

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 4

DIVIDE & CONQUER III :

ASYMPTOTIC NOTATION (CONTD.)

JULY 31, 2026

|

ROHIT VAISH

INTEGER MULTIPLICATION

Grade-school multiplication

$\leq 9n^2$ basic operations

Recursive algorithm (4 calls)

?

karatsuba algorithm (3 calls)

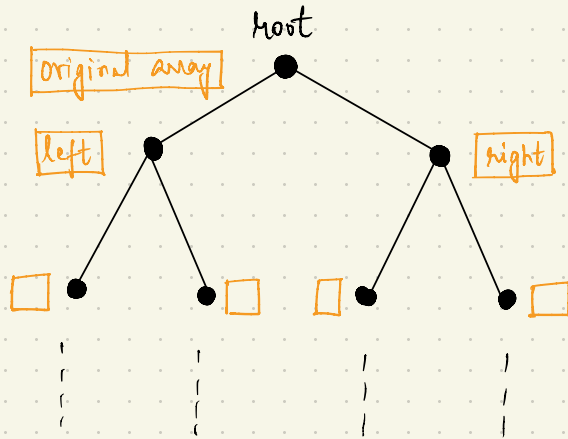
?

MERGE SORT

Theorem: For every input array of length $n \geq 1$, Merge Sort performs at most $6n \log_2 n + 6n$ operations.

MERGE SORT

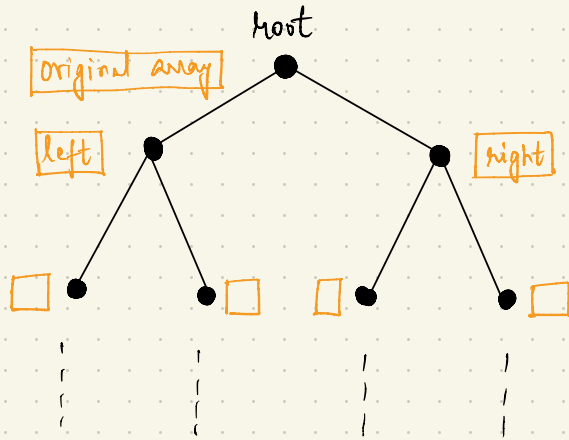
Theorem: For every input array of length $n \geq 1$, Merge Sort performs at most $6n \log_2 n + 6n$ operations.



.....
leaves (single-element arrays)

MERGE SORT

Theorem: For every input array of length $n \geq 1$, Merge Sort performs at most $6n \log_2 n + 6n$ operations.



$$\text{Work done at level } j = 2^j \times 6(n/2^j) = 6n$$

independent of j

THREE GUIDING PRINCIPLES

Worst-case (or adversarial) analysis

Not too worried about precise constants

Asymptotic analysis

THREE GUIDING PRINCIPLES

Worst-case (or adversarial) analysis

- no assumption on where the input comes from

Not too worried about precise constants

- transcend environment dependence
- mathematically easier and no loss in predictive power

Asymptotic analysis

- only large inputs are "interesting"

VOCABULARY: BIG OH NOTATION

VOCABULARY: BIG OH NOTATION

E.g. $6n \log_2 n + 6n$

VOCABULARY: BIG OH NOTATION

E.g. $6n \log_2 n + 6n$

Suppress constant factors and lower-order terms
system-dependent irrelevant for large inputs

VOCABULARY: BIG OH NOTATION

E.g. $6n \log_2 n + 6n$

Suppress constant factors and lower-order terms
system-dependent irrelevant for large inputs

equates with $n \log n$

VOCABULARY: BIG OH NOTATION

E.g. $6n \log_2 n + 6n$

Suppress constant factors and lower-order terms
system-dependent irrelevant for large inputs

equates with $n \log n$

The running time is $O(n \log n)$ "big-oh of $n \log n$ "

"order $n \log n$ "

VOCABULARY: BIG OH NOTATION

Sweet spot for reasoning
about algorithms



VOCABULARY: BIG OH NOTATION

sweet spot for reasoning
about algorithms



coarse enough to avoid environment-specific details

sharp enough to allow meaningful comparison among algorithms

Quick Examples

QUICK EXAMPLES

Searching for a number x in an array A of length n

for $i = 1$ to n

if $A[i] = x$

return TRUE

return FALSE

Running time : ?

