Algorithm Analysis

Class 5

Big-Theta

Prove that $T(n) = n^2 + 5n - 3 \in \Theta(n^2)$ using the definitions, and then illustrate graphically

Big-Theta

Prove that $T(n) = n^2 + 5n - 3 \in \Theta(n^2)$ using the definitions, and then illustrate graphically

Big-Theta is a two-part if and only if, so to prove it requires two subproofs:

1. $T(n) \in O(n^2)$
2. $T(n) \in \Omega(n^2)$

Subproof 1: Big-O

Show that $T(n) \in O(n^2)$ by finding c_1 and n_0 such that $T(n) \leq c_1 n^2$ when $n \geq n_0$

$$n^2 + 5n - 3 \leq c_1 n^2$$

or equivalently

$$c_1 \geq 1 + \frac{5}{n} - \frac{3}{n^2}$$

what is the maximum value of $1 + \frac{5}{n} - \frac{3}{n^2}$?
c_1 must be greater than or equal to that value

Big-O

$$\frac{d}{dn}\left(1+\frac{5}{n}-\frac{3}{n^2}\right)=\frac{6-5n}{n^3}$$

this has real zero at $n=\frac{6}{5}$, 2nd derivative shows it's a maximum

the value of $1+\frac{5}{n}-\frac{3}{n^2}$ at $n=\frac{6}{5}$ is $\frac{37}{12}$

thus we choose $c_1=4$ and $n_0=2$

and we can write

$$n^2+5n-3\leq 4n^2 \text{ when } n\geq 2$$

and big-Oh is proved by the definition.

Big-O

$$\frac{d}{dn}\left(1 + \frac{5}{n} - \frac{3}{n^2}\right) = \frac{6 - 5n}{n^3}$$

this has real zero at $n = \frac{6}{5}$, 2nd derivative shows it's a maximum

the value of $1 + \frac{5}{n} - \frac{3}{n^2}$ at $n = \frac{6}{5}$ is $\frac{37}{12}$

thus we choose $\boxed{c_1 = 4}$ and $\boxed{n_0 = 2}$

and we can write

$$n^2 + 5n - 3 \leq 4n^2 \text{ when } n \geq 2$$

and big-Oh is proved by the definition.

Subproof 2: Big-Omega

Show that $T(n) \in \Omega(n^2)$ by finding c_2 and n_0 s.t. $T(n) \geq c_2 n^2$ when $n \geq n_0$

$$n^2 + 5n - 3 \geq c_2 n^2$$

or equivalently

$$c_2 \leq 1 + \frac{5}{n} - \frac{3}{n^2}$$

what is the minimum value of $1 + \frac{5}{n} - \frac{3}{n^2}$?
c_2 must be equal to or smaller than that value

Big-Omega

$$\lim_{n\to\infty}\left(1+\frac{5}{n}-\frac{3}{n^2}\right)=1$$

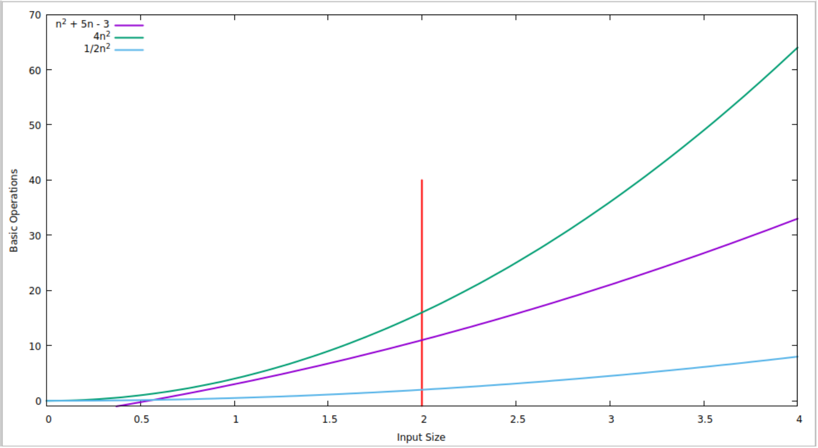
thus we choose $c_2 = \frac{1}{2}$ and, $n_0 = 2$ still works,

and so we can write

$$n^2 + 5n - 3 \geq \frac{1}{2}n^2 \text{ when } n \geq 2$$

and big-Omega is proved by the definition.

