^2 - 7$, $g(n) = n^2$ $\quad 3n^2 - 7 = O(n^2)$.

Consider $n_0 = 1$, $c = 3$

$f(n) = 3n^2 - 7$, $\quad c \cdot g(n) = 3 \cdot n^2$

Prove $3n^2 - 7 \le 3n^2$ ✓

If $f(n) = f_1(n) + f_2(n) + f_3(n)$, one of these 3 parts will "dominate" the function.

Intuitively, $f(n) = MAX(O(f_1(n)), O(f_2(n)), O(f_3(n)))$

→ There exists a constant c such that

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

$n = O(2^n)$ is true stmt.

$f(n) = \Theta(g(n)) \iff$ There exist constants c_1, c_2 and n_0 such that $0 < c_1 < c_2$ and for all $n > n_0$

$$c_1 g(n) \le f(n) \le c_2 g(n).$$

For CS1, we'll assume that if an algorithm runs in $O(f(n))$ time then for input size n, it will take $c \cdot f(n)$ sec/ms to complete for some constant c.

Why do we use this tool?
It's near impossible to predict the exact # of clock cycles a piece of code will take.
For prediction purposes, the best we can hope for is calculating a run-time within a constant multiplicative factor.

$3n^2 + 5n + 2$ stmts
as n grows large, this is the main piece of the pie

list of functions from smallest to largest "growing"

$O(1)$, $O(\lg \lg n)$, $O(\lg n)$, $O(\lg^2 n)$, $O(\sqrt{n})$, $O(\sqrt{n} \lg n)$, $O(n)$, $O(n \lg n)$, $O(n \lg^2 n)$, $O(n\sqrt{n})$, $O(n^2)$, Poly, $O(2^n)$, $O(3^n)$, $O(n!)$, $O(n^n)$

Algorithm processes an image with n pixels ~~in~~ runs in $O(n\sqrt{n})$ ~~time~~ time. for $n=10^4$ the algorithm takes 15ms to complete. How many seconds will it take to complete on an image w/ $n=10^6$ pixels?

Let $T(n) = \cancel{c\sqrt{}}\ cn\sqrt{n}$ be the run-time of the alg. on n pixels.

$$T(10^4) = c \cdot 10^4\sqrt{10^4} = 15\text{ ms}$$

$$= c \cdot 10^4 \cdot 10^2 = 15\text{ms}$$

$$c = \frac{15\text{ms}}{10^6}$$

$$T(10^6) = \frac{15\text{ms}}{\cancel{10^6}} \times \cancel{10^6}\sqrt{10^6}$$

$$= \underline{15\text{ms}} \times \underline{10^3}$$

$$= 15{,}000\text{ ms}$$

$$= \boxed{15\text{ sec}}$$

Inversion

3, 2, 6, 1, 5

an ~~##~~ inversion is an ordered pair (i, j) with $i<j$ but $a[i] > a[j]$

(3,2) (6,1)
(3,1) (6,5)
(2,1)

```
res = 0
for (i=0; i<n; i++)
    for (j=0; j<i; j++)
        if (a[j] > a[i])
            res++;
```

$O(1)$

1, 2, 3, ... (n-1)

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

$O(n^2)$ alg $n = 50{,}000$ time = 5 sec

$n = 100{,}000$

$n = 150{,}000$

$$T(n) = c \cdot n^2$$

$$T(50000) = c \cdot (50000)^2 = 5\,\text{sec}$$

$$c = \frac{5\,\text{sec}}{50000^2}$$

$$T(100000) = \frac{5\,\text{sec}}{(50000)^2} \times (100{,}000)^2 = 5\left(\frac{100{,}000}{50{,}000}\right)^2$$

$$= 20\,\text{sec}$$

22·9/4

$$T(150{,}000) = \frac{5\,\text{sec}}{50000^2} \times (150000)^2 = 5\left(\frac{150{,}000}{50000}\right)^2 = 45\,\text{sec}$$

Types of Code Segments and their analysis

---

(1)
```
sum = 0
for (int i = 0; i < n; i++)
    sum++;
```
$O(1)$ $O(n)$

(2)
```
for (int i = 0; i < strlen(s); i++)
    printf("%c", s[i]);
```
$O(|s|)$

getting called $|s|$ # times

$O(|s|^2)$

(3)
```
int len = strlen(s);
for (int i = 0; i < len; i++)
    printf("%c", s[i]);
```
$O(n)$ $n = |s|$

$+$

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

(4)
```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        // O(1) - no Δ to i or j
    }
}
```
$O(nm)$

m times

---

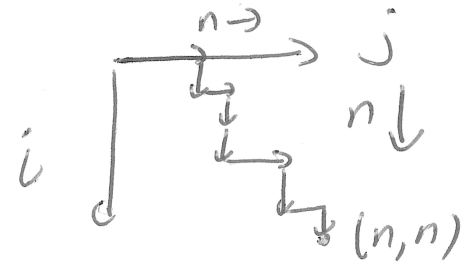
(5)
```
for (int i = 0; i < n; i++)
    for (int j = 0; j <= i; j++)
        // O(1)
```
$1 + 2 + 3 + \cdots + n$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$$

(6)

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

$O(n)$



(7) Add j = 0, then it ~~beme~~ becomes $O(n^2)$

$O(\lg n)$ (8)

```
i = 1
while (i < n)
    i = 2 * i
```

$n = 2^k$

$k = \log_2 n$

(9)

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

(10)

$n \to \frac{n}{2} \to \frac{n}{4} \dashrightarrow 1$

```
while (n > 0) {
    printf("%d", n % 2);
    n = n / 2;
}
```

$O(\lg n)$

$$\log_4 n = \frac{\log_2 n}{\log_2 4} = \underbrace{\left(\frac{1}{\log_2 4}\right)}_{\text{const}} \log_2 n$$

$O(n\sqrt{n})$ $\quad n=10^7 \quad t=7$

$$T(n) = c\,n\sqrt{n}$$

$$T(10^7) = c\,10^7 10^{3.5} = 7\text{sec}$$

$$c = \frac{7\text{sec}}{10^{10.5}}$$

$$T(4\times10^7) = \frac{7\text{sec}}{\cancel{10^{10.5}}} \cdot 4\times\cancel{10^7} \cdot \sqrt{4}\,\cancel{\sqrt{10^7}}$$

$$= 7\times4\times2 = 56\text{ sec}$$