QUICK EXAMPLES

Searching for a number x in an array A of length n

for $i = 1$ to n

if $A[i] = x$

return TRUE

return FALSE

Running time : $O(n)$

QUICK EXAMPLES

Searching for a number x in an array A of length n or
" " " B " " " .

```
for i = 1 to n  
    if  $A[i] = x$   
        return TRUE
```

```
for i = 1 to n  
    if  $B[i] = x$   
        return TRUE
```

```
return FALSE
```

Running time : ?

QUICK EXAMPLES

Searching for a number x in an array A of length n or
" " " B " " .

```
for i = 1 to n
    if A[i] = x
        return TRUE
```

```
for i = 1 to n
    if B[i] = x
        return TRUE
```

```
return FALSE
```

Running time : $O(n)$

QUICK EXAMPLES

Checking for a common element in arrays A and B

```
for i = 1 to n
  for j = 1 to n
    if A[i] = B[j]
      return TRUE
return FALSE
```

Running time : ?

QUICK EXAMPLES

Checking for a common element in arrays A and B

```
for i = 1 to n
  for j = 1 to n
    if A[i] = B[j]
      return TRUE
return FALSE
```

Running time : $O(n^2)$

QUICK EXAMPLES

Checking for a duplicate entry in array A

```
for i = 1 to n
  for j = i+1 to n
    if A[i] = A[j]
      return TRUE
return FALSE
```

Running time : ?

QUICK EXAMPLES

Checking for a duplicate entry in array A

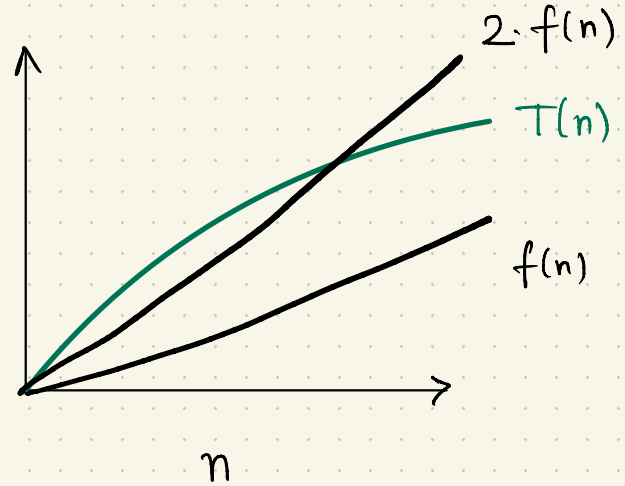
```
for i = 1 to n
  for j = i+1 to n
    if A[i] = A[j]
      return TRUE
return FALSE
```

Running time : $O(n^2)$

DEFINING BIG OH

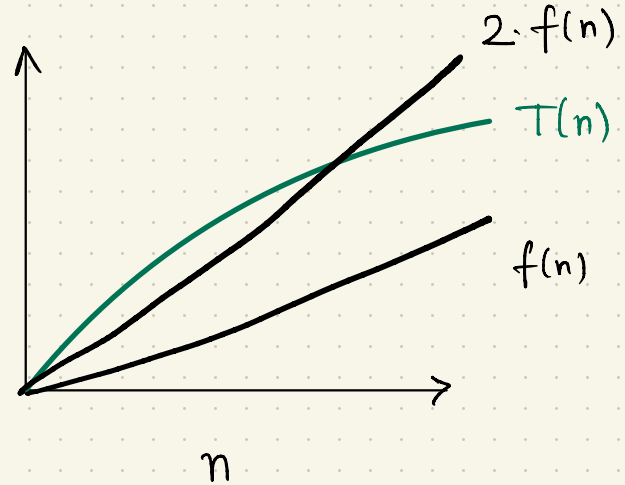
DEFINING BIG OH

$T(n)$ is "eventually bounded above"
by a constant multiple of $f(n)$.



