Asymptotic Analysis

Primality Testing

Primality Testing

Definition

Primality Testing

Definition: Let $num \geq 2$ be an integer.

Primality Testing

<u>Definition</u>: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime

12 is not prime

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime

12 is not prime

Definition:

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime   12 is not prime

Definition: Let $num \geq 2$ be an integer

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime

12 is not prime

Definition: Let $num \geq 2$ be an integer, and let d and k be two divisors of num.

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime

12 is not prime

Definition: Let $num \geq 2$ be an integer, and let d and k be two divisors of num. We say that *k and d are complementary divisors of num*

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime

12 is not prime

Definition: Let $num \geq 2$ be an integer, and let d and k be two divisors of num. We say that *k and d are complementary divisors of num*, if $d \cdot k = num$

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime   12 is not prime

Definition: Let $num \geq 2$ be an integer, and let d and k be two divisors of num. We say that *k and d are complementary divisors of num*, if $d \cdot k = num$

Examples:

4 and 25 are complementary divisors of 100

Primality Testing

Definition: Let $num \geq 2$ be an integer. We say that *num is prime*, if its only divisors are 1 and num

Examples:

13 is prime　　　　12 is not prime

Definition: Let $num \geq 2$ be an integer, and let d and k be two divisors of num. We say that *k and d are complementary divisors of num*, if $d \cdot k = num$

Examples:

4 and 25 are complementary divisors of 100

5 and 20 are complementary divisors of 100

Primality Testing

Version I: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , num}$

Primality Testing

Version I: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , num}$

Version II:

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

Primality Testing

Version I: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , num}$

Version II: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}}, \quad . \quad . \quad . \quad , num$

num = 100:

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

num = 100: 1 2 4 5 10 20 25 50 100

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

num = 100: 1 2 4 5 10 20 25 50 100

$\frac{100}{2}$

Primality Testing

$$1,2,3, \quad . \quad . \quad . \quad ,\frac{num}{2}, \quad . \quad . \quad . \quad ,num$$

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range.

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad \underbrace{. \quad . \quad . }_{k} \quad , num$$

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor

Primality Testing

$$1,2,3,\quad .\quad .\quad .\quad ,\frac{num}{2},\quad .\quad .\quad .\quad ,num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have:

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $\boxed{d = \frac{num}{k}}$

We have: $d = \dfrac{num}{k}$

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $\boxed{k > \frac{num}{2}}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} \; ? \; \dfrac{num}{\left(\frac{num}{2}\right)}$

Primality Testing

$$1, 2, 3, \;.\;\;.\;\;.\;, \frac{num}{2}, \;.\;\;.\;\;.\;, num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)}$

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num}$

Primality Testing

$$1,2,3,\quad .\quad .\quad .\quad ,\frac{num}{2},\quad .\quad .\quad .\quad ,num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Primality Testing

$$1,2,3,\ \ .\ \ .\ \ .\ \ ,\frac{num}{2},\ \ .\ \ .\ \ .\ \ ,num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Primality Testing

$$1,2,3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$.

Primality Testing

$$1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$. Since d is a divisor

Primality Testing

$$1,2,3,\quad .\quad .\quad .\quad ,\frac{num}{2},\quad .\quad .\quad .\quad ,num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$. Since d is a divisor

Primality Testing

$$1,2,3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$. Since d is a divisor, we get that $d=1$.

Primality Testing

$$1,2,3,\ .\ .\ .\ ,\frac{num}{2},\ .\ .\ .\ ,num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $\boxed{d = \frac{num}{k}}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$. Since d is a divisor, we get that $\boxed{d=1}$.

Primality Testing

$$1,2,3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad \underset{k}{.} \quad , num$$

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $\boxed{d = \frac{num}{k}}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$. Since d is a divisor, we get that $\boxed{d=1}$.

So: $\frac{num}{k} = 1$

Primality Testing

$$1,2,3,\quad .\quad .\quad .\quad ,\frac{num}{2},\quad .\quad .\quad .\quad ,num$$

k

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$. Since d is a divisor, we get that $d=1$.

So: $\frac{num}{k} = 1$, therefore $k = num$.

Primality Testing

$$1,2,3,\;.\;\;.\;\;.\;,\frac{num}{2},\;\underbrace{.\;\;.\;\;.\;,num}_{k}$$

Let k be a divisor of num in the second half of the range. That is, $k > \frac{num}{2}$

Let d be k's complementary divisor, therefore $d = \frac{num}{k}$

We have: $d = \dfrac{num}{k} < \dfrac{num}{\left(\frac{num}{2}\right)} = \frac{num}{1} \cdot \frac{2}{num} = 2$

Therefore $d<2$. Since d is a divisor, we get that $d=1$.

So: $\frac{num}{k}=1$, therefore $k = num$.

This shows that the only divisor in the second half of the range is num itself.

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

Version III:

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

Version III: $1, 2, 3, \ldots, \sqrt{num}, \quad . \quad . \quad . \quad , num$

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

Version III: $1, 2, 3, \ldots, \sqrt{num}, \quad . \quad . \quad . \quad , num$

num = 100: 1 2 4 5 10 20 25 50 100

Primality Testing

Version I: $1, 2, 3, \quad . \quad . \quad . \quad , num$

Version II: $1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}, \quad . \quad . \quad . \quad , num$

Version III: $1, 2, 3, \ldots, \sqrt{num}, \quad . \quad . \quad . \quad , num$

num = 100: 1 2 4 5 10 20 25 50 100

$\frac{100}{2}$

Primality Testing

Version I: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , num}$

Version II: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}}, \quad . \quad . \quad . \quad , num$

Version III: $\underbrace{1, 2, 3, \ldots, \sqrt{num}}, \quad . \quad . \quad . \quad , num$

$num = 100$: 1 2 4 5 10 20 25 50 100

$\sqrt{100}$ $\frac{100}{2}$

Primality Testing

1 2 4 5 10 20 25 50 100

Primality Testing

1 2 4 5 10 20 25 50 100

Primality Testing

1 2 4 5 10 20 25 50 100

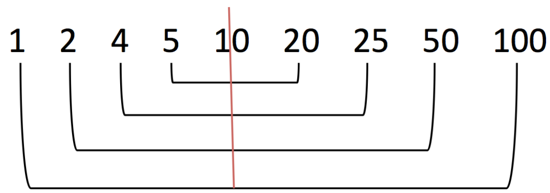
Primality Testing

1 2 4 5 10 20 25 50 100

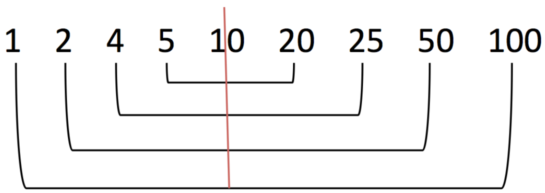
Primality Testing

1 2 4 5 10 20 25 50 100

Primality Testing

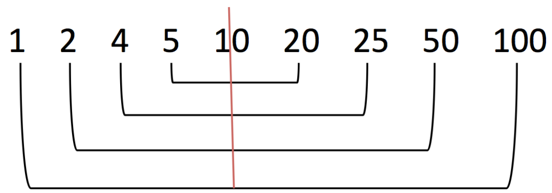


Primality Testing



$$1, 2, 3, \ldots, \sqrt{num}, \quad . \quad . \quad . \quad , num$$

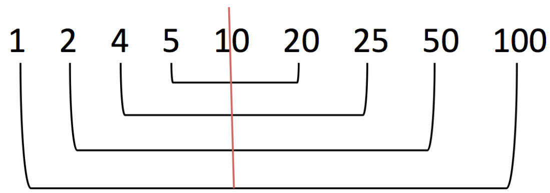
Primality Testing



$$1, 2, 3, \ldots, \sqrt{num}, \quad . \quad . \quad . \quad , num$$

k, d

Primality Testing



$$1, 2, 3, \ldots, \sqrt{num}, \quad . \quad . \quad . \quad , num$$

k, d

Let k and d be complementary divisors of num

Primality Testing

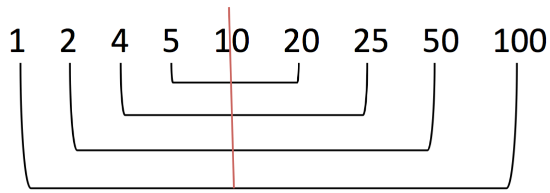
1 2 4 5 10 20 25 50 100

$$1, 2, 3, \ldots, \sqrt{num}, \quad . \quad . \quad . \quad , num$$

k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

Primality Testing



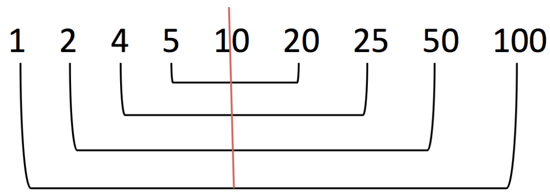
$$1,2,3,\ldots,\sqrt{num}, \quad . \quad . \quad . \quad ,num$$

k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have:

Primality Testing



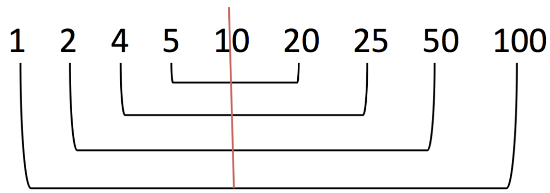
$$1,2,3,\ldots,\sqrt{num},\quad . \quad . \quad . \quad ,num$$

k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have: $num = k \cdot d$

Primality Testing



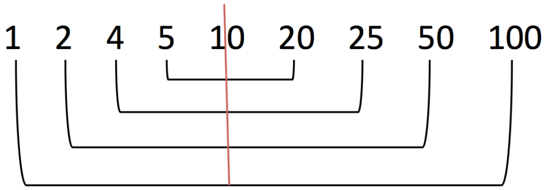
$$1,2,3,\ldots,\sqrt{num},\quad . \quad . \quad . \quad ,num$$

k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have: $num = k \cdot d > \sqrt{num} \cdot \sqrt{num}$

Primality Testing



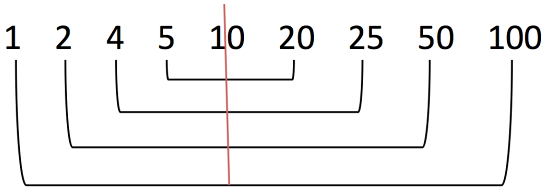
$$1,2,3,\ldots,\sqrt{num},\quad . \quad . \quad . \quad ,num$$

$k,\ d$

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have: $num = k \cdot d > \sqrt{num} \cdot \sqrt{num} = num$

Primality Testing



$$1,2,3,\ldots,\sqrt{num}, \quad . \quad . \quad . \quad ,num$$

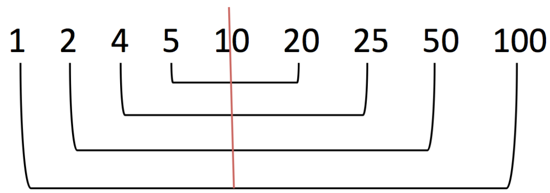
k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have: $num = k \cdot d > \sqrt{num} \cdot \sqrt{num} = num$

This implies that $num > num$

Primality Testing



$$1,2,3,\ldots,\sqrt{num},\quad .\quad .\quad .\quad ,num$$

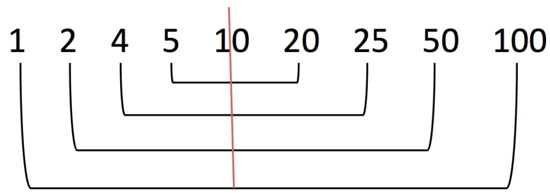
k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have: $num = k \cdot d > \sqrt{num} \cdot \sqrt{num} = num$

This implies that $num > num$, which is a contradiction.

Primality Testing



$$1,2,3,\ldots,\sqrt{num}, \quad . \quad . \quad . \quad ,num$$

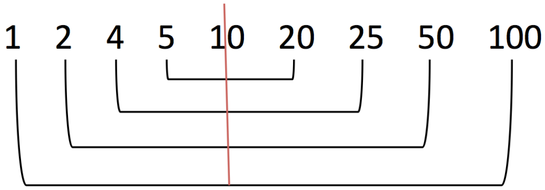
k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have: $num = k \cdot d > \sqrt{num} \cdot \sqrt{num} = num$

This implies that $num > num$, which is a contradiction.

Primality Testing



$$1,2,3,\ldots,\sqrt{num}, \quad . \quad . \quad . \quad ,num$$

k, d

Let k and d be complementary divisors of num, and assume that they are both greater than $\sqrt{num}$

We therefore have: $num = k \cdot d > \sqrt{num} \cdot \sqrt{num} = num$

This implies that $num > num$, which is a contradiction.

This shows that at least one in each pair of complementary divisors is less than or equal to $\sqrt{num}$

Primality Testing

Version I: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , num}$

Version II: $\underbrace{1, 2, 3, \quad . \quad . \quad . \quad , \frac{num}{2}}, \quad . \quad . \quad . \quad , num$

Version III: $\underbrace{1, 2, 3, \ldots, \sqrt{num}}, \quad . \quad . \quad . \quad , num$

Runtime Analysis

Runtime Analysis

- The running time depends on the size of the input

Runtime Analysis

- The running time depends on the size of the input
- The running time depends on the operators we use, and on the types of the data they are applied on

Runtime Analysis

- The running time depends on the size of the input

- The running time depends on the operators we use, and on the types of the data they are applied on

- The running time depends on the machine’s hardware technology

Runtime Analysis

- The running time depends on the size of the input
  - ✓ We parameterize running time by the size of the input
- The running time depends on the operators we use, and on the types of the data they are applied on
- The running time depends on the machine's hardware technology

Runtime Analysis

- The running time depends on the size of the input
  - ✓ We parameterize running time by the size of the input
- The running time depends on the operators we use, and on the types of the data they are applied on
  - ✓ The abstract model we use, ignores machine-dependent constants.
    We count each primitive operation as 1
- The running time depends on the machine’s hardware technology

Runtime Analysis

- The running time depends on the size of the input
  - ✓ We parameterize running time by the size of the input
- The running time depends on the operators we use, and on the types of the data they are applied on
  - ✓ The abstract model we use, ignores machine-dependent constants.
    We count each primitive operation as 1
- The running time depends on the machine's hardware technology
  - ✓ The abstract model we use, divides the algorithms to classes based on their ”quality”.
    We make asymptotic analysis: look at the order of growth of T(n)

Runtime Analysis

Informal Criteria

Runtime Analysis

Informal Criteria

We compare the asymptotic order of the number of primitive operations executed by a process, as a function of its input size

Runtime Analysis

Informal Criteria

We compare the asymptotic order of the number of primitive operations executed by a process, as a function of its input size

$$T(n) = 3n^2 + 6n - 15$$

Runtime Analysis
Informal Criteria

We compare the asymptotic order of the number of primitive operations executed by a process, as a function of its input size

$$T(n) = 3n^2 + 6n - 15 = \theta(n^2)$$

Runtime Analysis

Informal Criteria

We compare the asymptotic order of the number of primitive operations executed by a process, as a function of its input size

$$T(n) = 3n^2 + 6n - 15 = \theta(n^2)$$

Rule of thumb to get the order of growth:

Runtime Analysis

Informal Criteria

We compare the asymptotic order of the number of primitive operations executed by a process, as a function of its input size

$$T(n) = 3n^2 + \cancel{6n - 15} = \theta(n^2)$$

Rule of thumb to get the order of growth:

- Drop low-order terms

Runtime Analysis
Informal Criteria

We compare the asymptotic order of the number of primitive operations executed by a process, as a function of its input size

$$T(n) = \cancel{3}n^2 + \cancel{6n - 15} = \theta(n^2)$$

Rule of thumb to get the order of growth:
- Drop low-order terms
- Ignore leading constants

Runtime Analysis

Informal Criteria

We compare the asymptotic order of the number of primitive operations executed by a process, as a function of its input size

$$T(n) = \cancel{3}n^2 + \cancel{6n - 15} = \theta(n^2)$$

Rule of thumb to get the order of growth:

- Drop low-order terms
- Ignore leading constants

More Formally . . .

