Big O Notation

Robin Visser

IOI Training Camp
University of Cape Town

3 December 2016

Overview

Background

- Often times, it is an easy exercise to construct an algorithm which will solve the task at hand.

Background

- Often times, it is an easy exercise to construct an algorithm which will solve the task at hand.
- What is more challenging, is getting an algorithm which runs in the allocated *time* and *memory* constraints.

Background

- Often times, it is an easy exercise to construct an algorithm which will solve the task at hand.
- What is more challenging, is getting an algorithm which runs in the allocated *time* and *memory* constraints.
- Big O notation allows us to efficiently classify algorithms based on their time/memory performance for large inputs.

Definition

- Consider an algorithm which depends on some parameter n (for example: sorting an array of size n)
- Let $T(n)$ be the number of steps that the algorithm takes.

Definition

- Consider an algorithm which depends on some parameter n (for example: sorting an array of size n)
- Let $T(n)$ be the number of steps that the algorithm takes.

Definition

We say $T(n)$ is $O(g(n))$ if and only if there exists constants C and n_0 such that, for all $n \geq n_0$, we have:

$$T(n) \leq Cg(n)$$

Definition

- Consider an algorithm which depends on some parameter n (for example: sorting an array of size n)
- Let $T(n)$ be the number of steps that the algorithm takes.

Definition

We say $T(n)$ is $O(g(n))$ if and only if there exists constants C and n_0 such that, for all $n \geq n_0$, we have:

$$T(n) \leq Cg(n)$$

Alternate Definition

We say $T(n)$ is $O(g(n))$ if and only if:

$$\lim_{n \to \infty} \frac{T(n)}{g(n)} < \infty$$

Examples

Let $T(n) = n^2 + 42n + 7$. Can we say $T(n)$ is $O(n)$? What about $O(n^2)$ or $O(n^3)$?

Examples

Let $T(n) = n^2 + 42n + 7$. Can we say $T(n)$ is $O(n)$? What about $O(n^2)$ or $O(n^3)$?

$$\lim_{n\to\infty} \frac{T(n)}{n} = \lim_{n\to\infty} \left(n + 42 + \frac{7}{n}\right) = \infty$$

Therefore, $T(n) \neq O(n)$

Examples

Let $T(n) = n^2 + 42n + 7$. Can we say $T(n)$ is $O(n)$? What about $O(n^2)$ or $O(n^3)$?

$$\lim_{n\to\infty} \frac{T(n)}{n} = \lim_{n\to\infty} \left(n + 42 + \frac{7}{n}\right) = \infty$$

Therefore, $T(n) \neq O(n)$

$$\lim_{n\to\infty} \frac{T(n)}{n^2} = \lim_{n\to\infty} \left(1 + \frac{42}{n} + \frac{7}{n^2}\right) = 1 < \infty$$

Therefore, $T(n) = O(n^2)$

Examples

Let $T(n) = n^2 + 42n + 7$. Can we say $T(n)$ is $O(n)$? What about $O(n^2)$ or $O(n^3)$?

$$\lim_{n\to\infty} \frac{T(n)}{n} = \lim_{n\to\infty} \left(n + 42 + \frac{7}{n}\right) = \infty$$

Therefore, $T(n) \neq O(n)$

$$\lim_{n\to\infty} \frac{T(n)}{n^2} = \lim_{n\to\infty} \left(1 + \frac{42}{n} + \frac{7}{n^2}\right) = 1 < \infty$$

Therefore, $T(n) = O(n^2)$

$$\lim_{n\to\infty} \frac{T(n)}{n^3} = \lim_{n\to\infty} \left(\frac{1}{n} + \frac{42}{n^2} + \frac{7}{n^3}\right) = 0 < \infty$$

Therefore, we can also say, $T(n) = O(n^3)$

Examples

More examples:

- $3n^3 + 7n^2 + n + 36 = O(n^3)$
  (for polynomials, we only care about the leading term)

Examples

More examples:

- $3n^3 + 7n^2 + n + 36 = O(n^3)$
  (for polynomials, we only care about the leading term)
- $5n^2 \log_2 n + 4n \log_2 n + 6n = O(n^2 \log_2 n)$
  (we can ignore constants, and only take the highest-order term)

Examples

More examples:

- $3n^3 + 7n^2 + n + 36 = O(n^3)$
  (for polynomials, we only care about the leading term)
- $5n^2 \log_2 n + 4n \log_2 n + 6n = O(n^2 \log_2 n)$
  (we can ignore constants, and only take the highest-order term)
- $13 \log_2 n = O(\log_2 n) = O(\log_7 n) = O(\log n)$
  (the base for logarithms does not matter)

Examples

More examples:

- $3n^3 + 7n^2 + n + 36 = O(n^3)$
  (for polynomials, we only care about the leading term)
- $5n^2 \log_2 n + 4n \log_2 n + 6n = O(n^2 \log_2 n)$
  (we can ignore constants, and only take the highest-order term)
- $13 \log_2 n = O(\log_2 n) = O(\log_7 n) = O(\log n)$
  (the base for logarithms does not matter)
- $9 \log n + 5(\log n)^4 + 18\sqrt{n} \log n + 2n^2 = O(n^2)$

Formalities

- Given some function $g(n)$, we note $O(g(n))$ as a *set* of functions. Hence we should actually say: $n^2 + 42n + 7 \in O(n^2)$, although most just abuse notation and use the equality symbol.