DEFINING BIG OH

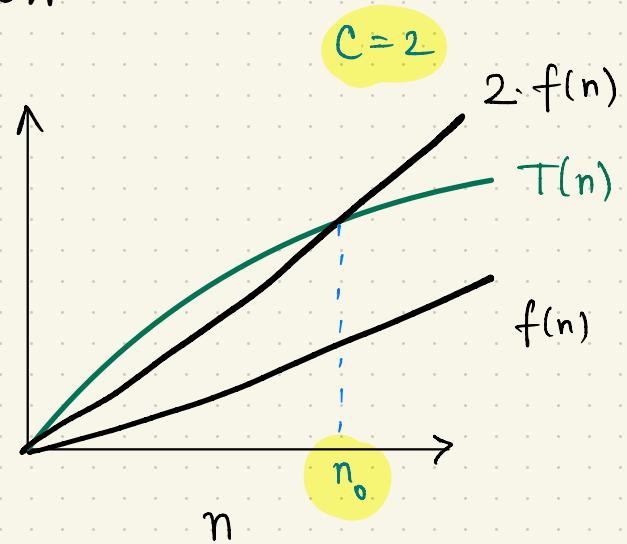
$T(n)$ is "eventually bounded above"
by a constant multiple of $f(n)$.



$T(n) = O(f(n))$ if there exist positive constants c and n_0
such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

DEFINING BIG OH

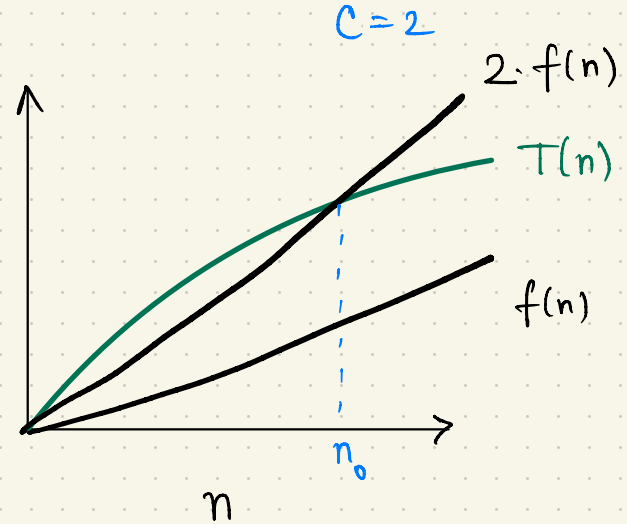
$T(n)$ is "eventually bounded above" by a constant multiple of $f(n)$.



$T(n) = O(f(n))$ if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

DEFINING BIG OH

NOTE : $O(f(n))$ is a set of functions



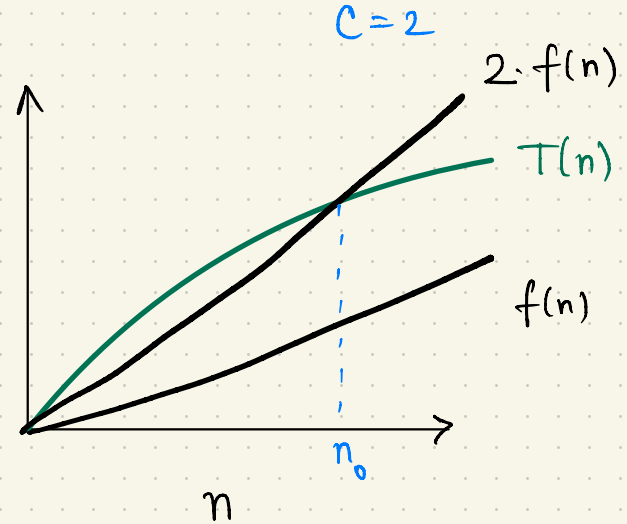
$T(n) = O(f(n))$ if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