Asymptotic Analysis

O definition

Asymptotic Analysis

O definition

<u>Definition</u>

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

Asymptotic Analysis
O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $\boldsymbol{f(n)=O(g(n))}$

Asymptotic Analysis
O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c

Asymptotic Analysis

O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c and a positive integer constant n_0

Asymptotic Analysis

O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

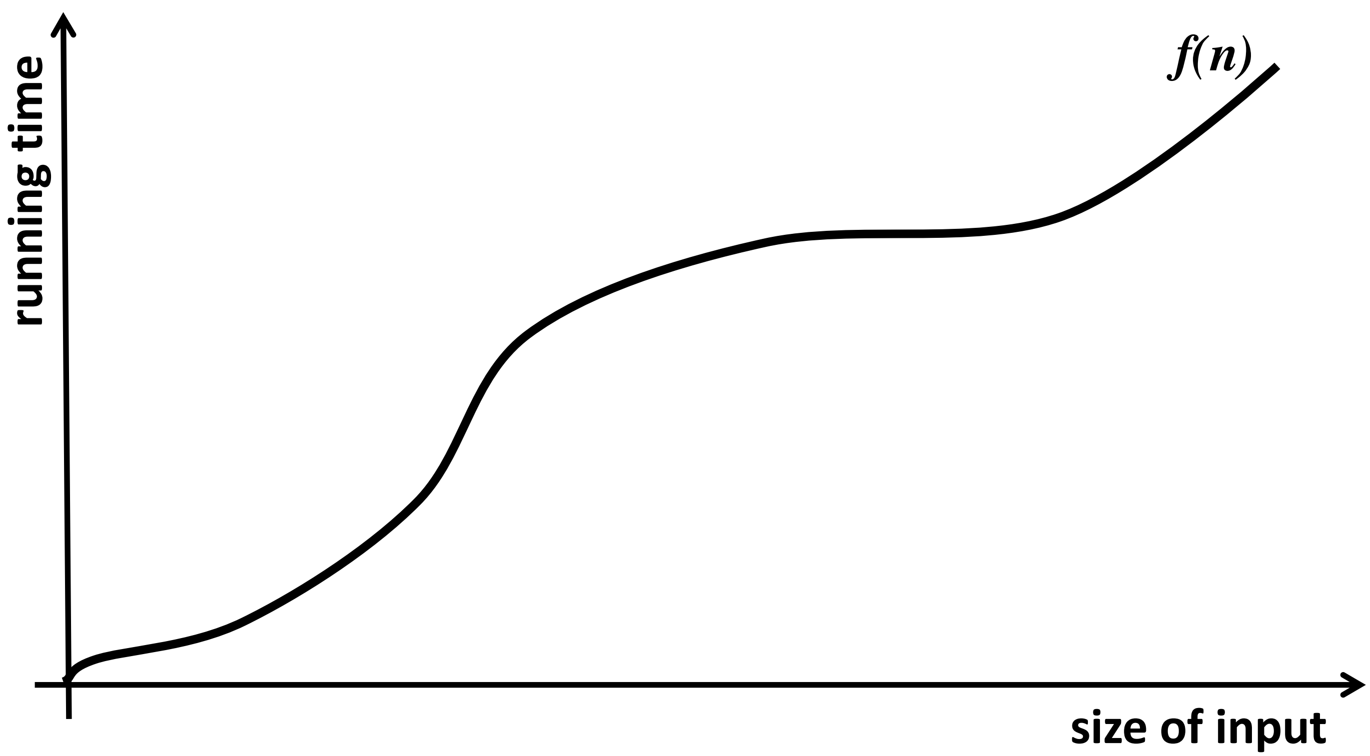
Asymptotic Analysis

O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

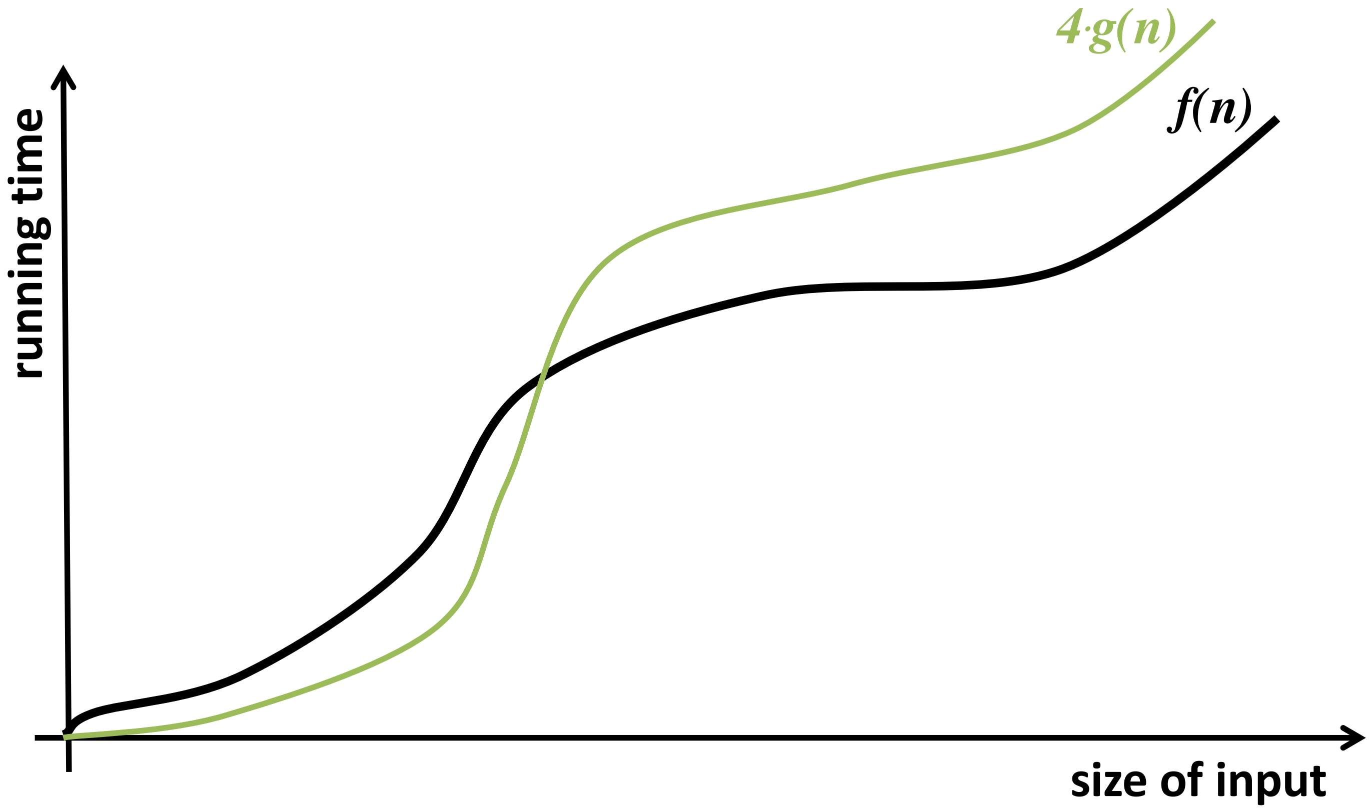


Asymptotic Analysis
O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \le c \cdot g(n)$ for all $n \geq n_0$



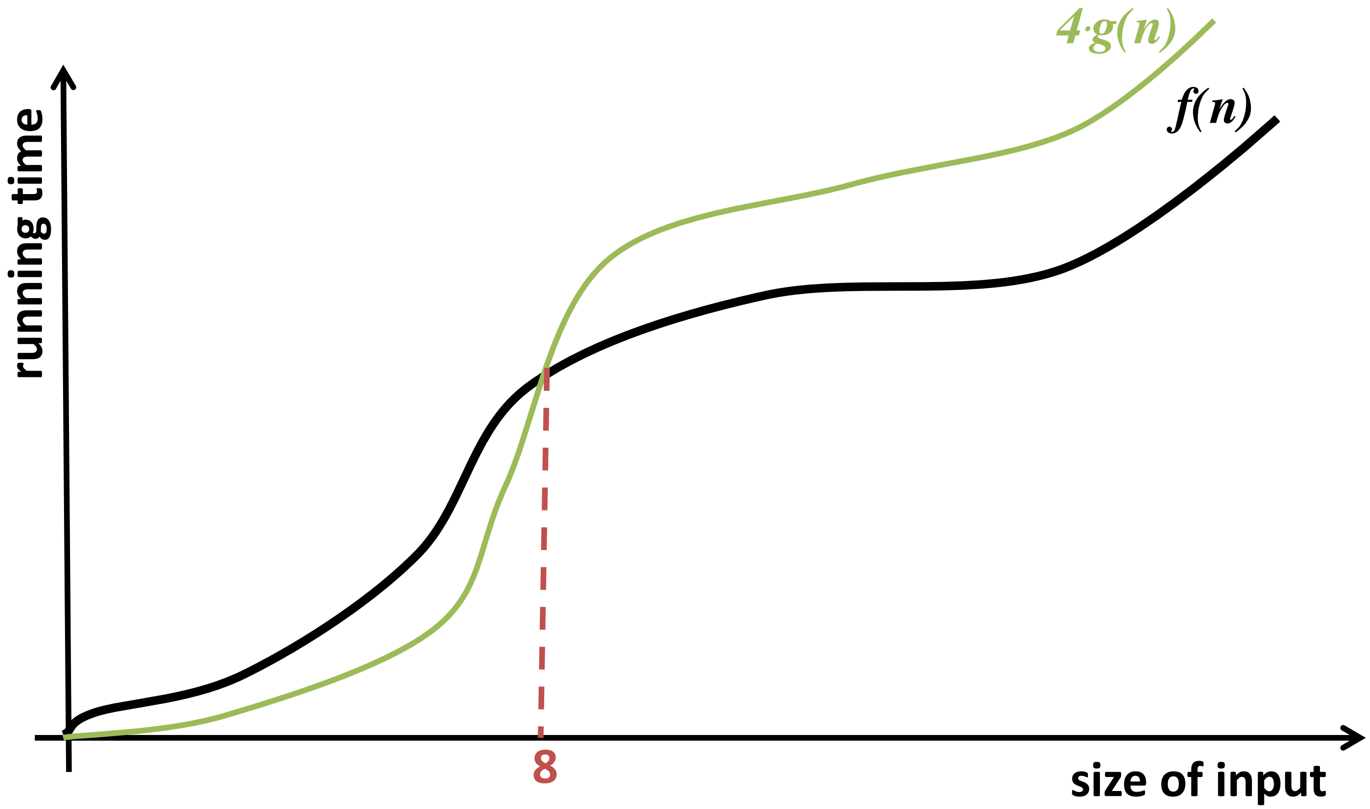
Asymptotic Analysis

O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $\mathbf{f(n) \leq c \cdot g(n)}$ for all $n \geq n_0$



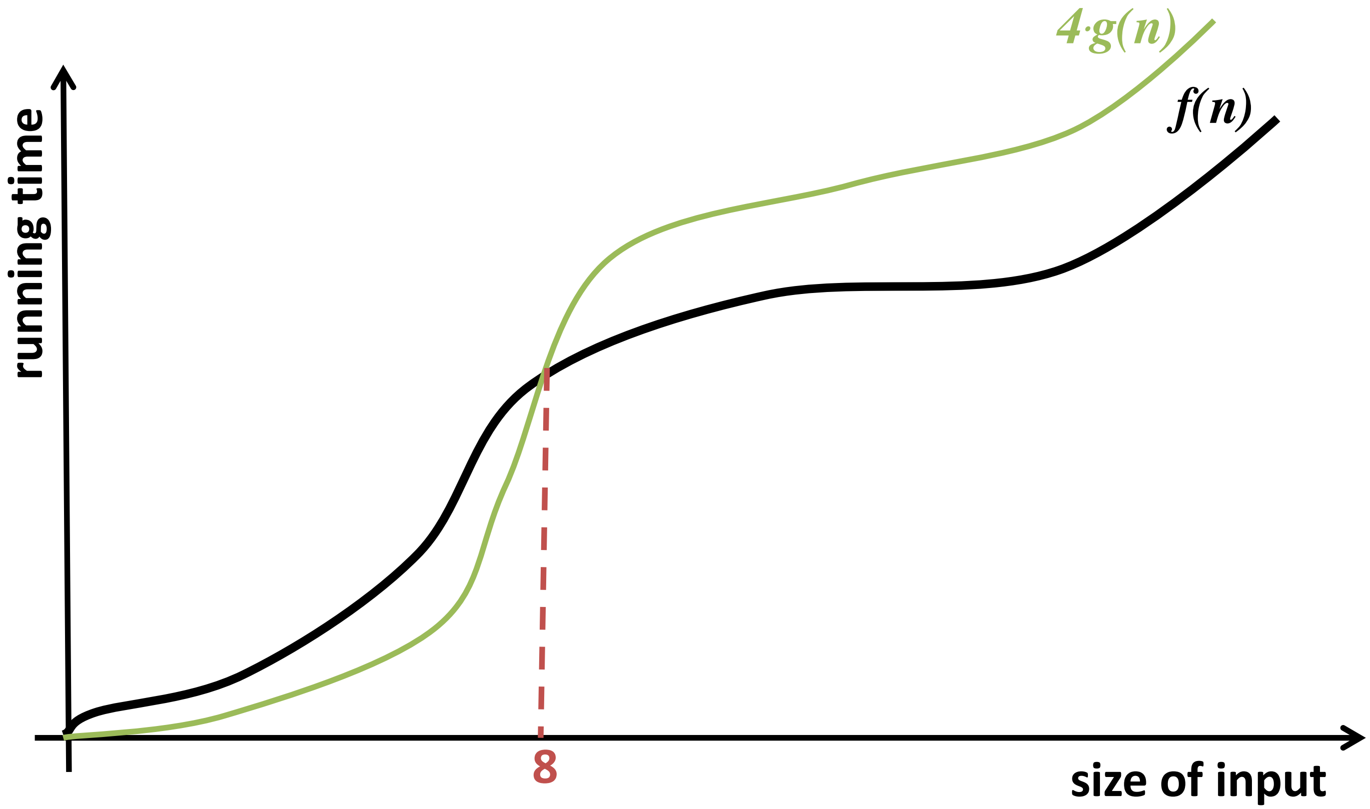
Asymptotic Analysis
O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

