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

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

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

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

- 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 \( O(n^2) \)
\[ n^2 + 42n + 7 = O(n^2) \]

For all \( n \geq 50 \)

\[ n^2 + 42n + 7 \leq 2n^2 \]
Example

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

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

\[
n^2 + 42n + 7 \leq n^2 + 42n^2 + 7n^2 \quad \text{for } n \geq 1
\]
\[
= 50n^2
\]
Example

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

  $n^2 + 42n + 7 \leq n^2 + 42n^2 + 7n^2 \quad \text{for } n \geq 1$

  $= 50n^2$

- So, $n^2 + 42n + 7 \leq 50n^2$ for all $n \geq 1$
Example

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

- $n^2 + 42n + 7 \leq n^2 + 42n^2 + 7n^2$ for $n \geq 1$
  
  \[= 50n^2\]

- So, $n^2 + 42n + 7 \leq 50n^2$ for all $n \geq 1$

- $n^2 + 42n^2 + 7n^2 = O(n^2)$ [ $c = 50$, $n_0 = 1$ ]
Example

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

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

\[
5n \log_2 n + 8n - 200 \leq 5n \log_2 n + 8n
\]
\[
\leq 5n \log_2 n + 8n \log_2 n \quad \text{for } n \geq 2 \left(\log_2 n \geq 1\right)
\]
\[
\leq 13n \log_2 n
\]
Example

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

$$5n \log_2 n + 8n - 200 \leq 5n \log_2 n + 8n$$
$$\leq 5n \log_2 n + 8n \log_2 n \quad \text{for } n \geq 2 \quad (\log_2 n \geq 1)$$
$$\leq 13n \log_2 n$$

- $5n \log_2 n + 8n - 200 \leq 13n \log_2 n$ for all $n \geq 2$
Example

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

$$5n \log_2 n + 8n - 200 \leq 5n \log_2 n + 8n$$

$$\leq 5n \log_2 n + 8n \log_2 n \quad \text{for } n \geq 2 \ (\log_2 n \geq 1)$$

$$\leq 13n \log_2 n$$

- $5n \log_2 n + 8n - 200 \leq 13n \log_2 n$ for all $n \geq 2$

- $5n \log_2 n + 8n - 200 = O(n \log_2 n) \ [c = 13, \ n_0 = 2]$
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: $O(n^{1.5}) \subset O(n^{1.5} \log n)$
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

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

- 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:

```java
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.
Why Use big-O Notation?

Consider the following (simple) code:

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

The running time is

\[ \text{running time} = a + b(n + 1) + cn + dn + en, \]

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.
Why Use big-O Notation?

Consider the following (simple) code:

```c
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:

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

The running time is

- 1 assignment (int i = 0)
- n+1 comparisons (i < n)
Consider the following (simple) code:

```java
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++)
Consider the following (simple) code:

```java
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:

```java
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:

```java
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:

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

The running time is

- 1 assignment (\texttt{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