DEFINING BIG OH

NOTE : $O(f(n))$ is a **set** of functions

Correct : $T(n) \in O(f(n))$

Common : $T(n) = O(f(n))$



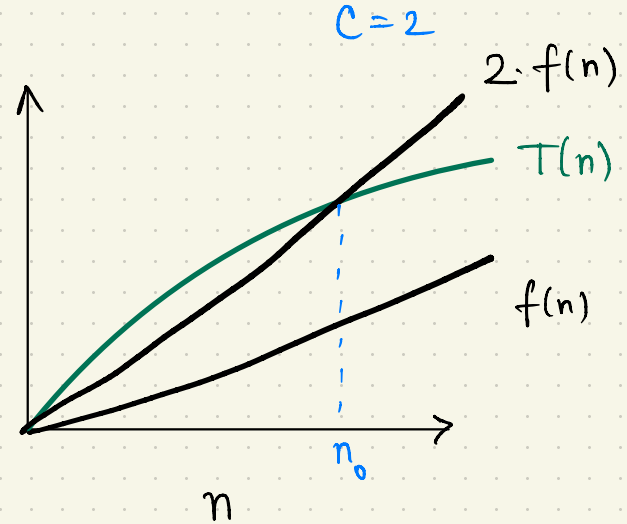
$T(n) = O(f(n))$ if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

DEFINING BIG OH

Game!

First, you pick c and n_0

Then, your opponent picks n .



$T(n) = O(f(n))$ if there exist positive constants c and n_0 such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

EXAMPLES

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof:

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof: Choose $n_0 = 1$ and $c = a_k + a_{k-1} + \dots + a_0$

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof: Choose $n_0 = 1$ and $c = \underbrace{a_k + a_{k-1} + \dots + a_0}_{\text{might be negative}}$

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof: Choose $n_0 = 1$ and $c = |a_k| + |a_{k-1}| + \dots + |a_0|$

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof: Choose $n_0 = 1$ and $c = |a_k| + |a_{k-1}| + \dots + |a_0|$.
Fix an arbitrary $n \geq n_0$.

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof: Choose $n_0 = 1$ and $c = |a_k| + |a_{k-1}| + \dots + |a_0|$.
Fix an arbitrary $n \geq n_0$.

$$T(n) \leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0|$$

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof: Choose $n_0 = 1$ and $c = |a_k| + |a_{k-1}| + \dots + |a_0|$

Fix an arbitrary $n \geq n_0$.

$$\begin{aligned} T(n) &\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0| \\ &\leq |a_k| n^k + |a_{k-1}| n^k + \dots + |a_1| n^k + |a_0| n^k \end{aligned}$$

EXAMPLES

Claim: If $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$,
then $T(n) = O(n^k)$.

Proof: Choose $n_0 = 1$ and $c = |a_k| + |a_{k-1}| + \dots + |a_0|$.
Fix an arbitrary $n \geq n_0$.

$$\begin{aligned} T(n) &\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0| \\ &\leq |a_k| n^k + |a_{k-1}| n^k + \dots + |a_1| n^k + |a_0| n^k \\ &= c \cdot n^k. \end{aligned}$$



EXAMPLES

Claim: If $T(n) = n^k$ then $T(n)$ is **not** $O(n^{k-1})$.

EXAMPLES

Claim: If $T(n) = n^k$ then $T(n)$ is **not** $O(n^{k-1})$.

Proof: (by contradiction)

Suppose $T(n) = O(n^{k-1})$.

EXAMPLES

Claim: If $T(n) = n^k$ then $T(n)$ is **not** $O(n^{k-1})$.

Proof: (by contradiction)

Suppose $T(n) = O(n^{k-1})$. Then, for some positive constants c and n_0 and for all $n \geq n_0$,

$$T(n) \leq c \cdot n^{k-1}$$

EXAMPLES

Claim: If $T(n) = n^k$ then $T(n)$ is **not** $O(n^{k-1})$.

Proof: (by contradiction)

Suppose $T(n) = O(n^{k-1})$. Then, for some positive constants c and n_0 and for all $n \geq n_0$,

$$T(n) \leq c \cdot n^{k-1}$$

$$\Rightarrow n \leq c.$$

EXAMPLES

Claim: If $T(n) = n^k$ then $T(n)$ is **not** $O(n^{k-1})$.

Proof: (by contradiction)

Suppose $T(n) = O(n^{k-1})$. Then, for some positive constants c and n_0 and for all $n \geq n_0$,

$$T(n) \leq c \cdot n^{k-1}$$

$$\Rightarrow n \leq c.$$

Contradiction!