$n_0 = 8$
$c = 4$



Asymptotic Analysis

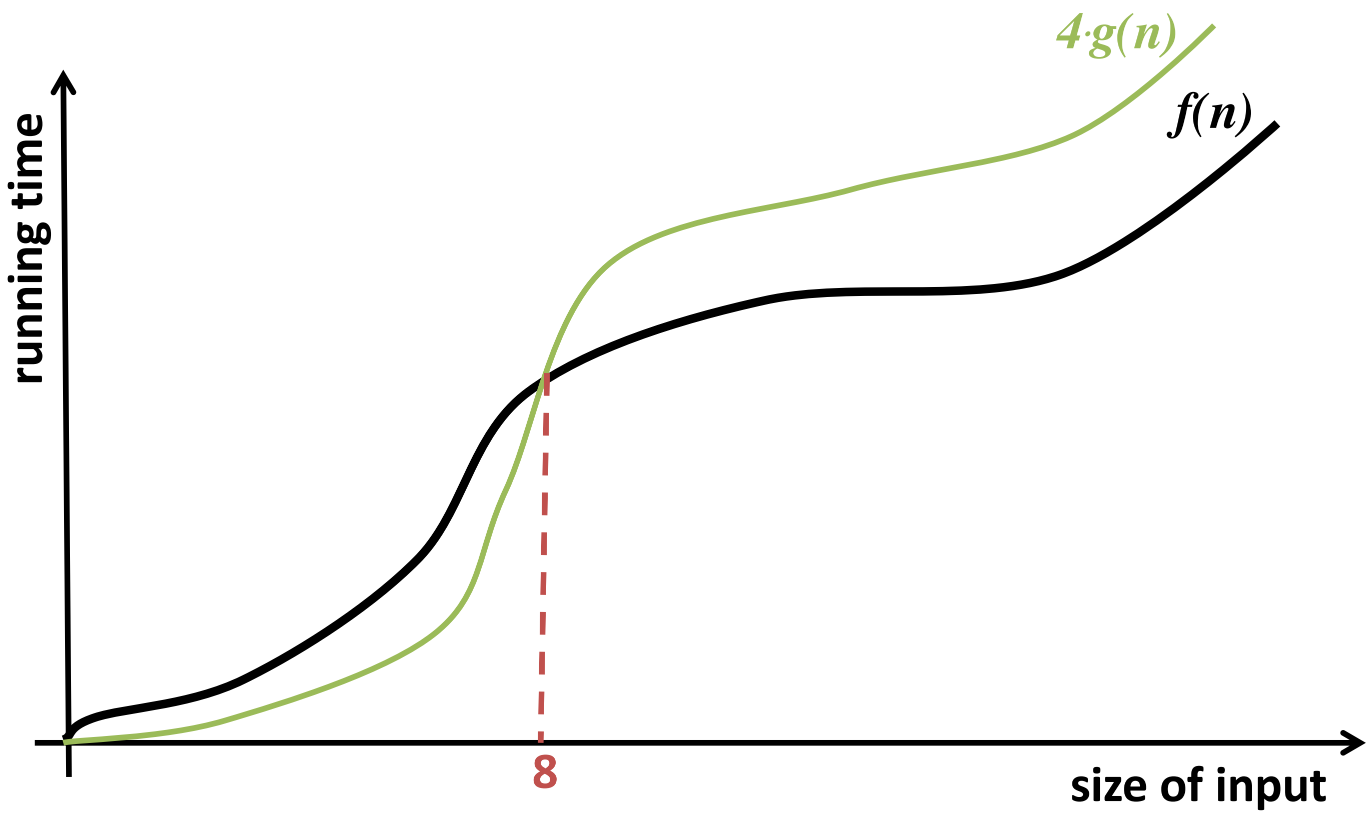
O definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=O(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

$n_0 = 8$
$c = 4$
$\Downarrow$
$f(n) = O(g(n))$



Asymptotic Analysis

Ω definition

Asymptotic Analysis

Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

Asymptotic Analysis
Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\Omega(g(n))$

Asymptotic Analysis

Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\Omega(g(n))$ if there exist positive real constant c

Asymptotic Analysis

Ω definition

<u>Definition</u>

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $\boldsymbol{f(n)=\Omega(g(n))}$ if there exist positive real constant c and a positive integer constant n_0

Asymptotic Analysis

Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

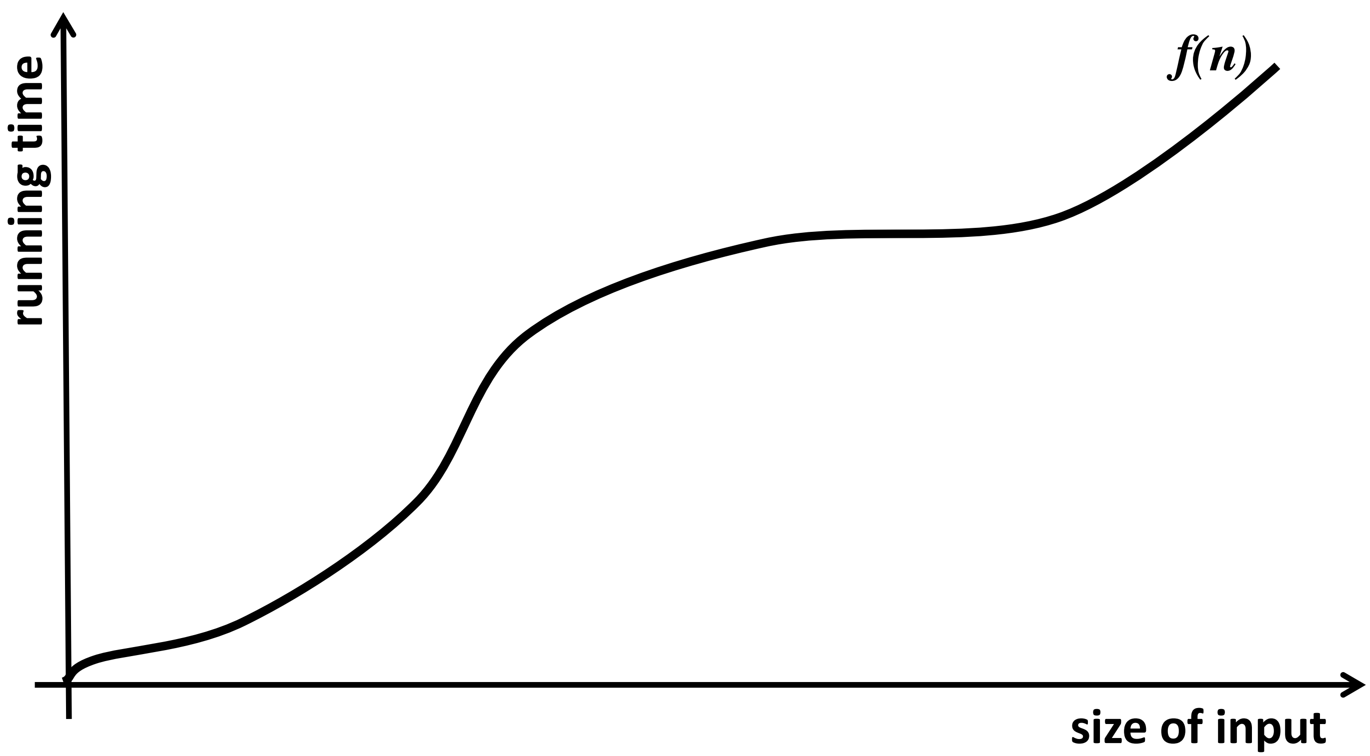
We say that $f(n)=\Omega(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$

Asymptotic Analysis
Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\Omega(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$

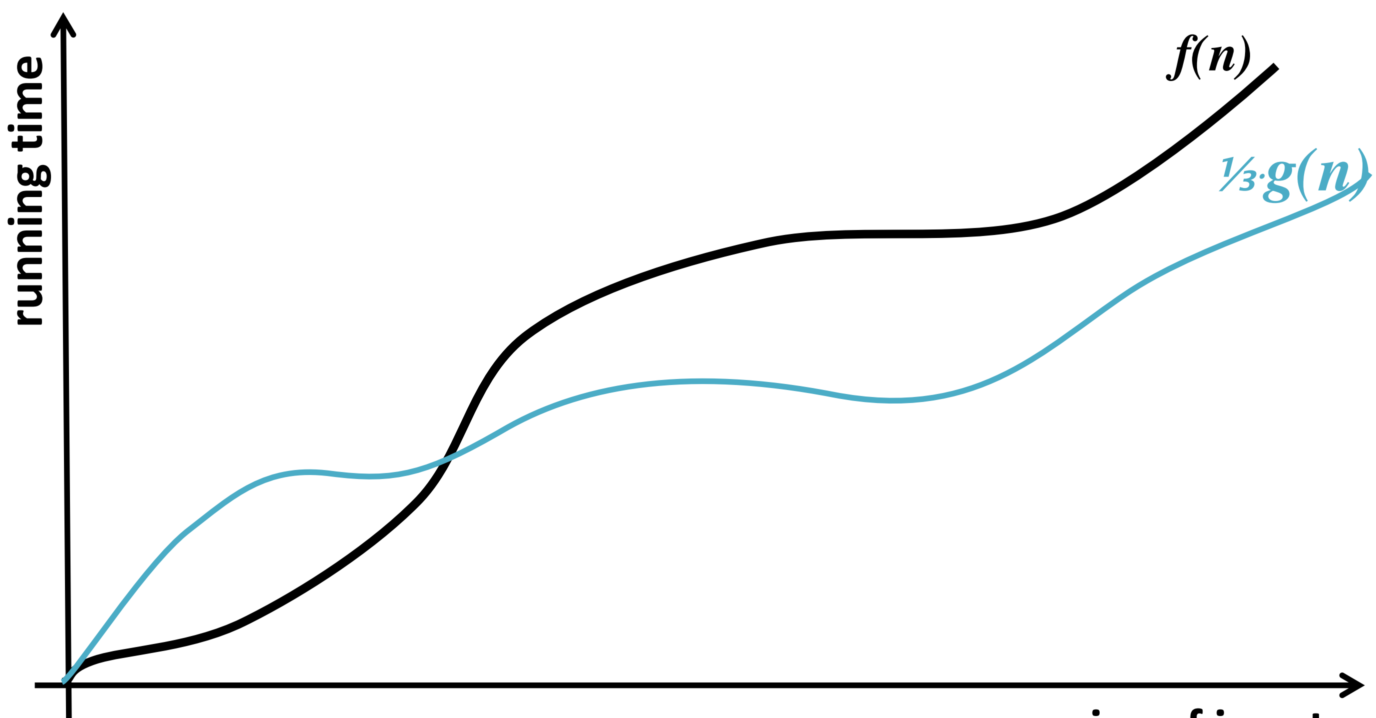


Asymptotic Analysis
Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\Omega(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$



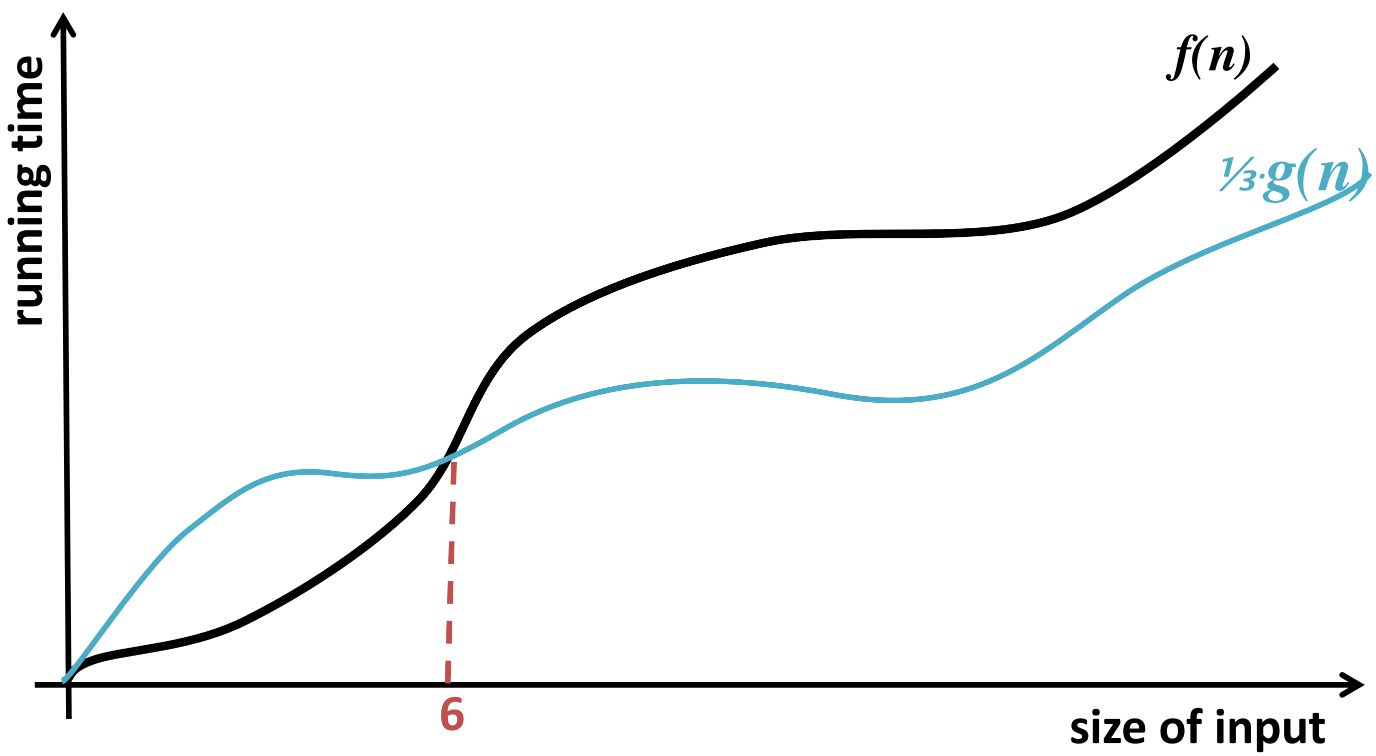
Asymptotic Analysis

Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\Omega(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$



Asymptotic Analysis

Ω definition

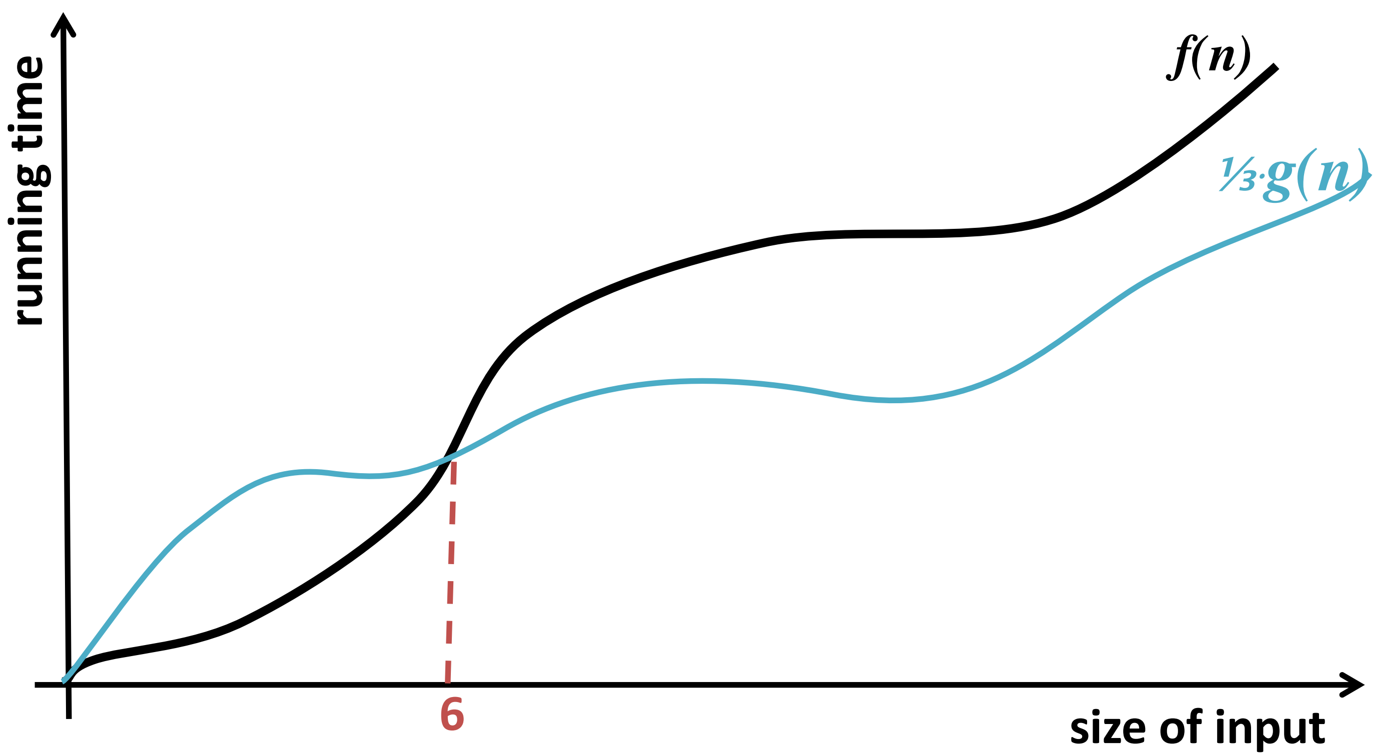
Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\Omega(g(n))$ if there exist positive real constant c and a positive integer constant n_0 such that $f(n) \geq c \cdot g(n)$ for all $n \geq n_0$