Graphic Illustration



Rules of Thumb

- lower-order terms are irrelevant — we can ignore them
- the highest-order term corresponds to the basic operations that occur most frequently — these are what we count, e.g.,

$$f(n) \leq n^3 - \frac{27}{3}n^2 + 16n - 4 \in O(n^3)$$

Rules of Thumb

- lower-order terms are irrelevant — we can ignore them
- the highest-order term corresponds to the basic operations that occur most frequently — these are what we count, e.g.,

$$f(n) \leq n^3 - \frac{27}{3}n^2 + 16n - 4 \in O(n^3)$$

- leading coefficients indicate how many most-frequently occurring basic operations are performed
- since we know we can ignore leading coefficients, the exact count of basic operations does not affect the final analysis, e.g.,

$$f(n) \geq 3n^2 \in \Omega(n^2)$$

Analyze an Algorithm

- in the previous set of slides we analyzed the continuous function $n^2 + 5n - 3$
- in CS, we do not wish to analyze continuous functions
- we wish to analyze discrete algorithms
- analyze the algorithm `find_max` (only the function) (this means find a big-Theta set or a pair of big-Oh and big-Omega sets for the algorithm)

we must determine and state several things:

Analyze an Algorithm

- in the previous set of slides we analyzed the continuous function $n^2 + 5n - 3$
- in CS, we do not wish to analyze continuous functions
- we wish to analyze discrete algorithms
- analyze the algorithm `find_max` (only the function) (this means find a big-Theta set or a pair of big-Oh and big-Omega sets for the algorithm)

we must determine and state several things:

1. what is the input and how do we measure its size?

2. does the input arrangement matter?

Analyze an Algorithm

- in the previous set of slides we analyzed the continuous function $n^2 + 5n - 3$
- in CS, we do not wish to analyze continuous functions
- we wish to analyze discrete algorithms
- analyze the algorithm `find_max` (only the function) (this means find a big-Theta set or a pair of big-Oh and big-Omega sets for the algorithm)

we must determine and state several things:

1. what is the input and how do we measure its size?
   the input is the array; the size is the number of elements
2. does the input arrangement matter?

Analyze an Algorithm

- in the previous set of slides we analyzed the continuous function $n^2 + 5n - 3$
- in CS, we do not wish to analyze continuous functions
- we wish to analyze discrete algorithms
- analyze the algorithm `find_max` (only the function) (this means find a big-Theta set or a pair of big-Oh and big-Omega sets for the algorithm)

we must determine and state several things:

1. what is the input and how do we measure its size?
   the input is the array; the size is the number of elements
2. does the input arrangement matter?
   no; the algorithm cannot end early even if the max element is the very first element

Counting Basic Operations

3. what basic operations are we counting? when find_max runs, what statements are executed, and how many times?

let n be the size of the array (line 45)

Counting Basic Operations

3. what basic operations are we counting? when find_max runs, what statements are executed, and how many times?

let n be the size of the array (line 45)

- line 44: one assignment (regardless of n)
- line 45: one assignment (regardless of n)
- line 47: one assignment or increment, and one comparison $\times n = 2n$ operations
- line 49: one array access, one comparison, and one assignment $\times n - 1 = 3(n - 1)$ operations
- line 51: we do not count return statements

if n is 3, exactly how many operations are performed?

Counting Basic Operations

3. what basic operations are we counting? when find_max runs, what statements are executed, and how many times?

let n be the size of the array (line 45)

- line 44: one assignment (regardless of n)
- line 45: one assignment (regardless of n)
- line 47: one assignment or increment, and one comparison $\times n = 2n$ operations
- line 49: one array access, one comparison, and one assignment $\times n - 1 = 3(n-1)$ operations
- line 51: we do not count return statements

if n is 3, exactly how many operations are performed? 14

Counting Basic Operations

if n is 4, exactly how many operations?