Formalities

- Given some function $g(n)$, we note $O(g(n))$ as a *set* of functions. Hence we should actually say: $n^2 + 42n + 7 \in O(n^2)$, although most just abuse notation and use the equality symbol.
- Note that $3n^3 = O(n^3)$. We can technically also say $3n^3 = O(n^5)$ or $3n^3 = O(2^n)$ or $3n^3 = O(n!)$, although we usually only care about the tightest possible bound.

Examples

Calculate the time complexity of the following:

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

Examples

Calculate the time complexity of the following:

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

To calculate the total number of steps $T(n)$, we note:

Examples

Calculate the time complexity of the following:

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

To calculate the total number of steps $T(n)$, we note:

- 1 assignment (`int i = 0`)

Examples

Calculate the time complexity of the following:

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

To calculate the total number of steps $T(n)$, we note:

- 1 assignment (`int i = 0`)
- $n + 1$ comparisons (`i < n`)

Examples

Calculate the time complexity of the following:

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

To calculate the total number of steps $T(n)$, we note:

- 1 assignment (`int i = 0`)
- $n + 1$ comparisons (`i < n`)
- n increments (`i++`)

Examples

Calculate the time complexity of the following:

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

To calculate the total number of steps $T(n)$, we note:

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

Examples

Calculate the time complexity of the following:

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

To calculate the total number of steps $T(n)$, we note:

- 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`)

Examples

Calculate the time complexity of the following:

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

To calculate the total number of steps $T(n)$, we note:

- 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`)

In total: $T(n) = a + b(n+1) + cn + dn + en$, where the constants a, b, c, d, e depend on the compiler/machine running the code. It is much easier to simply say the code runs in $O(n)$ time.

Examples

What is the time complexity of the following:

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

Examples

What is the time complexity of the following:

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

Answer: $O(n^2)$
(also technically $O(n^3), O(n^4), O(5^n) \dots$)

Examples

What is the time complexity of the following:

```
int lo = 0;
int hi = n;
while (hi - lo > 1) {
    mid = (lo+hi)/2;
    if (a[mid] < x) {
        lo = mid;
    } else {
        hi = mid;
    }
}
```

Examples

What is the time complexity of the following:

```
int lo = 0;
int hi = n;
while (hi - lo > 1) {
    mid = (lo+hi)/2;
    if (a[mid] < x) {
        lo = mid;
    } else {
        hi = mid;
    }
}
```

Answer: $O(\log n)$
(also technically $O(n^2 \log n), O(n^3 \cdot 2^n), O(n^{n^n}) \dots$)

Examples

What is the time complexity of the following:

```
for (int i = 0; i < 10000; i++) {
    for (int j = 0; j < i; j++) {
        for (int k = 0; k < j; k++) {
            d[i][j] = dist[i][k] + dist[k][j];
        }
    }
}
```

Examples

What is the time complexity of the following:

```
for (int i = 0; i < 10000; i++) {
    for (int j = 0; j < i; j++) {
        for (int k = 0; k < j; k++) {
            d[i][j] = dist[i][k] + dist[k][j];
        }
    }
}
```

Answer: $O(1)$
(there isn't a dependency on n or any other parameter)

Examples

What is the time complexity of the following:

```
void  specialsum(int left, int right) {
    if (left == right) {return;}
    for (int i = left; i <= right; i++) {
        total += vec[i];
    }
    int mid = (left+right)/2;
    specialsum(left, mid);
    specialsum(mid, right);
}
total = 0;
specialsum(0, n);
```

Examples

What is the time complexity of the following:

```
void  specialsum(int left, int right) {
    if (left == right) {return;}
    for (int i = left; i <= right; i++) {
        total += vec[i];
    }
    int mid = (left+right)/2;
    specialsum(left, mid);
    specialsum(mid, right);
}
total = 0;
specialsum(0, n);
```

Answer: $O(n \log n)$

Summary

When calculating how long an algorithm will take, a good rule of thumb is to assume between 10^7 and 10^8 operations per second. This allows us to estimate what the time Big O time needs to be given some constraint:

Constraint	**Time Complexity**
$N \leq 10\,000\,000$	$O(n)$
$N \leq 500\,000$	$O(n \log n)$
$N \leq 5\,000$	$O(n^2)$
$N \leq 200$	$O(n^3)$
$N \leq 20$	$O(2^n)$
$N \leq 10$	$O(n!)$

Summary

Notation	Name	Example
$O(1)$	constant	Calculating $(-1)^n$
$O(\log n)$	logarithmic	Binary Search
$O(n)$	linear	Searching for item in array
$O(n \log n)$	linearithmic	Heapsort, merge sort
$O(n^2)$	quadratic	Selection sort, insertion sort
$O(n^3)$	cubic	Naive multiplication of 2 $n \times n$ matrices
$O(c^n)$	exponential	Solving travelling salesmen using DP
$O(n!)$	factorial	Solving travelling salesmen by brute force

Limitations

- The Big O time complexity of an algorithm only considers large input.

Limitations

- The Big O time complexity of an algorithm only considers large input.
- For smaller input, constants do matter.

Limitations

- The Big O time complexity of an algorithm only considers large input.
- For smaller input, constants do matter.
- Two algorithms could have the same Big-O time complexity but one might be practically faster than the other.

Limitations

- The Big O time complexity of an algorithm only considers large input.
- For smaller input, constants do matter.
- Two algorithms could have the same Big-O time complexity but one might be practically faster than the other.
- Big O only gives the worst case bound. For competition purposes, one can often assume test data which will exhibit worst case behaviour, however for most practical test cases, the time taken may be far below the worst case bound.