$n_0 = 6$

$c_2 = ⅓$



Asymptotic Analysis

Ω definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

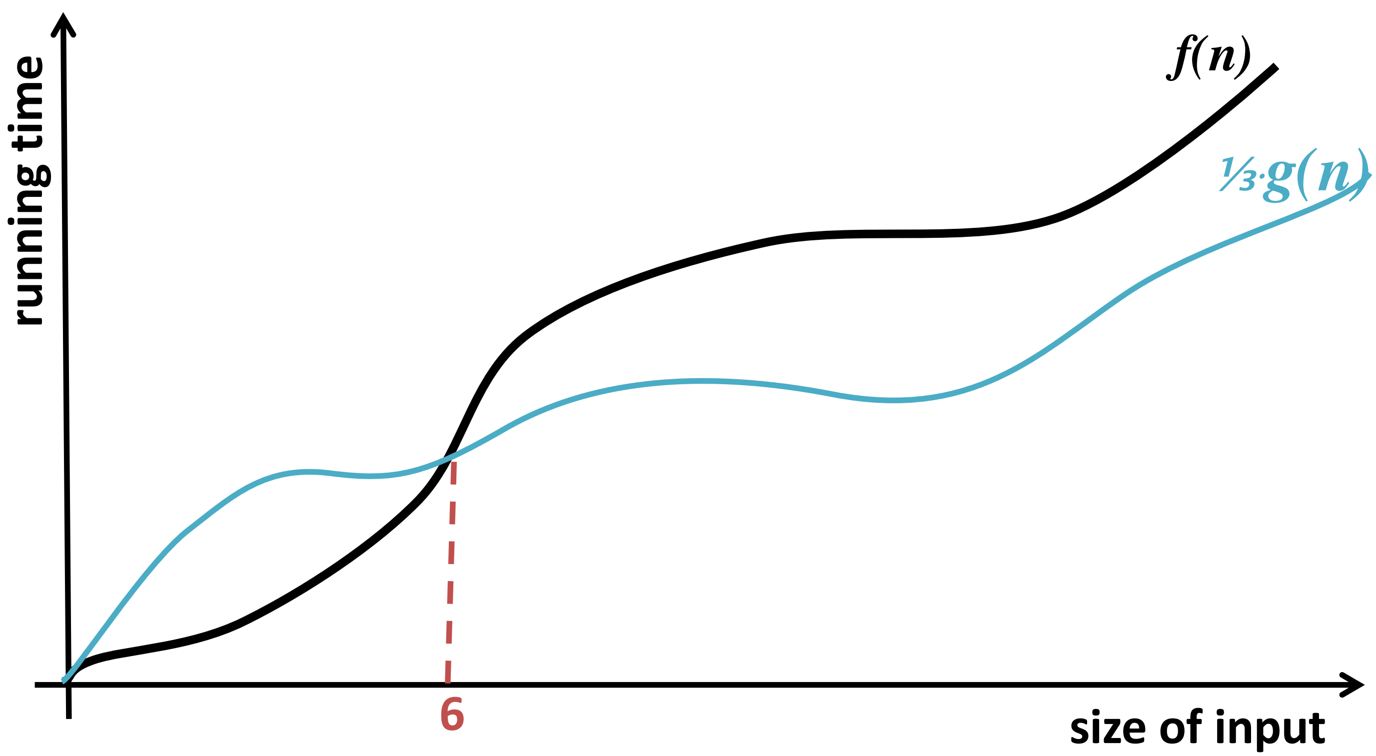
We say that $\boldsymbol{f(n)=\Omega(g(n))}$ if there exist positive real constant c and a positive integer constant n_0 such that $\boldsymbol{f(n) \geq c \cdot g(n)}$ for all $n \geq n_0$

$n_0 = 6$

$c_2 = ⅓$

$\Downarrow$

$f(n) = \Omega(g(n))$



Asymptotic Analysis

Θ definition

Asymptotic Analysis
Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

Asymptotic Analysis

Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $\boldsymbol{f(n)=\theta(g(n))}$

Asymptotic Analysis
Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $\boldsymbol{f(n)=\theta(g(n))}$ if there exist positive real constants c_1, c_2

Asymptotic Analysis

Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $\boldsymbol{f(n)=\theta(g(n))}$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0

Asymptotic Analysis

Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

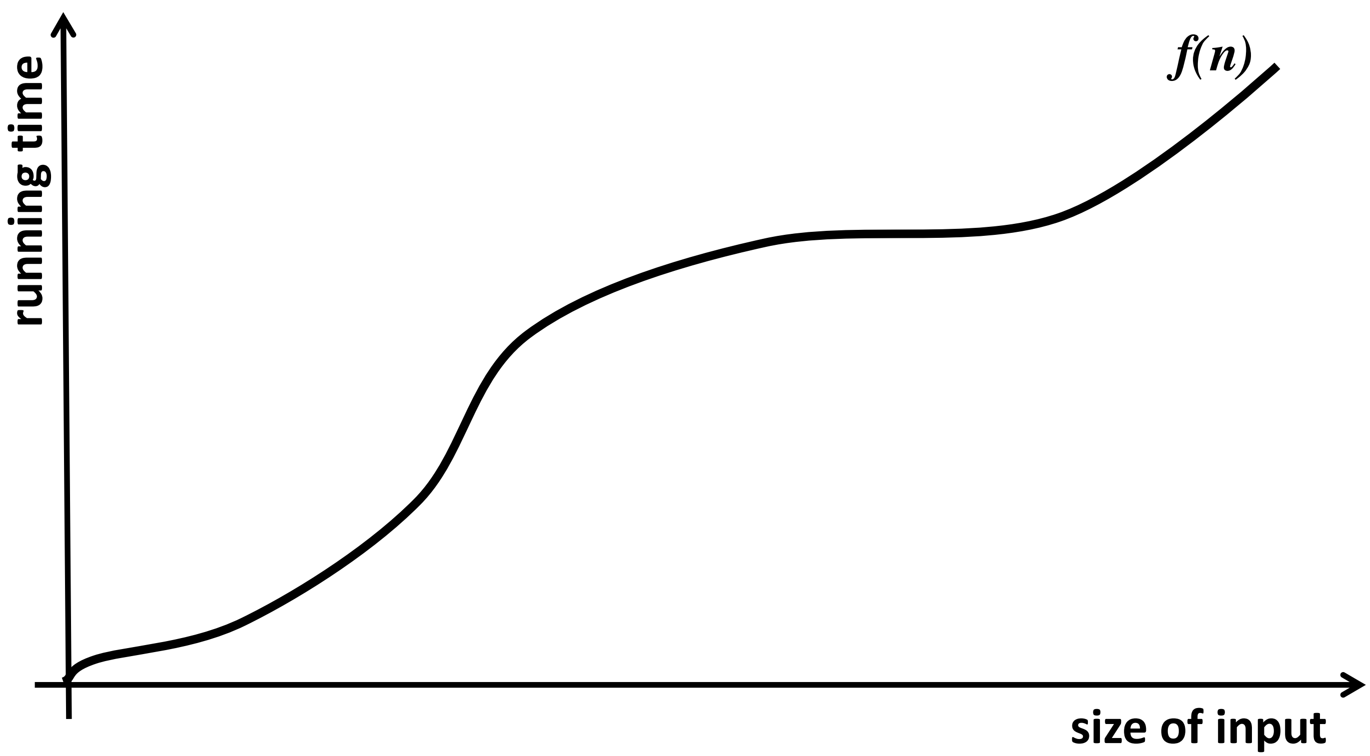
We say that $\boldsymbol{f(n)=\theta(g(n))}$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0 such that $\boldsymbol{c_2 g(n) \leq f(n) \leq c_1 g(n)}$ for all $n \geq n_0$

Asymptotic Analysis
Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $\boldsymbol{f(n)=\theta(g(n))}$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0 such that $\boldsymbol{c_2 g(n) \leq f(n) \leq c_1 g(n)}$ for all $n \geq n_0$

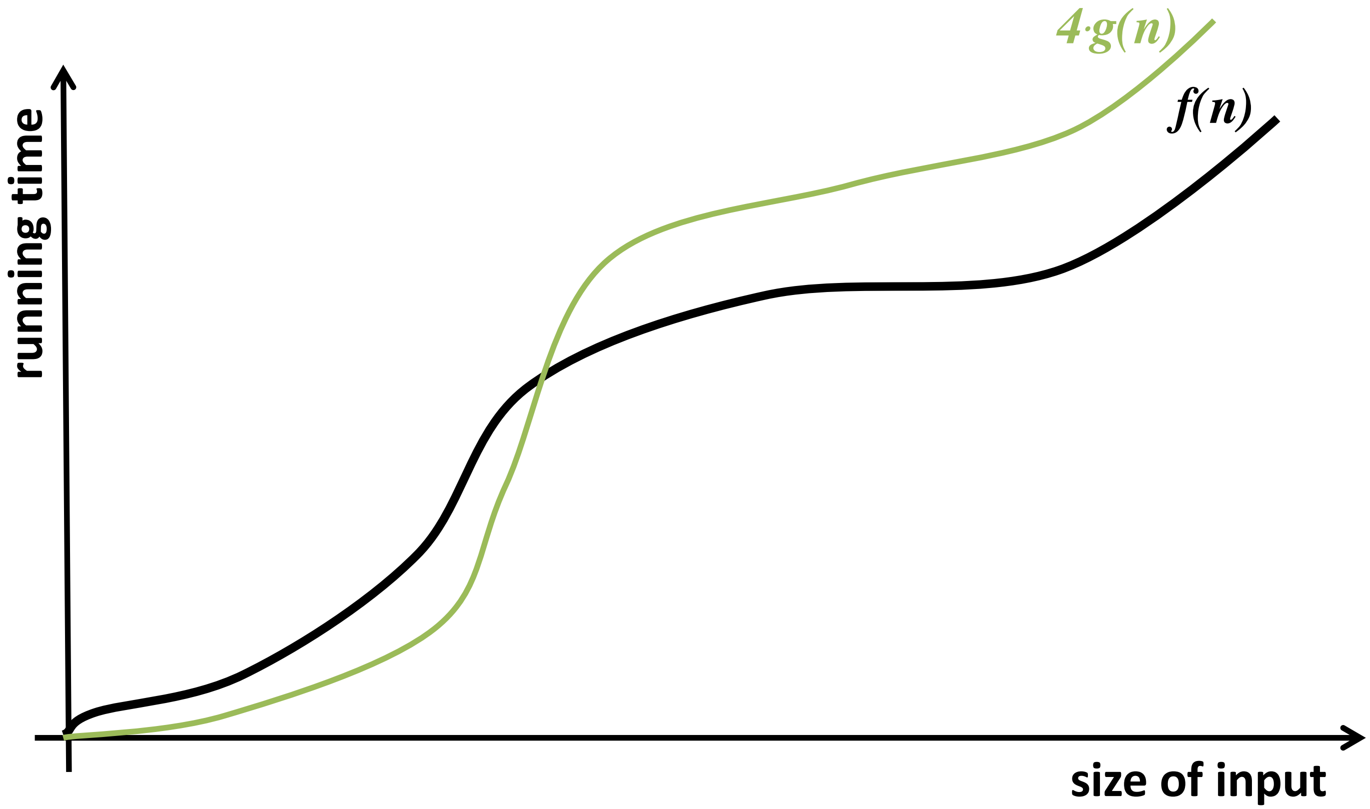


Asymptotic Analysis
Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\theta(g(n))$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0 such that $\mathbf{c_2 g(n) \leq f(n) \leq c_1 g(n)}$ for all $n \geq n_0$

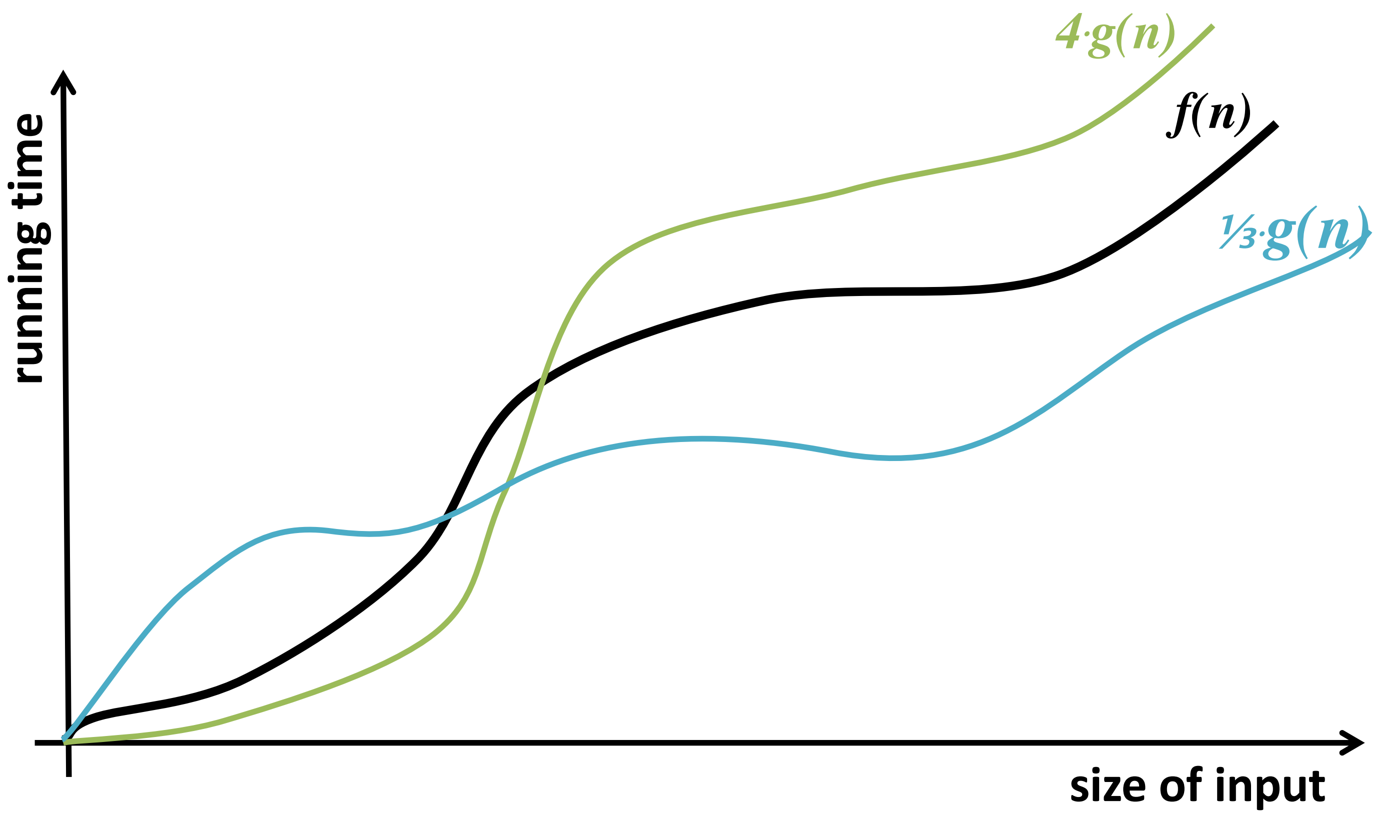


Asymptotic Analysis
Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\theta(g(n))$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0 such that $c_2 g(n) \leq f(n) \leq c_1 g(n)$ for all $n \geq n_0$



