Big O: A Review

Pat Morin
COMP2402/2002

Carleton University

Big O: Definition

$$O(g(n)) = \{f(n) : \text{ there exists positive constants } c \text{ and } n_0 \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

Big O: Definition

$$O(g(n)) = \{f(n) : \text{ there exists positive constants } c \text{ and } n_0 \text{ such that } f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

- Notice: $O(g(n))$ is a set of functions

Big O: Definition

$$O(g(n)) = \{f(n) : \begin{array}{l}\text{there exists positive constants } c \text{ and } n_0 \\ \text{such that } f(n) \le cg(n) \text{ for all } n \ge n_0\}\end{array}$$

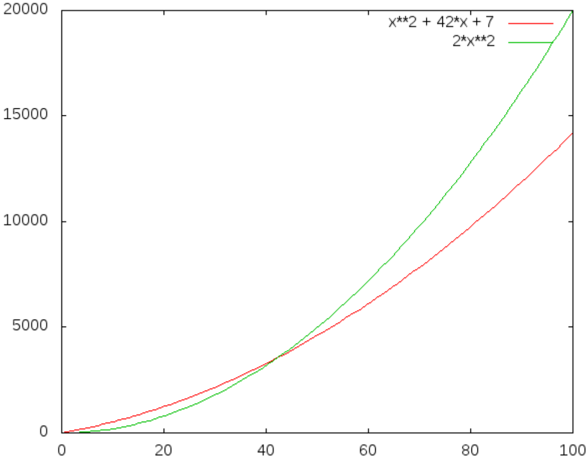
- Notice: $O(g(n))$ is a set of functions
  - When we say $f(n) = O(g(n))$ we really mean $f(n) \in O(g(n))$

Big O: Definition

$$O(g(n)) = \{f(n) : \text{ there exists positive constants } c \text{ and } n_0 \text{ such that } f(n) \le cg(n) \text{ for all } n \ge n_0\}$$

- Notice: $O(g(n))$ is a set of functions
  - When we say $f(n) = O(g(n))$ we really mean $f(n) \in O(g(n))$
  - E.g., $n^2 + 42n + 7 = O(n^2)$ means:
    - The function $f(n) = n^2 + 42n + 7$ is in the set $O(n^2)$

$n^2 + 42n + 7 = O(n^2)$



$$n^2 + 42n + 7 \leq 2n^2 \text{ for all } n \geq 50$$

Example

- Prove $n^2 + 42n + 7 = O(n^2)$

Example

- Prove $n^2 + 42n + 7 = O(n^2)$
- $$\begin{aligned} n^2 + 42n + 7 &\le n^2 + 42n^2 + 7n^2 \quad \text{for } n \ge 1 \\ &= 50n^2 \end{aligned}$$

Example

- Prove $n^2 + 42n + 7 = O(n^2)$
- $$\begin{aligned} n^2 + 42n + 7 &\leq n^2 + 42n^2 + 7n^2 \quad \text{for } n \geq 1 \\ &= 50n^2 \end{aligned}$$
- So, $n^2 + 42n + 7 \leq 50n^2$ for all $n \geq 1$

Example

- Prove $n^2 + 42n + 7 = O(n^2)$
-
$$\begin{aligned} n^2 + 42n + 7 &\le n^2 + 42n^2 + 7n^2 \quad \text{for } n \ge 1 \\ &= 50n^2 \end{aligned}$$
- So, $n^2 + 42n + 7 \le 50n^2$ for all $n \ge 1$
- $n^2 + 42n^2 + 7n^2 = O(n^2)$ [$c = 50,\ n_0 = 1$]

Example

- Prove $5n \log_2 n + 8n - 200 = O(n \log_2 n)$

Example

- Prove $5n \log_2 n + 8n - 200 = O(n \log_2 n)$

$$\begin{aligned}
5n \log_2 n + 8n - 200 &\le 5n \log_2 n + 8n \\
&\le 5n \log_2 n + 8n \log_2 n \quad \text{for } n \ge 2 \ (\log_2 n \ge 1) \\
&\le 13n \log_2 n
\end{aligned}$$

Example

- Prove $5n\log_2 n + 8n - 200 = O(n\log_2 n)$

$$
\begin{aligned}
5n\log_2 n + 8n - 200 &\le 5n\log_2 n + 8n \\
&\le 5n\log_2 n + 8n\log_2 n \qquad \text{for } n \ge 2 \ (\log_2 n \ge 1) \\
&\le 13n\log_2 n
\end{aligned}
$$