BIG OMEGA & BIG THETA

BIG OMEGA & BIG THETA

$T(n) = \Omega(f(n))$ if there exist
positive constants c and n_0 such that

$$T(n) \geq c \cdot f(n)$$

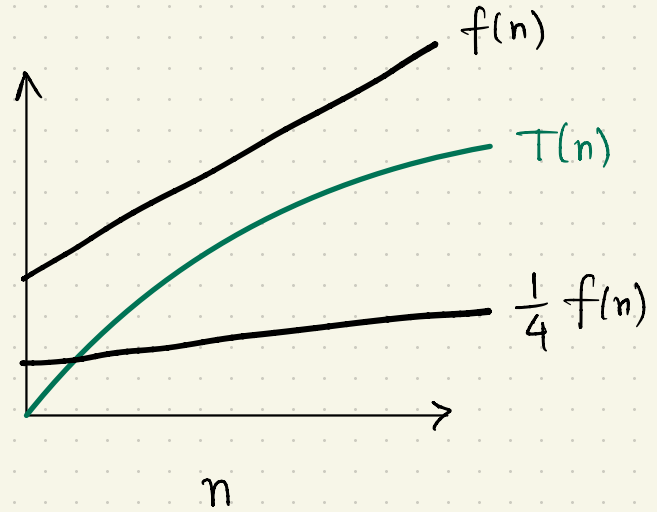
for all $n \geq n_0$.

BIG OMEGA & BIG THETA

$T(n) = \Omega(f(n))$ if there exist positive constants c and n_0 such that

$$T(n) \geq c \cdot f(n)$$

for all $n \geq n_0$.

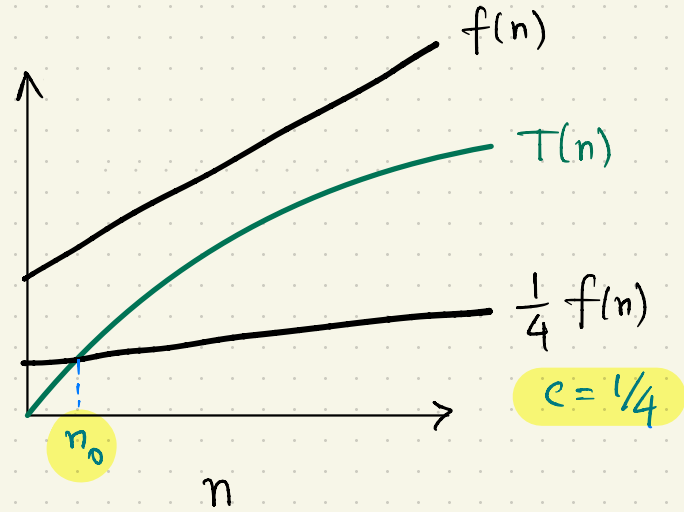


BIG OMEGA & BIG THETA

$T(n) = \Omega(f(n))$ if there exist positive constants c and n_0 such that

$$T(n) \geq c \cdot f(n)$$

for all $n \geq n_0$.

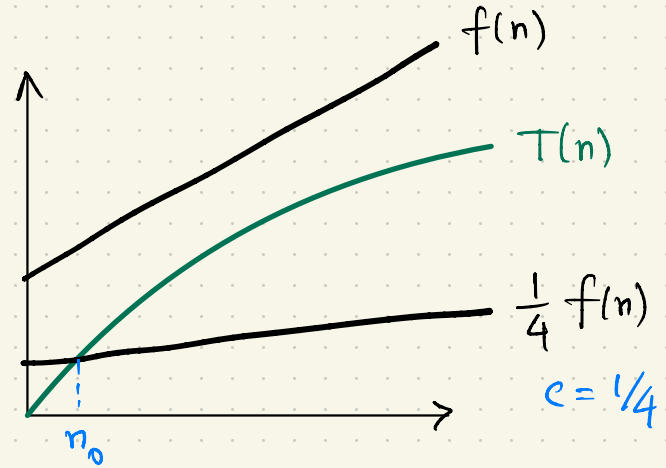


BIG OMEGA & BIG THETA

$T(n) = \Omega(f(n))$ if there exist positive constants c and n_0 such that

$$T(n) \geq c \cdot f(n)$$

for all $n \geq n_0$.