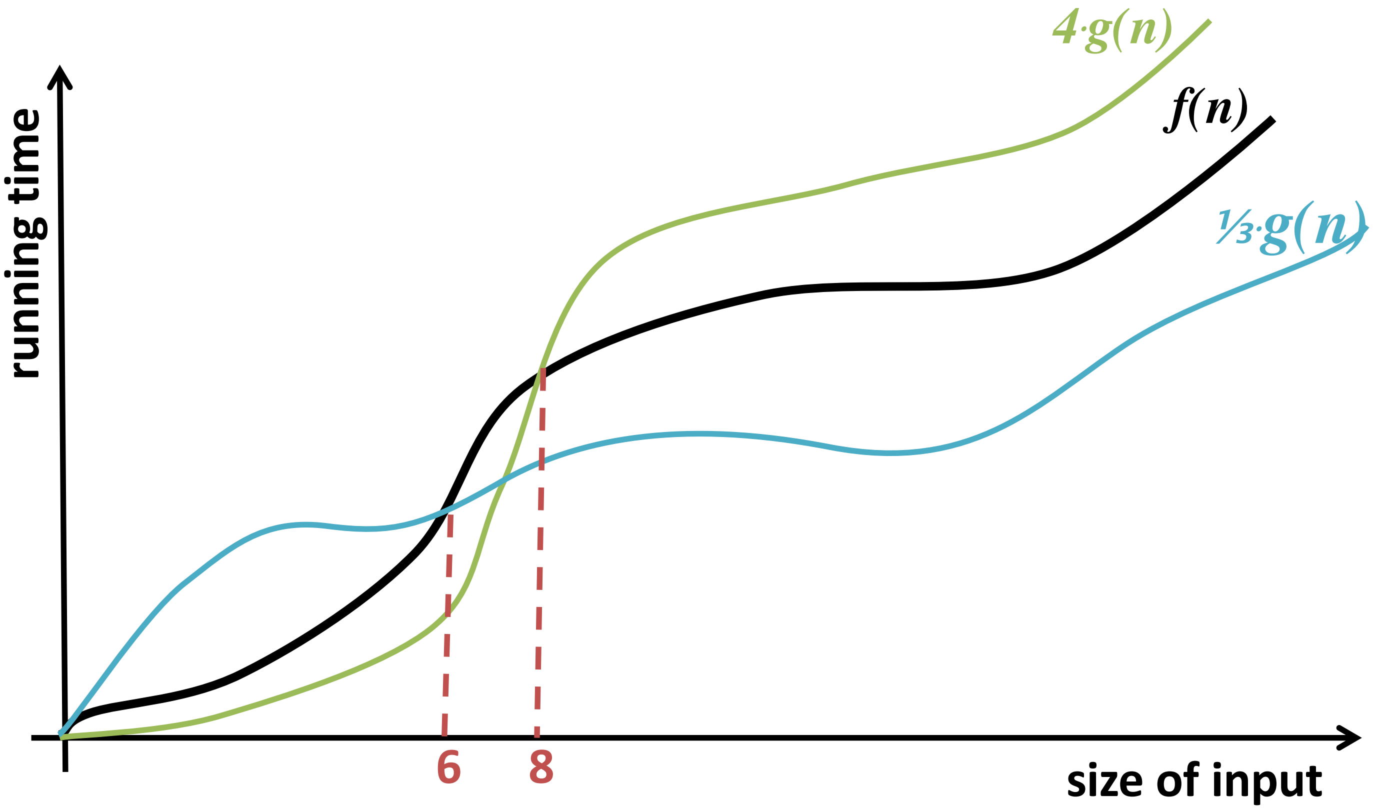
Asymptotic Analysis

Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\theta(g(n))$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0 such that $c_2 g(n) \leq f(n) \leq c_1 g(n)$ for all $n \geq n_0$



Asymptotic Analysis

Θ definition

Definition

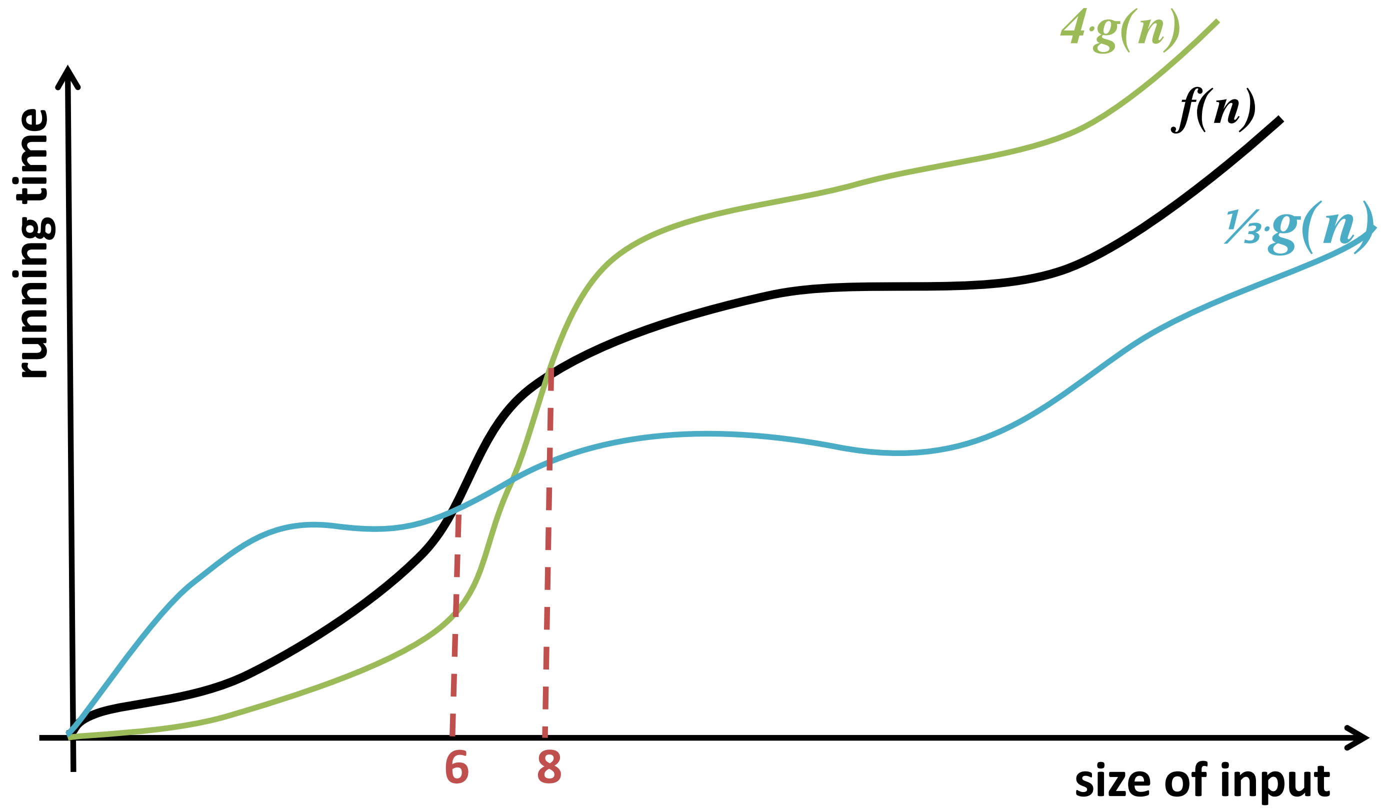
Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $\boldsymbol{f(n)=\theta(g(n))}$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0 such that $\boldsymbol{c_2 g(n) \leq f(n) \leq c_1 g(n)}$ for all $n \geq n_0$

$n_0 = 8$

$c_1 = 4$

$c_2 = ⅓$



Asymptotic Analysis
Θ definition

Definition

Let $f(n)$ and $g(n)$ be two functions mapping positive integers to positive real numbers.

We say that $f(n)=\theta(g(n))$ if there exist positive real constants c_1, c_2 and a positive integer constant n_0 such that $c_2 g(n) \leq f(n) \leq c_1 g(n)$ for all $n \geq n_0$

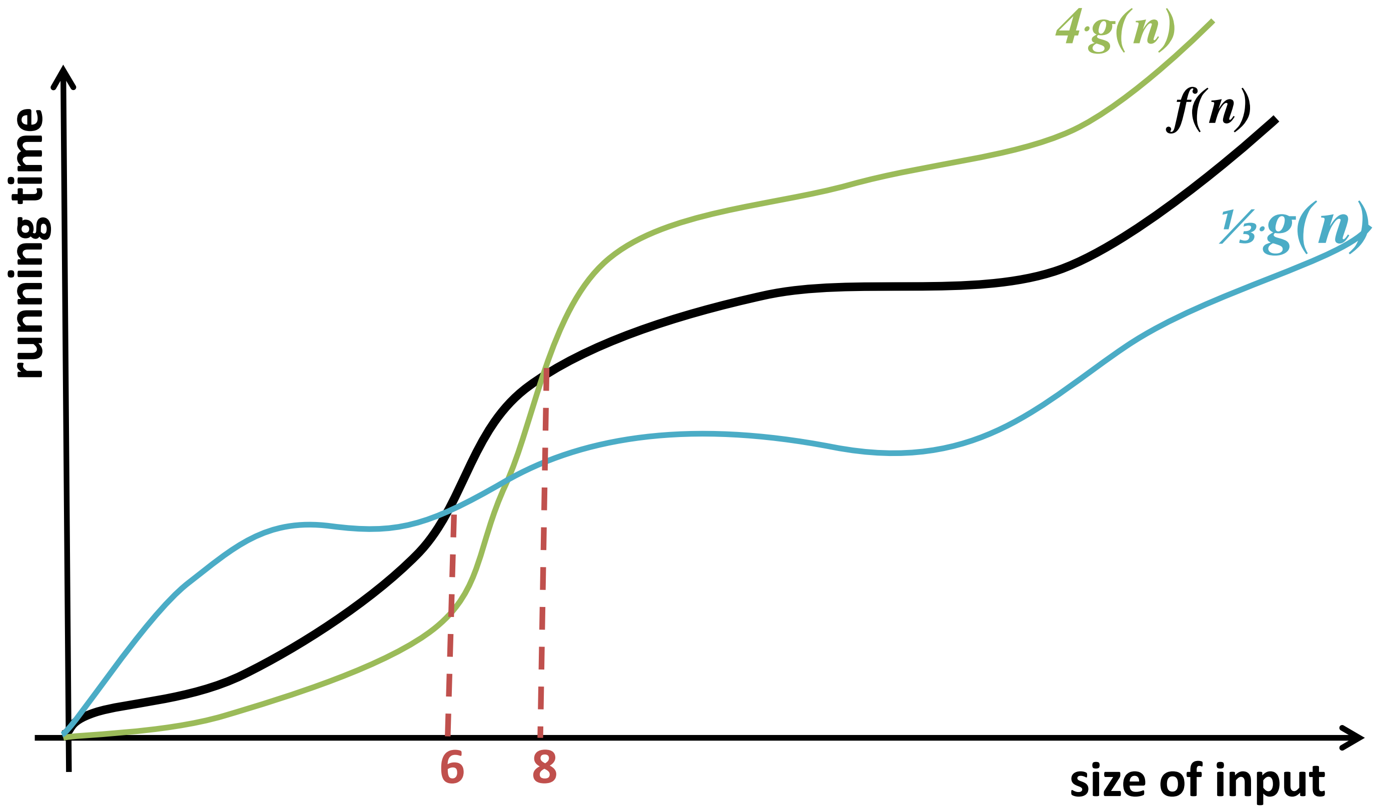
$n_0 = 8$

$c_1 = 4$

$c_2 = ⅓$

⇩

$f(n) = \theta(g(n))$



Asymptotic Analysis

Θ definition

Show that: $3n^2 + 6n - 15 = \theta(n^2)$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take c_1 = _____

c_2 = _____

n_0 = _____

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take c_1 = ____

c_2 = ____

n_0 = ____

Then for all $n \geq n_0$ we have:

Asymptotic Analysis

Θ definition

<u>Show that</u>: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

<u>Proof</u>:

if we take $c_1 =$ _____

$c_2 =$ _____

$n_0 =$ _____

Then for all $n \geq n_0$ we have:

$$3n^2 + 6n - 15$$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take c_1 = _____

c_2 = _____

n_0 = _____

Then for all $n \geq n_0$ we have:

$$3n^2 + 6n - 15 \leq 3n^2 + 6n$$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take c_1 = _____

c_2 = _____

n_0 = _____

Then for all $n \geq n_0$ we have:

$$3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2$$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take c_1 = _____

c_2 = _____

n_0 = _____

Then for all $n \geq n_0$ we have:

$$3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take $c_1 = \underline{\ 9\ \ \ \ }$

$c_2 = \underline{\ \ \ \ \ \ }$

$n_0 = \underline{\ \ \ \ \ \ }$

Then for all $n \geq n_0$ we have:

$$3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

Asymptotic Analysis

Θ definition

<u>Show that</u>: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

<u>Proof</u>:

if we take

$c_1 =$ $\underline{\;9\;\;\;\;}$

$c_2 =$ _____

$n_0 =$ _____

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take $c_1 =$ $\underline{\ 9\ }$

$c_2 =$ ____

$n_0 =$ ____

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

$6n - 15 \geq 0$

Asymptotic Analysis

Θ definition

<u>Show that</u>: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

<u>Proof</u>:

if we take $c_1 =$ $\underline{\ 9\ }$

$c_2 =$ _____

$n_0 =$ _____

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

$6n - 15 \geq 0$

$\Updownarrow$

$6n \geq 15$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take $c_1 =$ 9

$c_2 =$ ____

$n_0 =$ ____

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

$6n - 15 \geq 0$

$\Updownarrow$

$6n \geq 15$

$\Updownarrow$

$n \geq 2.5$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take

$c_1 =$ 9

$c_2 =$ _____

$n_0 =$ 3

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

$$6n - 15 \geq 0$$
$$\Updownarrow$$
$$6n \geq 15$$
$$\Updownarrow$$
$$n \geq 2.5$$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take $c_1 = \underline{\ 9\ }$

$c_2 = \underline{\ 3\ }$

$n_0 = \underline{\ 3\ }$

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

$6n - 15 \geq 0$

$\Updownarrow$

$6n \geq 15$

$\Updownarrow$

$n \geq 2.5$

Asymptotic Analysis

Θ definition

Show that: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

Proof:

if we take $c_1 = \underline{\ 9\ }$

$c_2 = \underline{\ 3\ }$

$n_0 = \underline{\ 3\ }$

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

$6n - 15 \geq 0$

$\Updownarrow$

$6n \geq 15$

$\Updownarrow$

$n \geq 2.5$

$$\Downarrow$$

$$3n^2 \leq 3n^2 + 6n - 15 \leq 9n^2$$

Asymptotic Analysis

Θ definition

<u>Show that</u>: $\underbrace{3n^2 + 6n - 15}_{f(n)} = \theta(\underbrace{n^2}_{g(n)})$

<u>Proof</u>:

if we take $c_1 = \underline{\ 9\ }$

$c_2 = \underline{\ 3\ }$

$n_0 = \underline{\ 3\ }$

Then for all $n \geq n_0$ we have:

$$3n^2 \leq 3n^2 + 6n - 15 \leq 3n^2 + 6n \leq 3n^2 + 6n^2 = 9n^2$$

$6n - 15 \geq 0$

$\Updownarrow$

$6n \geq 15$

$\Updownarrow$

$n \geq 2.5$

$$\Downarrow$$

$$3n^2 \leq 3n^2 + 6n - 15 \leq 9n^2$$

Therefore: $3n^2 + 6n - 15 = \theta(n^2)$

The Searching Problem

The Searching Problem

Problem

Implement the following function:

```
def linear_search(lst, val)
```

The Searching Problem

Problem

Implement the following function:

```
def linear_search(lst, val)
```

The function should return an index in lst, where val appears first

The Searching Problem

<u>Problem</u>

Implement the following function:

```
def linear_search(lst, val)
```

The function should <u>return an index</u> in lst, where val appears first, or **None** if val is not one of lst's elements.