- $5n\log_2 n + 8n - 200 \le 13n\log_2 n$ for all $n \ge 2$

Example

- Prove $5n\log_2 n + 8n - 200 = O(n\log_2 n)$

$$\begin{aligned}5n\log_2 n + 8n - 200 &\le 5n\log_2 n + 8n \\ &\le 5n\log_2 n + 8n\log_2 n \qquad \text{for } n \ge 2\ (\log_2 n \ge 1) \\ &\le 13n\log_2 n\end{aligned}$$

- $5n\log_2 n + 8n - 200 \le 13n\log_2 n$ for all $n \ge 2$
- $5n\log_2 n + 8n - 200 = O(n\log_2 n)$ $[\ c = 13,\ n_0 = 2\ ]$

Some common relations

- $O(n^{c_1}) \subset O(n^{c_2})$ for any $c_1 < c_2$
- For any constants $a, b, c > 0$,

$$O(a) \subset O(\log n) \subset O(n^b) \subset O(c^n)$$

Some common relations

- $O(n^{c_1}) \subset O(n^{c_2})$ for any $c_1 < c_2$
- For any constants $a, b, c > 0$,

$$O(a) \subset O(\log n) \subset O(n^b) \subset O(c^n)$$

- These make things faster

$$2\log_2 n + 2 = O(\log n)$$

$$n + 2 = O(n)$$

$$2n + 15n^{1/2} = O(n)$$

Some common relations

- $O(n^{c_1}) \subset O(n^{c_2})$ for any $c_1 < c_2$
- For any constants $a, b, c > 0$,

$$O(a) \subset O(\log n) \subset O(n^b) \subset O(c^n)$$

- These make things faster

$$2\log_2 n + 2 = O(\log n)$$

$$n + 2 = O(n)$$

$$2n + 15n^{1/2} = O(n)$$

- We can multiply these to learn about other functions,

$$O(an) = O(n) \subset O(n \log n) \subset O(n^{1+b}) \subset O(nc^n)$$

Some common relations

- $O(n^{c_1}) \subset O(n^{c_2})$ for any $c_1 < c_2$
- For any constants $a, b, c > 0$,

$$O(a) \subset O(\log n) \subset O(n^b) \subset O(c^n)$$

- These make things faster

$$2\log_2 n + 2 = O(\log n)$$

$$n + 2 = O(n)$$

$$2n + 15n^{1/2} = O(n)$$

- We can multiply these to learn about other functions,

$$O(an) = O(n) \subset O(n \log n) \subset O(n^{1+b}) \subset O(nc^n)$$

- Examples: $O(n^{1.5}) \subseteq O(n^{1.5} \log n)$

An indulgence

- In this course, we have seen expressions like $O(n - i)$
  - Two argument function $g(n, i) = n - i$
  - *For the purposes of this course*, we will take $O(g(n, i))$ to be

$$O(g(n,i)) = \{f(n,i) : \begin{array}{l}\text{there exists positive constants } c \text{ and } n_0 \\ \text{such that } f(n,i) \leq cg(n,i) \text{ for all } n \geq n_0 \\ \text{and all valid arguments } i\}\end{array}$$

- For example (Lists) valid values of i are $\{0, \ldots, n-1\}$ or (sometimes) $\{0, \ldots, n\}$

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is
  - 1 assignment (`int i = 0`)

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is
  - 1 assignment (`int i = 0`)
  - n+1 comparisons (`i < n`)

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is
  - 1 assignment (`int i = 0`)
  - n+1 comparisons (`i < n`)
  - n increments (`i++`)

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is
  - 1 assignment (`int i = 0`)
  - n+1 comparisons (`i < n`)
  - n increments (`i++`)
  - n array offset calculations (`a[i]`)

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is
  - 1 assignment (`int i = 0`)
  - n+1 comparisons (`i < n`)
  - n increments (`i++`)
  - n array offset calculations (`a[i]`)
  - n indirect assignments (`a[i] = i`)

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is
  - 1 assignment (`int i = 0`)
  - n+1 comparisons (`i < n`)
  - n increments (`i++`)
  - n array offset calculations (`a[i]`)
  - n indirect assignments (`a[i] = i`)
  - $= a + b(n+1) + cn + dn + en$, where a, b, c, d, and e are constants that depend on the machine running the code

Why Use big-O Notation?

- Consider the following (simple) code:

```
for (int i = 0; i < n; i++) {
  a[i] = i;
}
```

- The running time is
  - 1 assignment (`int i = 0`)
  - n+1 comparisons (`i < n`)
  - n increments (`i++`)
  - n array offset calculations (`a[i]`)
  - n indirect assignments (`a[i] = i`)
  - $= a + b(n+1) + cn + dn + en$, where a, b, c, d, and e are constants that depend on the machine running the code
- Easier just to say $O(n)$ (constant-time) operations