Big-Oh

Definition:

f(n) = O(g(n)) means

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \leq c \cdot g(n)$

Meaning: f grows no faster than g, up to constant factors

Big-Oh

Definition:

$f(n) = O(g(n))$ means

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \leq c \cdot g(n)$

Example 1:

$10n = O(n \log n)$?

$c =$?, $n_0 =$? such that $\forall n \geq n_0$, $10n \leq c \cdot n \log n$.

Big-Oh

Definition:

$f(n) = O(g(n))$ means

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \leq c \cdot g(n)$

Example 1:

$10n = O(n \log n)$.

$c = 10$, $n_0 = 2$ such that $\forall n \geq 2$, $10n \leq 10n \log n$.

Are there other options for c, n_0?

Big-Oh

Definition:

f(n) = O(g(n)) means

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \leq c \cdot g(n)$

Example 1:

10n = O(n log n).

c =10, n_0= 2 such that $\forall n \geq 2$, $10n \leq 10n \log n$.

c =1 , $n_0 = 2^{10}$ such that $\forall n \geq 2^{10}$, $10n \leq n \log n$.

Example 2:

$100n^2 = O(2^n)$?

$c = ?$, $n_0 = ?$ such that $\forall n \geq n_0$, $100\, n^2 \leq c \cdot 2^n$.

Example 2:

$100n^2 = O(2^n)$.

$c = 100$, $n_0 = 4$ such that $\forall n \geq 4$, $100n^2 \leq 100 \cdot 2^n$.

Are there other options for c, n_0?

Example 2:

$100n^2 = O(2^n)$.

$c = 100$, $n_0 = 4$ such that $\forall n \geq 4$, $100n^2 \leq 100 \cdot 2^n$.

$c = 25$, $n_0 = 8$ such that $\forall n \geq 8$, $100n^2 \leq 25.2^n$.

Example 2:

$100n^2 = O(2^n)$.

$c = 100$, $n_0 = 4$ such that $\forall n \geq 4$, $100n^2 \leq 100 \cdot 2^n$.

$c = 25$, $n_0 = 8$ such that $\forall n \geq 8$, $100n^2 \leq 25.2^n$.

Example 3:

$n^2 \log n = O(100\ n^2)$?

$c =$?, $n_0 =$? such that $\forall n \geq n_0$, $n^2 \log n \leq c \cdot 100\ n^2$.

Example 2:

$100n^2 = O(2^n)$

$c = 100$, $n_0 = 4$ such that $\forall n \geq 4$, $100n^2 \leq 100 \cdot 2^n$.

Example 3:

$n^2 \log n \neq O(100\ n^2)$.

$\forall c, n_0\ \exists\ n \geq n_0$ such that $n^2 \log_2 n > c \cdot 100\ n^2$.

$n > 2^{100c}$

$\Rightarrow\ n^2 \log_2 n > c \cdot 100n^2$.

Example 4:

$2^n = O(2^{n/2})$?

$c = ?$, $n_0 = ?$ such that $\forall\ n \geq n_0$, $2^n \leq c \cdot 2^{n/2}$.

Example 4:

$2^n \neq O(2^{n/2})$.

$\forall c, n_0 \exists n \geq n_0$ such that $2^n > c \cdot 2^{n/2}$.

$n > 2 \log_2 c$

$2^n = 2^{n/2}\ 2^{n/2} > c \cdot 2^{n/2}$.