The Searching Problem

Problem

Implement the following function:

```
def linear_search(lst, val)
```

The function should return an index in lst, where val appears first, or **None** if val is not one of lst's elements.

Examples

If lst is: [5, 8, 12, 7, 8, 10]

The Searching Problem

Problem

Implement the following function:

```
def linear_search(lst, val)
```

The function should return an index in lst, where val appears first, or **None** if val is not one of lst's elements.

Examples

If lst is: [5, 8, 12, 7, 8, 10]

- The call: linear_search(lst, 8)

The Searching Problem

Problem

Implement the following function:

```
def linear_search(lst, val)
```

The function should return an index in lst, where val appears first, or **None** if val is not one of lst's elements.

Examples

If lst is: [5, 8, 12, 7, 8, 10]

- The call: linear_search(lst, 8) should return 1

The Searching Problem

Problem

Implement the following function:

```
def linear_search(lst, val)
```

The function should return an index in lst, where val appears first, or **None** if val is not one of lst's elements.

Examples

If lst is: [5, 8, 12, 7, 8, 10]

- The call: linear_search(lst, 8) should return 1
- The call: linear_search (lst, 4)

The Searching Problem

Problem

Implement the following function:

```
def linear_search(lst, val)
```

The function should return an index in lst, where val appears first, or **None** if val is not one of lst's elements.

Examples

If lst is: [5, 8, 12, 7, 8, 10]

- The call: linear_search(lst, 8) should return 1
- The call: linear_search (lst, 4) should return **None**

Linear Search

```
def linear_search(lst, val):
```

Linear Search

```
def  linear_search(lst,  val):
        for __ in  _______________:
```

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
```

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
```

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

$\Theta(1)$

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

$\Theta(1)$

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

$\Theta(1)$

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

$\Theta(1)$

$\Theta(1)$

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

Θ(# of iterations)

Θ(1)

Θ(1)

Linear Search

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

Θ(# of iterations)

$\Theta(1)$

$\Theta(1)$

- $T(n) = \Theta(\text{\# of iterations})$

Linear Search

Θ(# of iterations)

Θ(1)

```
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

Θ(1)

- *$T(n) = \Theta(\text{\# of iterations})$*
- *In worst-case: (# of iterations) = n*

Linear Search

```python
def linear_search(lst, val):
    for i in range(len(lst)):
        if (lst[i] == val):
            return i
    return None
```

$\Theta(\text{\# of iterations})$ — for loop

$\Theta(1)$ — if body

$\Theta(1)$ — return None

- $T(n) = \Theta(\text{\# of iterations})$
- In worst-case: $(\text{\# of iterations}) = n$

$\Downarrow$

$$T_{worst}(n) = \Theta(n)$$

The Sorted-Search Problem

The Sorted-Search Problem

Problem

Implement the following function:

```
def  sorted_search(srt_lst,  val)
```

The Sorted-Search Problem

Problem

Implement the following function:

```
def  sorted_search(srt_lst,  val)
```

The function is given a **sorted** list srt_lst, and val to search for

The Sorted-Search Problem

Problem

Implement the following function:

```
def  sorted_search(srt_lst,  val)
```

The function is given a **sorted** list srt_lst, and val to search for. It should return an index, where val appears, or **None** if val is not one of srt_lst's elements.

The Sorted-Search Problem

Problem

Implement the following function:

```
def  sorted_search(srt_lst,  val)
```

The function is given a **sorted** list srt_lst, and val to search for. It should return an index, where val appears, or **None** if val is not one of srt_lst's elements.

Example

If srt_lst is: [5, 7, 8, 8, 10, 12]

The Sorted-Search Problem

Problem

Implement the following function:

```
def  sorted_search(srt_lst,  val)
```

The function is given a **sorted** list srt_lst, and val to search for. It should return an index, where val appears, or **None** if val is not one of srt_lst's elements.

Example

If srt_lst is: [5, 7, 8, 8, 10, 12]

The call sortedSearch(srt_lst, 8)

The Sorted-Search Problem

Problem

Implement the following function:

```
def sorted_search(srt_lst, val)
```

The function is given a **sorted** list srt_lst, and val to search for. It should return an index, where val appears, or **None** if val is not one of srt_lst's elements.

Example

If srt_lst is: [5, 7, 8, 8, 10, 12]

The call sortedSearch(srt_lst, 8) could return 3

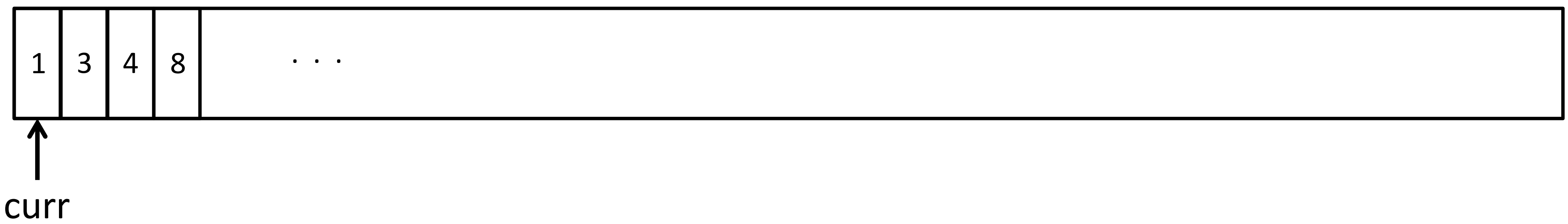
The Sorted-Search Problem

`val = 79`

1	3	4	8	. . .

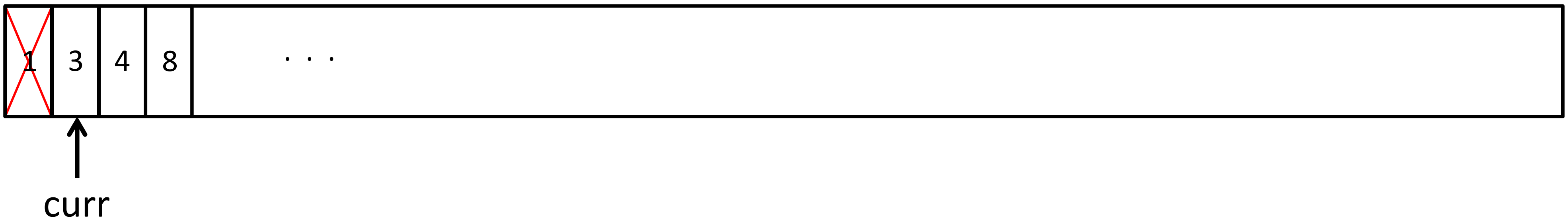
The Sorted-Search Problem

`val = 79`



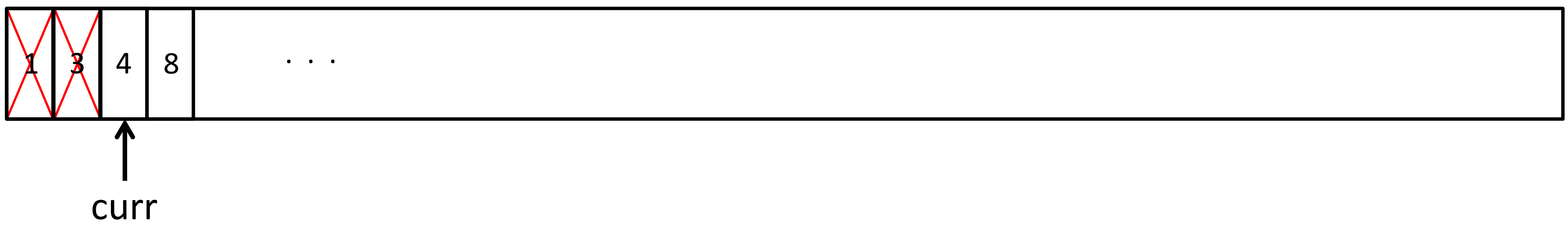
The Sorted-Search Problem

`val = 79`



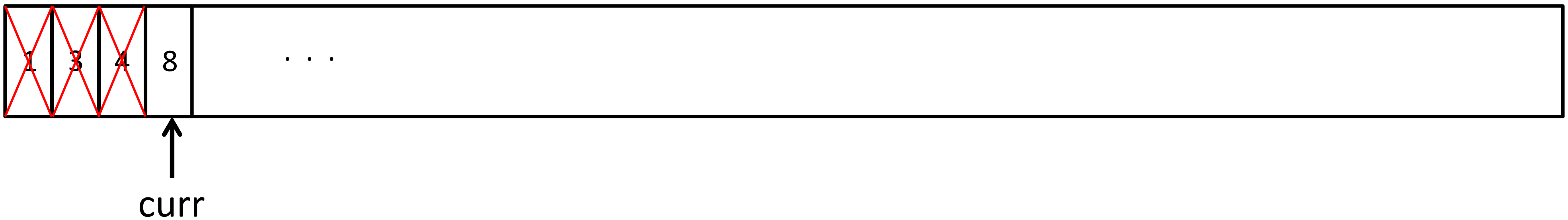
The Sorted-Search Problem

`val = 79`



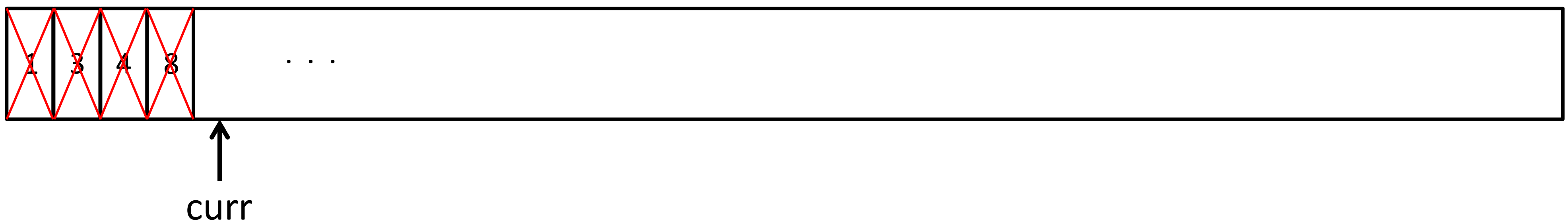
The Sorted-Search Problem

`val = 79`



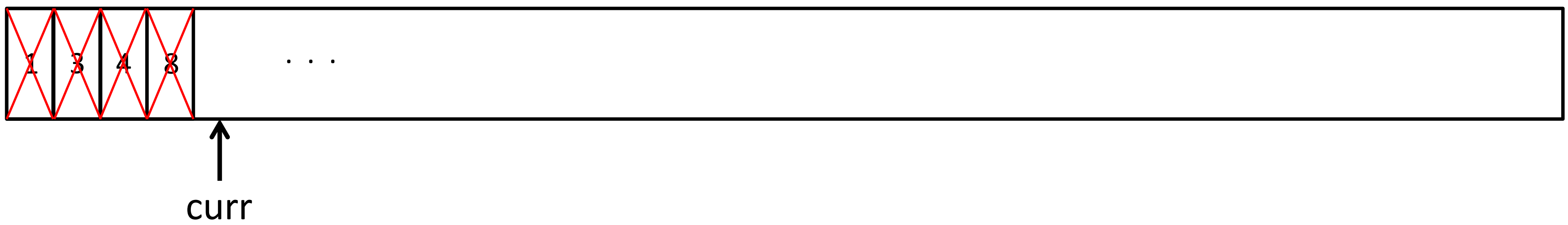
The Sorted-Search Problem

val = 79



The Sorted-Search Problem

`val = 79`



$$T(n) = \Theta(n)$$

The Sorted-Search Problem

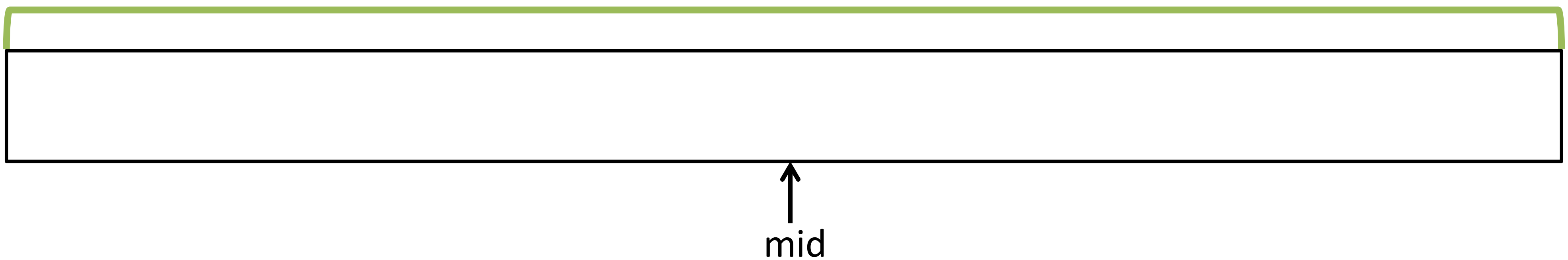
```
val = 79
```

The Sorted-Search Problem

`val = 79`

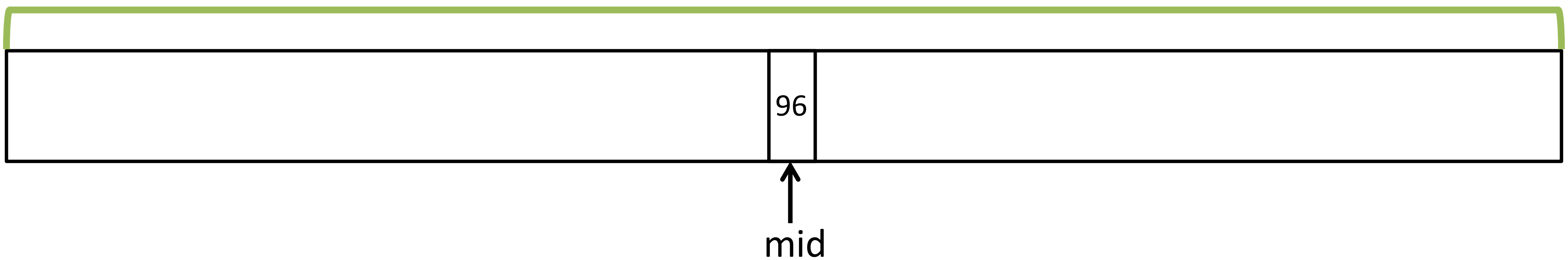
The Sorted-Search Problem