Counting Basic Operations

if n is 4, exactly how many operations? 19

Counting Basic Operations

if n is 4, exactly how many operations? 19

how many for $n = 5$?

Counting Basic Operations

if n is 4, exactly how many operations? 19

how many for $n = 5$?

n	ops
3	14
4	19
5	24

what is a formula for $T(n)$, and what is its analysis?

Counting Basic Operations

if n is 4, exactly how many operations? 19

how many for $n = 5$?

n	ops
3	14
4	19
5	24

what is a formula for $T(n)$, and what is its analysis?

$$T(n) = 5n - 1$$
$$T(n) \in \Theta(n)$$

Number of Operations

- some algorithms execute an exact number of operations as a function of input size: $T(n) = f(n)$

Number of Operations

- some algorithms execute an exact number of operations as a function of input size: $T(n) = f(n)$

- some algorithms execute at most a number of operations based on input size: $T(n) \leq f(n)$ (worst case)
- and execute at least different number of operations: $T(n) \geq g(n)$ (best case)
- in other words: $g(n) \leq T(n) \leq f(n)$

Efficiency Classes

big-Theta

- if we can determine an algorithm's exact number of operations, we can find c_1, c_2, and n_0 for some standard function $f(n)$ and thus we have $T(n) \in \Theta(f(n))$

Efficiency Classes

big-Theta

- if we can determine an algorithm's exact number of operations, we can find c_1, c_2, and n_0 for some standard function $f(n)$ and thus we have $T(n) \in \Theta(f(n))$

big-O

- if we can determine that an algorithm executes at most a certain number of operations, we can find a c and n_0 for some standard function $f(n)$ and thus we have $T(n) \in O(f(n))$

Efficiency Classes

big-Theta

- if we can determine an algorithm's exact number of operations, we can find c_1, c_2, and n_0 for some standard function $f(n)$ and thus we have $T(n) \in \Theta(f(n))$

big-O

- if we can determine that an algorithm executes at most a certain number of operations, we can find a c and n_0 for some standard function $f(n)$ and thus we have $T(n) \in O(f(n))$

big-Omega

- if we can determine that an algorithm executes at least a certain number of operations, we can find a c and n_0 for some standard function $f(n)$ and thus we have $T(n) \in \Omega(f(n))$

Big-Theta

to analyze an algorithm, we must find an upper and a lower bound for its behavior

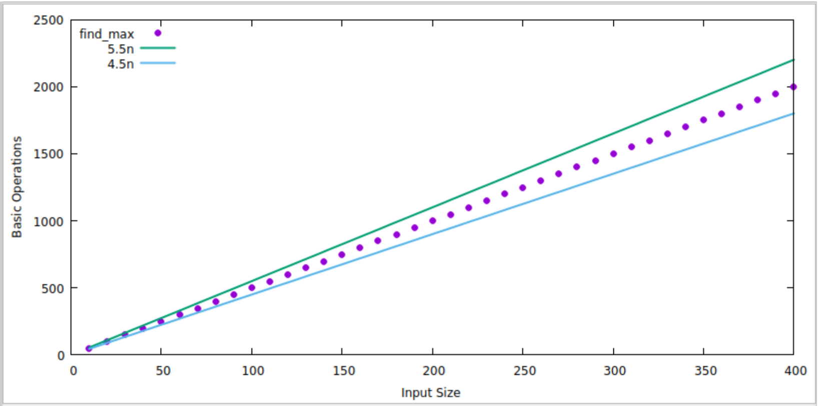
- if we find a big-Theta set for an algorithm
- that function is both an upper and a lower bound for the algorithm
- in this case we know exactly how the algorithm scales

Big-Oh and Big-Omega

- if we cannot find a single function and thus a big-Theta set
- we must find a big-Oh set for its upper bound
- and a big-Omega set for its lower bound

Big-Theta

- we saw an illustration of big-Theta earlier with the function we analyzed
- find_max has a similar picture



Big-Oh and Big-Omega

if there is not a big-Theta set, we have one big-Oh set and a different big-Omega set