$T(n) = \Theta(f(n))$ if there exist positive constants c_1, c_2 , and n_0 such that

$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$$

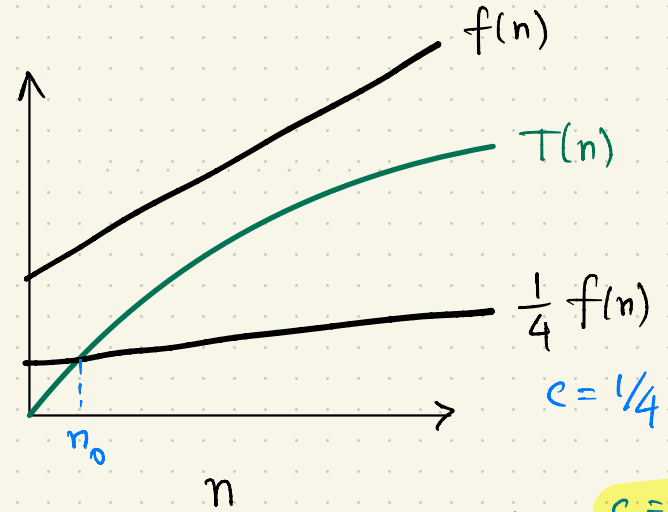
for all $n \geq n_0$.

BIG OMEGA & BIG THETA

$T(n) = \Omega(f(n))$ if there exist positive constants c and n_0 such that

$$T(n) \geq c \cdot f(n)$$

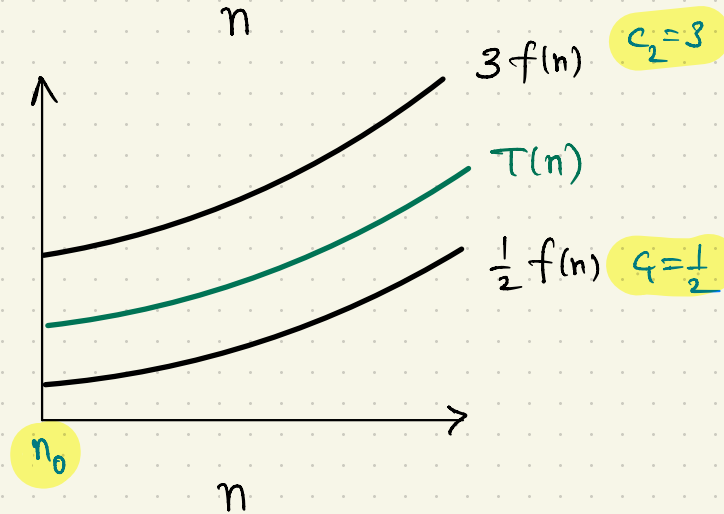
for all $n \geq n_0$.



$T(n) = \Theta(f(n))$ if there exist positive constants c_1, c_2 , and n_0 such that

$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$$

for all $n \geq n_0$.