- $n \log n = O(n^2)$?
- $n^2 = O(n^{1.5} \log 10n)$?
- $2^n = O(n^{1000000})$?
- $\sqrt{2}^{\log n} = O(n^{1/3})$?
- $n^{\log \log n} = O((\log n)^{\log n})$?
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 = O(n^{1.5} \log 10n)$?
- $2^n = O(n^{1000000})$?
- $\sqrt{2}^{\log n} = O(n^{1/3})$?
- $n^{\log \log n} = O((\log n)^{\log n})$?
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n = O(n^{1000000})$?
- $\sqrt{2}^{\log n} = O(n^{1/3})$?
- $n^{\log \log n} = O((\log n)^{\log n})$?
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$?
- $\sqrt{2}^{\log n} = O(n^{1/3})$?
- $n^{\log \log n} = O((\log n)^{\log n})$?
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} = O(n^{1/3})$? $\sqrt{2}^{\log n} = n^{1/2} \neq O(n^{1/3})$
- $n^{\log \log n} = O((\log n)^{\log n})$?
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$?
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$?
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

$n^{\log \log n} =$

$2^{\log n . \log \log n} =$

$(\log n)^{\log n}$.

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$.
- $2^n = O(4^{\log n})$?
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$.
- $2^n = O(4^{\log n})$? $4^{\log n} = 2^{2 \log n}$ $2^n = 2^{2^{\log n}}$.
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$.
- $2^n \neq O(4^{\log n})$.
- $n! = O(2^n)$?
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$.
- $2^n \neq O(4^{\log n})$.
- $n! \neq O(2^n)$. $2.5\sqrt{n}(n/e)^n \leq n! \leq 2.8\sqrt{n}(n/e)^n$
- $n! = O(n^n)$?
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$.
- $2^n \neq O(4^{\log n})$.
- $n! \neq O(2^n)$.
- $n! = O(n^n)$.
- $n2^n = O(2^{n \log n})$?

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$.
- $2^n \neq O(4^{\log n})$.
- $n! \neq O(2^n)$.
- $n! = O(n^n)$.
- $n2^n = O(2^{n \log n})$? $n2^n = 2^{\log n + n}$.

- $n \log n = O(n^2)$.
- $n^2 \neq O(n^{1.5} \log 10n)$.
- $2^n \neq O(n^{1000000})$.
- $\sqrt{2}^{\log n} \neq O(n^{1/3})$.
- $n^{\log \log n} = O((\log n)^{\log n})$.
- $2^n \neq O(4^{\log n})$.
- $n! \neq O(2^n)$.
- $n! = O(n^n)$.
- $n2^n = O(2^{n \log n})$.

Big-omega

Definition:

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

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \geq c \cdot g(n).$

Big-omega

Definition:

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

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \geq c \cdot g(n).$

Example 1:

$n = \Omega(10 \log n)$?

$c =$?, $n_0 =$? such that $\forall n \geq n_0$, $n \geq c \cdot 10 \log n$.

Big-omega

Definition:

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

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \geq c \cdot g(n).$

Example 1:

$n = \Omega(10 \log n)$

$c = 1$, $n_0 = 64$ such that $\forall n \geq 64$, $n \geq 10 \log n$.

Are there other options for c, n_0?

Big-omega

Definition:

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

$\exists\, n_0, c > 0 \quad \forall n \in N\ n \geq n_0, \quad f(n) \geq c \cdot g(n).$

Example 1:

$n = \Omega(10 \log n)$

$c = 1$, $n_0 = 64$ such that $\forall n \geq 64$, $n \geq 10 \log n$.

$c = 1/10$, $n_0 = 2$ such that $\forall n \geq 2$, $n \geq (1/10)10 \log n$.

Example 2:

$n^2/100 = \Omega(n \log n)$?

$c = ?$, $n_0 = ?$ such that $\forall n \geq n_0$, $n^2/100 \geq c \cdot n \log n$.

Example 2:

$n^2/100 = \Omega(n \log n)$.

$c = 1$, $n_0 = 2^{10}$ such that $\forall n \geq 2^{10}$, $n^2/100 \geq n \log n$.

Are there other options for c, n_0?

Example 2:

$n^2/100 = \Omega(n \log n)$.

$c = 1$, $n_0 = 2^{10}$ such that $\forall n \geq 2^{10}$, $n^2/100 \geq n \log n$.

$c = 1/100$, $n_0 = 2$ such that $\forall n \geq 2$, $n^2/100 \geq n \log n/100$.