```
val = 79
```



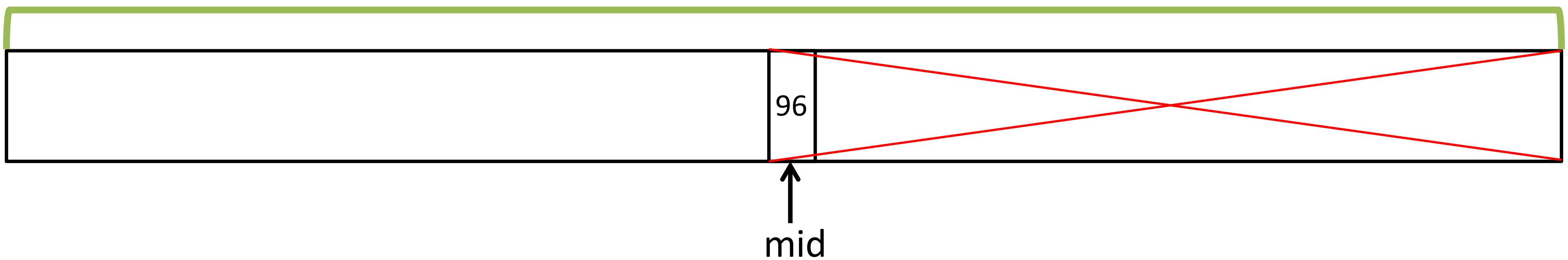
The Sorted-Search Problem

`val = 79`



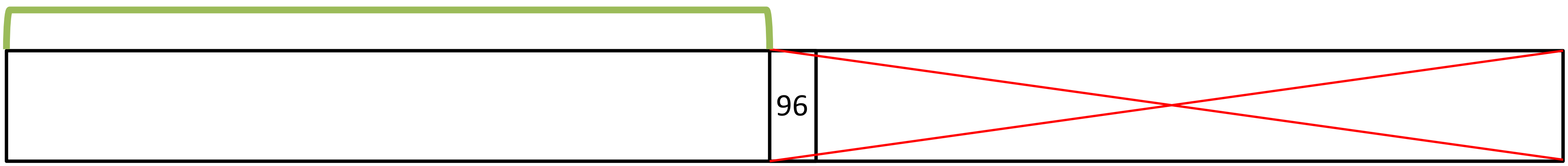
The Sorted-Search Problem

`val = 79`



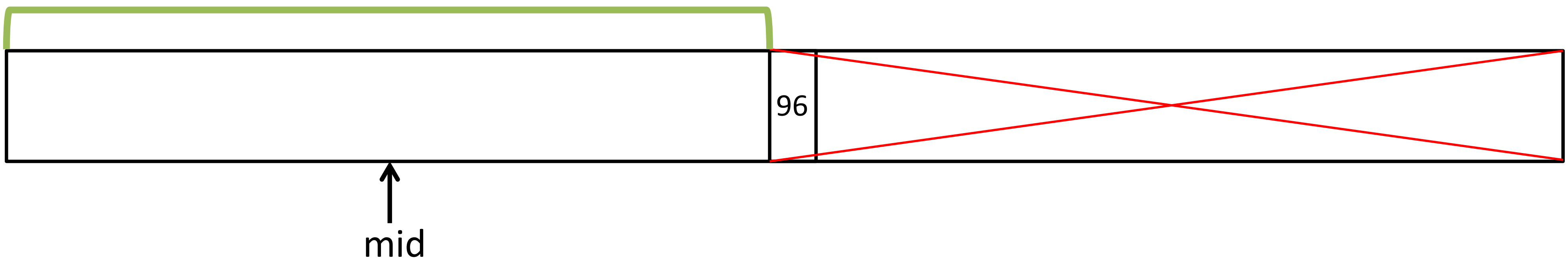
The Sorted-Search Problem

`val = 79`



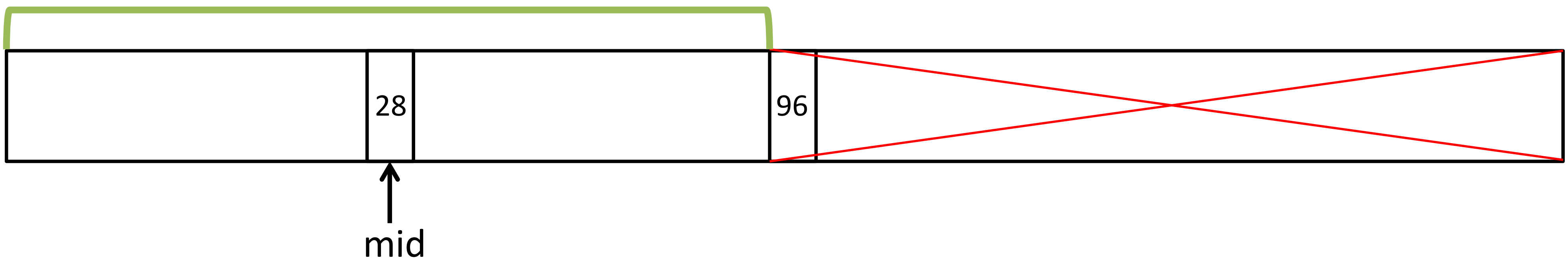
The Sorted-Search Problem

`val = 79`



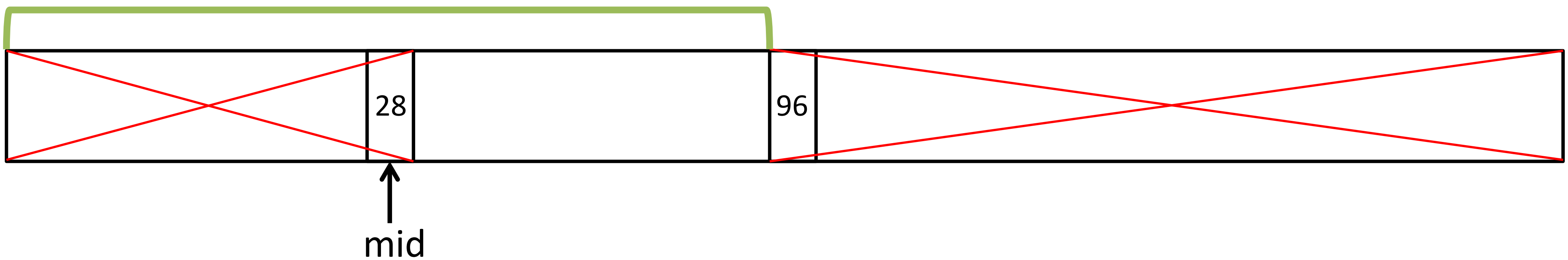
The Sorted-Search Problem

`val = 79`



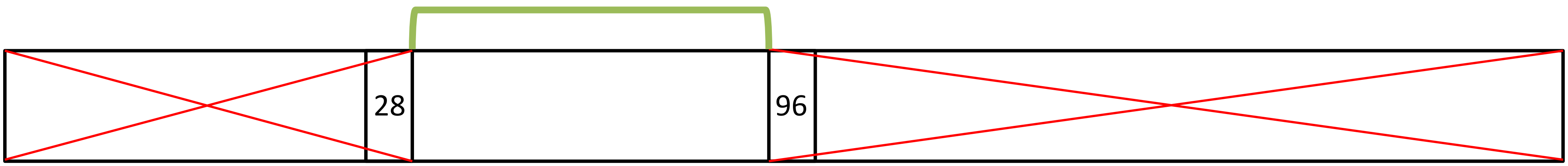
The Sorted-Search Problem

`val = 79`



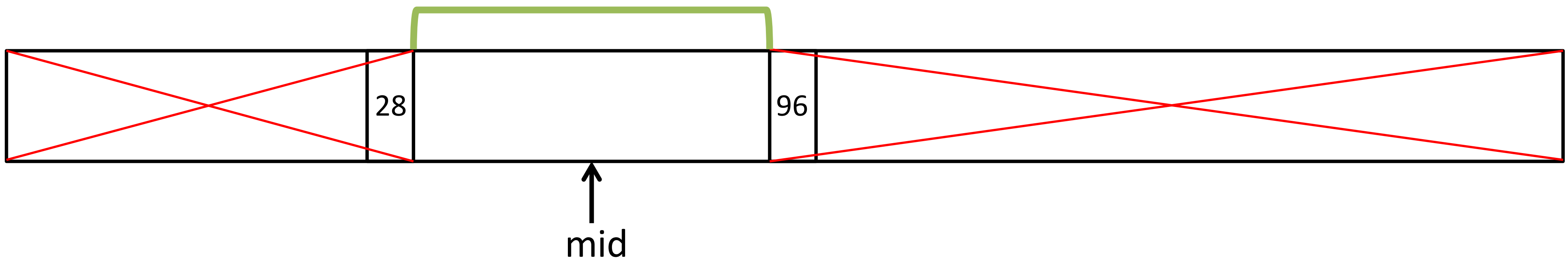
The Sorted-Search Problem

`val = 79`



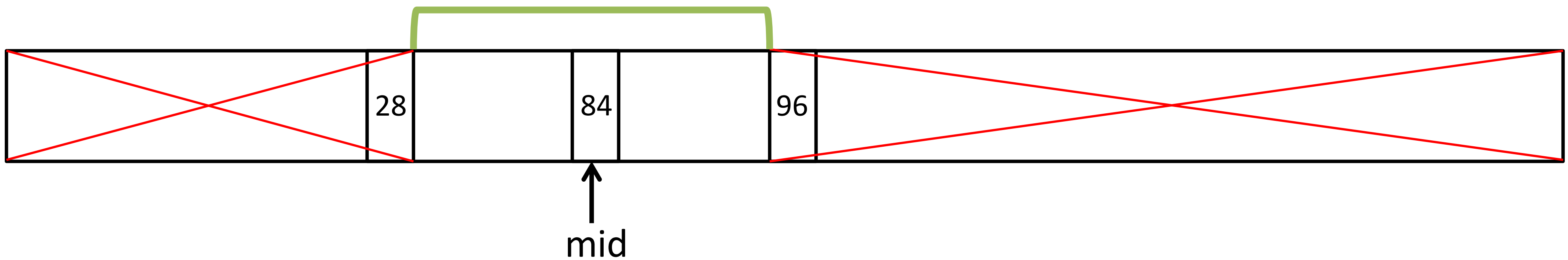
The Sorted-Search Problem

`val = 79`



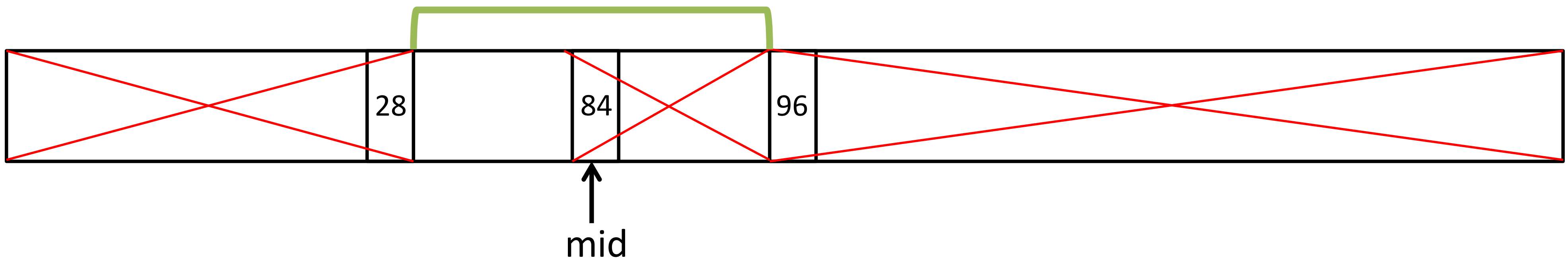
The Sorted-Search Problem

`val = 79`



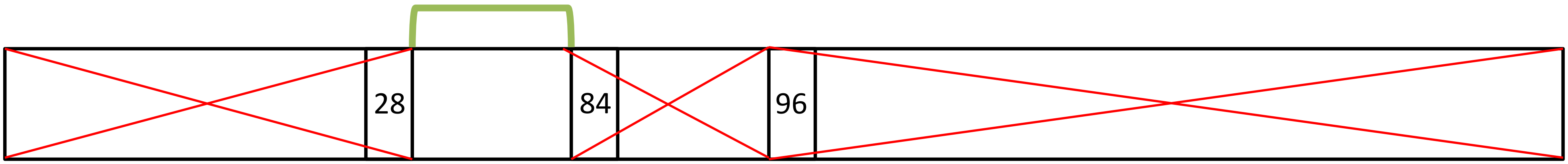
The Sorted-Search Problem

`val = 79`



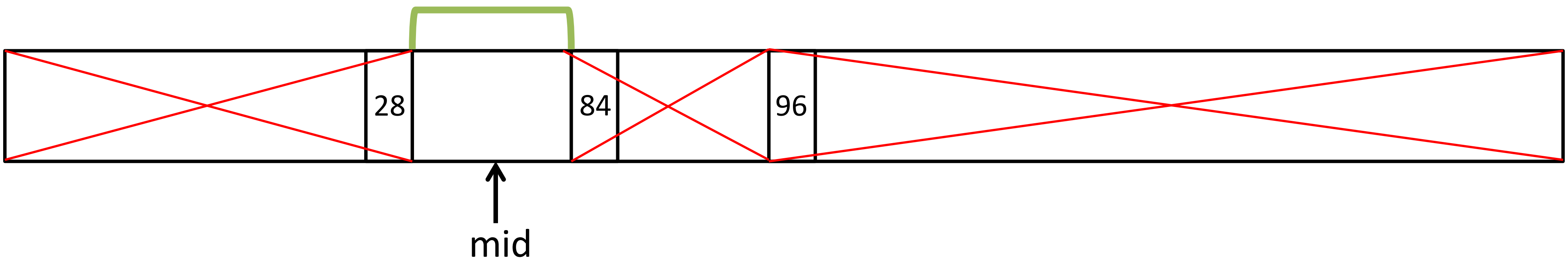
The Sorted-Search Problem

`val = 79`



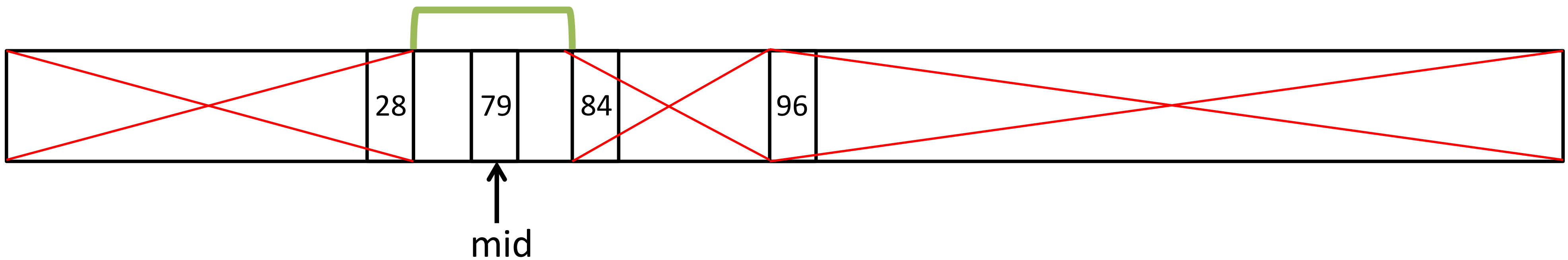
The Sorted-Search Problem

`val = 79`



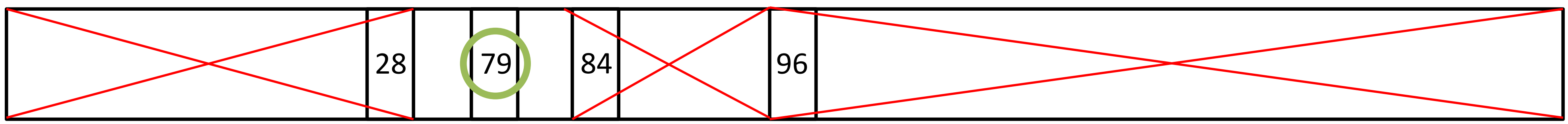
The Sorted-Search Problem

val = 79



The Sorted-Search Problem

`val = 79`