BIG OMEGA & BIG THETA

$T(n) = \Omega(f(n))$ if there exist positive constants c and n_0 such that

$$T(n) \geq c \cdot f(n)$$

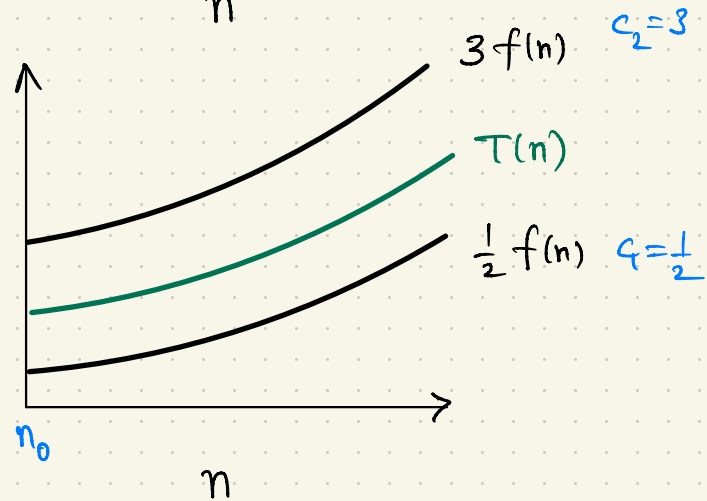
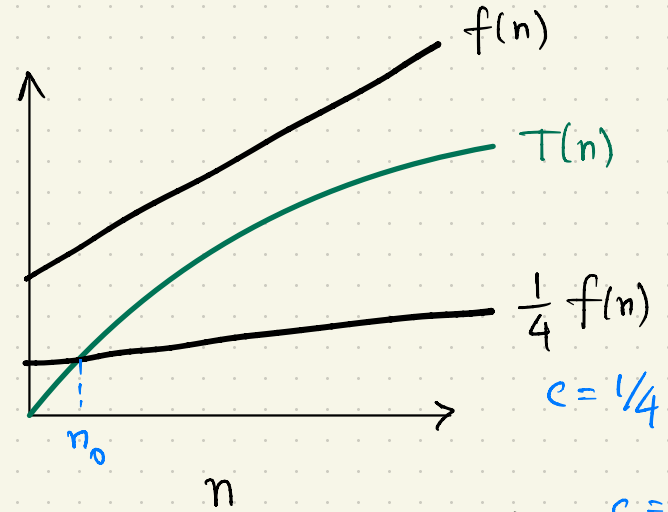
for all $n \geq n_0$.

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

$T(n) = \Theta(f(n))$ if there exist positive constants c_1, c_2 , and n_0 such that

$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$$

for all $n \geq n_0$.



EXAMPLES

$$\frac{n}{\log n} = O(n), \Omega(n), \Theta(n) ?$$

EXAMPLES

$$\frac{n}{\log n} = \cancel{\checkmark} O(n), \Omega(n), \Theta(n) ?$$

EXAMPLES

$$\frac{n}{\log n} = \cancel{O(n)}, \Omega(n), \Theta(n) ?$$

$$\frac{n}{100} = O(n), \Omega(n), \Theta(n) ?$$

EXAMPLES

$$\frac{n}{\log n} = \checkmark O(n), \Omega(n), \Theta(n) ?$$

$$\frac{n}{100} = \checkmark O(n), \checkmark \Omega(n), \checkmark \Theta(n) ?$$

EXAMPLES

$$\frac{n}{\log n} = \checkmark O(n), \Omega(n), \Theta(n) ?$$

$$\frac{n}{100} = \checkmark O(n), \checkmark \Omega(n), \checkmark \Theta(n) ?$$

$$n^2 = O(n), \Omega(n), \Theta(n) ?$$

EXAMPLES

$$\frac{n}{\log n} = \cancel{O(n)}, \Omega(n), \Theta(n) ?$$

$$\frac{n}{100} = \cancel{O(n)}, \cancel{\Omega(n)}, \cancel{\Theta(n)} ?$$

$$n^2 = O(n), \cancel{\Omega(n)}, \Theta(n) ?$$

EXAMPLES

$$\frac{n}{\log n} = \checkmark O(n), \Omega(n), \Theta(n) ?$$

$$\frac{n}{100} = \checkmark O(n), \checkmark \Omega(n), \checkmark \Theta(n) ?$$

$$n^2 = O(n), \checkmark \Omega(n), \Theta(n) ?$$

Others : $T(n) = o(f(n))$
"little oh"

$T(n) = \omega(f(n))$
"little omega"

self-reading

BIG OMICRON AND BIG OMEGA AND BIG THETA

Donald E. Knuth
Computer Science Department
Stanford University
Stanford, California 94305

Well, I think I have beat this issue to death, knowing of no other arguments pro or con the introduction of Ω and Θ . On the basis of the issues discussed here, I propose that members of SIGACT, and editors of computer science and mathematics journals, adopt the O , Ω , and Θ notations as defined above, unless a better alternative can be found reasonably soon. Furthermore I propose that the relational notations of Hardy be adopted in those situations where a relational notation is more appropriate.