```python
def binary_search(srt_lst, val):
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
    while ((found == False) and (left <= right)):
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:  # val > srt_lst[mid]
            left = mid + 1
    return ind
```

$\Theta(1)$

```python
def binary_search(srt_lst, val):
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
    while ((found == False) and (left <= right)):
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:   # val > srt_lst[mid]
            left = mid + 1
    return ind
```

```python
def binary_search(srt_lst, val):
```

$\Theta(1)$

```python
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
```

```python
    while ((found == False) and (left <= right)):
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:    # val > srt_lst[mid]
            left = mid + 1
```

$\Theta(1)$

```python
    return ind
```

```python
def binary_search(srt_lst, val):
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
    while ((found == False) and (left <= right)):
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:   # val > srt_lst[mid]
            left = mid + 1
    return ind
```

$\Theta(1)$

$\Theta(1)$

```
def binary_search(srt_lst, val):
```

$\Theta(1)$

```
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
    while ((found == False) and (left <= right)):
```

$\Theta(1)$

```
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:   # val > srt_lst[mid]
            left = mid + 1
```

$\Theta(1)$

```
    return ind
```

```
      def binary_search(srt_lst, val):
          left = 0
          right = len(srt_lst) - 1
Θ(1)      ind = None
          found = False
          while ((found == False) and (left <= right)):
              mid = (left + right) // 2
              if (srt_lst[mid] == val):
                  ind = mid
                  found = True
Θ(1)          elif (val < srt_lst[mid]):
                  right = mid - 1
              else:   # val > srt_lst[mid]
                  left = mid + 1
Θ(1)      return ind
```

```python
def binary_search(srt_lst, val):
```

$\Theta(1)$

```python
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
```

$\Theta(\#\ of\ iterations)$

```python
    while ((found == False) and (left <= right)):
```

$\Theta(1)$

```python
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:   # val > srt_lst[mid]
            left = mid + 1
```

$\Theta(1)$

```python
    return ind
```

The Sorted-Search Problem

The Sorted-Search Problem

Iteration Number	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n}$
2	$\frac{\boldsymbol{n}}{2}$
3	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4}$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4}$
4	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4}$
4	$\frac{n}{8}$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4}$
4	$\frac{n}{8}$
$\vdots$	$\vdots$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4}$
4	$\frac{n}{8}$
$\vdots$	$\vdots$
k	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	n
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n} = \frac{\boldsymbol{n}}{2^0}$
2	$\frac{\boldsymbol{n}}{2} = \frac{\boldsymbol{n}}{2^1}$
3	$\frac{\boldsymbol{n}}{4} = \frac{\boldsymbol{n}}{2^2}$
4	$\frac{\boldsymbol{n}}{8} = \frac{\boldsymbol{n}}{2^3}$
$\vdots$	$\vdots$
k	

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$n = \frac{n}{2^0}$
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	$\frac{n}{2^{k-1}}$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$n = \frac{n}{2^0}$
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	$\frac{n}{2^{k-1}}$
$\vdots$	$\vdots$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$n = \frac{n}{2^0}$
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	$\frac{n}{2^{k-1}}$
$\vdots$	$\vdots$
	1

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n} = \frac{\boldsymbol{n}}{2^0}$
2	$\frac{\boldsymbol{n}}{2} = \frac{\boldsymbol{n}}{2^1}$
3	$\frac{\boldsymbol{n}}{4} = \frac{\boldsymbol{n}}{2^2}$
4	$\frac{\boldsymbol{n}}{8} = \frac{\boldsymbol{n}}{2^3}$
$\vdots$	$\vdots$
k	$\frac{\boldsymbol{n}}{2^{k-1}}$
$\vdots$	$\vdots$
$?$	1

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$n = \frac{n}{2^0}$
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	$\frac{n}{2^{k-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{n}{2^{k-1}} = 1$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$n = \frac{n}{2^0}$
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	$\frac{n}{2^{k-1}}$
$\vdots$	$\vdots$
$?$	1

$$\frac{n}{2^{k-1}} = 1$$

$$\Downarrow$$

$$n = 2^{k-1}$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$n = \frac{n}{2^0}$
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	$\frac{n}{2^{k-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{n}{2^{k-1}} = 1$$

$$\Downarrow$$

$$n = 2^{k-1}$$

$$\Downarrow$$

$$\log_2(n) = \log_2(2^{k-1})$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$n = \frac{n}{2^0}$
2	$\frac{n}{2} = \frac{n}{2^1}$
3	$\frac{n}{4} = \frac{n}{2^2}$
4	$\frac{n}{8} = \frac{n}{2^3}$
$\vdots$	$\vdots$
k	$\frac{n}{2^{k-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{n}{2^{k-1}} = 1$$

$$\Downarrow$$

$$n = 2^{k-1}$$

$$\Downarrow$$

$$\log_2(n) = \log_2(2^{k-1})$$

$$\Downarrow$$

$$\log_2(n) = (k-1)\log_2(2)$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n} = \frac{\boldsymbol{n}}{2^0}$
2	$\frac{\boldsymbol{n}}{2} = \frac{\boldsymbol{n}}{2^1}$
3	$\frac{\boldsymbol{n}}{4} = \frac{\boldsymbol{n}}{2^2}$
4	$\frac{\boldsymbol{n}}{8} = \frac{\boldsymbol{n}}{2^3}$
$\vdots$	$\vdots$
k	$\frac{\boldsymbol{n}}{2^{k-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{\boldsymbol{n}}{2^{k-1}} = 1$$

$$\Downarrow$$

$$\boldsymbol{n} = 2^{k-1}$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = \log_2(2^{k-1})$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = (\boldsymbol{k} - 1)\underbrace{\log_2(2)}_{=1}$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n} = \frac{\boldsymbol{n}}{2^0}$
2	$\frac{\boldsymbol{n}}{2} = \frac{\boldsymbol{n}}{2^1}$
3	$\frac{\boldsymbol{n}}{4} = \frac{\boldsymbol{n}}{2^2}$
4	$\frac{\boldsymbol{n}}{8} = \frac{\boldsymbol{n}}{2^3}$
$\vdots$	$\vdots$
k	$\frac{\boldsymbol{n}}{2^{k-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{\boldsymbol{n}}{2^{k-1}} = 1$$

$$\Downarrow$$

$$\boldsymbol{n} = 2^{k-1}$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = \log_2(2^{k-1})$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = (\boldsymbol{k} - 1)\underbrace{\log_2(2)}_{\shortparallel\atop 1}$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = (\boldsymbol{k} - 1)$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n} = \frac{\boldsymbol{n}}{2^0}$
2	$\frac{\boldsymbol{n}}{2} = \frac{\boldsymbol{n}}{2^1}$
3	$\frac{\boldsymbol{n}}{4} = \frac{\boldsymbol{n}}{2^2}$
4	$\frac{\boldsymbol{n}}{8} = \frac{\boldsymbol{n}}{2^3}$
$\vdots$	$\vdots$
k	$\frac{\boldsymbol{n}}{2^{k-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{\boldsymbol{n}}{2^{\boldsymbol{k}-1}} = 1$$
$$\Downarrow$$
$$\boldsymbol{n} = 2^{\boldsymbol{k}-1}$$
$$\Downarrow$$
$$\log_2(\boldsymbol{n}) = \log_2(2^{\boldsymbol{k}-1})$$
$$\Downarrow$$
$$\log_2(\boldsymbol{n}) = (\boldsymbol{k}-1)\log_2(2)$$
$$\Downarrow$$
$$\log_2(\boldsymbol{n}) = (\boldsymbol{k}-1)$$
$$\Downarrow$$
$$\boldsymbol{k} = 1 + \log_2(\boldsymbol{n})$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n} = \frac{\boldsymbol{n}}{2^0}$
2	$\frac{\boldsymbol{n}}{2} = \frac{\boldsymbol{n}}{2^1}$
3	$\frac{\boldsymbol{n}}{4} = \frac{\boldsymbol{n}}{2^2}$
4	$\frac{\boldsymbol{n}}{8} = \frac{\boldsymbol{n}}{2^3}$
$\vdots$	$\vdots$
k	$\frac{\boldsymbol{n}}{2^{\boldsymbol{k}-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{\boldsymbol{n}}{2^{\boldsymbol{k}-1}} = 1$$

$$\Downarrow$$

$$\boldsymbol{n} = 2^{\boldsymbol{k}-1}$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = \log_2(2^{\boldsymbol{k}-1})$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = (\boldsymbol{k}-1)\log_2(2)$$

$$\Downarrow$$

$$\log_2(\boldsymbol{n}) = (\boldsymbol{k}-1)$$

$$\Downarrow$$

$$\boldsymbol{k} = 1 + \log_2(\boldsymbol{n}) = \theta(\log_2(\boldsymbol{n}))$$

The Sorted-Search Problem

Iteration Number	Size of Searching-Range
1	$\boldsymbol{n} = \frac{\boldsymbol{n}}{2^0}$
2	$\frac{\boldsymbol{n}}{2} = \frac{\boldsymbol{n}}{2^1}$
3	$\frac{\boldsymbol{n}}{4} = \frac{\boldsymbol{n}}{2^2}$
4	$\frac{\boldsymbol{n}}{8} = \frac{\boldsymbol{n}}{2^3}$
$\vdots$	$\vdots$
k	$\frac{\boldsymbol{n}}{2^{k-1}}$
$\vdots$	$\vdots$
?	*1*

$$\frac{\boldsymbol{n}}{2^{\boldsymbol{k}-1}} = 1$$
$$\Downarrow$$
$$\boldsymbol{n} = 2^{\boldsymbol{k}-1}$$
$$\Downarrow$$
$$\log_2(\boldsymbol{n}) = \log_2(2^{\boldsymbol{k}-1})$$
$$\Downarrow$$
$$\log_2(\boldsymbol{n}) = (\boldsymbol{k}-1)\log_2(2)$$
$$\Downarrow$$
$$\log_2(\boldsymbol{n}) = (\boldsymbol{k}-1)$$
$$\Downarrow$$
$$\boldsymbol{k} = 1 + \log_2(\boldsymbol{n}) = \theta(\log_2(\boldsymbol{n}))$$
$$\Downarrow$$
$$\left(\begin{array}{c}\boldsymbol{\#\ of}\\ \boldsymbol{iterations}\end{array}\right) = \theta(\log_2(\boldsymbol{n}))$$

```python
def binary_search(srt_lst, val):
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
    while ((found == False) and (left <= right)):
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:   # val > srt_lst[mid]
            left = mid + 1
    return ind
```

$\Theta(1)$

Θ(# of iterations)

$\Theta(1)$

$\Theta(1)$

```python
def binary_search(srt_lst, val):
    left = 0
    right = len(srt_lst) - 1
    ind = None
    found = False
    while ((found == False) and (left <= right)):
        mid = (left + right) // 2
        if (srt_lst[mid] == val):
            ind = mid
            found = True
        elif (val < srt_lst[mid]):
            right = mid - 1
        else:   # val > srt_lst[mid]
            left = mid + 1
    return ind
```

$\Theta(1)$

$\Theta(\#$ of iterations$)$

$\Theta(1)$

$\Theta(1)$

$$T_{worst}(n) = \Theta(\log_2 n)$$

Linear vs. Logarithmic

Linear vs. Logarithmic

n	$\log_2(n)$

Linear vs. Logarithmic

n	$\log_2(n)$
2	

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2)$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2)$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$\log_2(2) = 1$
$\vdots$	$\vdots$
4	$\log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3)$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3) = 3$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$\log_2(2) = 1$
$\vdots$	$\vdots$
4	$\log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$\log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$log_2(2^{10})$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$\log_2(2) = 1$
$\vdots$	$\vdots$
4	$\log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$\log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$\log_2(2^{10}) = 10$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	$log_2(2^{32})$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	$log_2(2^{32}) = 32$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	$log_2(2^{32}) = 32$
$\vdots$	$\vdots$
2^{1000}	

Linear vs. Logarithmic

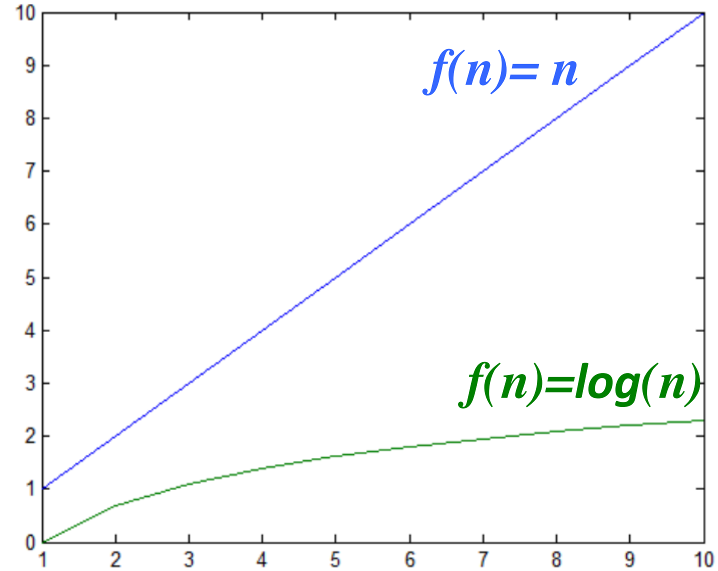
n	$\log_2(n)$
2	$log_2(2) = 1$
$\vdots$	$\vdots$
4	$log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	$log_2(2^{32}) = 32$
$\vdots$	$\vdots$
2^{1000}	$log_2(2^{1000})$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$\log_2(2) = 1$
$\vdots$	$\vdots$
4	$\log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$\log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$\log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	$\log_2(2^{32}) = 32$
$\vdots$	$\vdots$
2^{1000}	$\log_2(2^{1000}) = 1000$
$\vdots$	$\vdots$

Linear vs. Logarithmic

n	$\log_2(n)$
2	$\log_2(2) = 1$
$\vdots$	$\vdots$
4	$\log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$\log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$\log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	$\log_2(2^{32}) = 32$
$\vdots$	$\vdots$
2^{1000}	$\log_2(2^{1000}) = 1000$
$\vdots$	$\vdots$



Linear vs. Logarithmic

n	$\log_2(n)$
2	$\log_2(2) = 1$
$\vdots$	$\vdots$
4	$\log_2(2^2) = 2$
$\vdots$	$\vdots$
2^3	$\log_2(2^3) = 3$
$\vdots$	$\vdots$
2^{10}	$\log_2(2^{10}) = 10$
$\vdots$	$\vdots$
2^{32}	$\log_2(2^{32}) = 32$
$\vdots$	$\vdots$
2^{1000}	$\log_2(2^{1000}) = 1000$
$\vdots$	$\vdots$