Example 2:

$n^2/100 = \Omega(n \log n)$.

$c = 1$, $n_0 = 2^{10}$ such that $\forall n \geq 2^{10}$, $n^2/100 \geq n \log n$.

$c = 1/100$, $n_0 = 2$ such that $\forall n \geq 2$, $n^2/100 \geq n \log n/100$.

Example 3:

$\sqrt{n} = \Omega(n/100)$?

$c =$?, $n_0 =$? such that $\forall n \geq n_0$, $\sqrt{n} \geq c \cdot n/100$.

Example 2:

$n^2/100 = \Omega(n \log n)$.

$c = 1$, $n_0 = 2^{10}$ such that $\forall n \geq 2^{10}$, $n^2/100 \geq n \log n$.

$c = 1/100$, $n_0 = 2$ such that $\forall n \geq 2$, $n^2/100 \geq n \log n/100$.

Example 3:

$\sqrt{n} \neq \Omega(n/100)$

$\forall c, n_0 \; \exists \; n \geq n_0$ such that , $\sqrt{n} < c \cdot n/100$.

Example 4:

$2^{n/2} = \Omega(2^n)$?

c = ?, n_0 = ? such that $\forall\ n \geq n_0$, $2^{n/2} \geq c \cdot 2^n$.

Example 4:

$2^{n/2} \neq \Omega(2^n)$

$\forall c, n_0 \; \exists \; n \geq n_0$ such that $2^{n/2} < c \cdot 2^n$.

Big-omega, Big-Oh

Note: $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$

$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$.

Example:

$n = \Omega(10 \log n)$.

$c = 1$, $n_0 = 6$ such that $\forall n \geq 6$, $n \geq 10 \log n$.

$10 \log n = O(n)$.

$c = 1$, $n_0 = 6$ such that $\forall n \geq 6$ $10 \log n \leq n$

Theta

Definition:

$f(n) = \Theta(g(n))$ means

$\exists\, n_0, c_1, c_2 > 0 \quad \forall n \in N\ n \geq n_0,$

$f(n) \leq c_1 \cdot g(n)$ and $g(n) \leq c_2 \cdot f(n)$.

Theta

Definition:

$f(n) = \Theta(g(n))$ means

$\exists\, n_0, c_1, c_2 > 0 \quad \forall n \in N\ n \geq n_0,$

$f(n) \leq c_1 \cdot g(n)$ and $g(n) \leq c_2 \cdot f(n)$.

Example:

$n = \Theta(n + \log n)$.

$c_1 = ?,\ c_2 = ?\ n_0 = ?$ such that $\forall n \geq n_0,$

$n \leq c_1(n + \log n)$ and $n + \log n \leq c_2 n$.

Theta

Definition:

$f(n) = \Theta(g(n))$ means

$\exists\, n_0, c_1, c_2 > 0 \quad \forall n \in N\ n \geq n_0,$

$f(n) \leq c_1 \cdot g(n)$ and $g(n) \leq c_2 \cdot f(n)$.

Example:

$n = \Theta(n + \log n)$.

$c_1 = 1$, $c_2 = 2$ $n_0 = 2$ such that $\forall n \geq 2$,

$n \leq 1\,(n + \log n)$ and $n + \log n \leq 2\,n$.

Theta

Definition:

$f(n) = \Theta(g(n))$ means

$\exists\, n_0, c_1, c_2 > 0 \quad \forall n \in N\ n \geq n_0,$

$f(n) \leq c_1 \cdot g(n)$ and $g(n) \leq c_2 \cdot f(n)$.

Note:

$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$

$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.

Mixing things up

We also write things like $n + O(\log n) = O(n)$

This means $\forall\ c\ \ \exists\ c', n_0 : \forall\ n > n_0\ \ n + c \log n < c' n$

Similarly, $n^3 \log(n) = n^{O(1)}$

$1/\ \Omega(\log n) = O(1/\log n)$

$2^n + n^{O(1)} = \Theta(2^n)$

...

Algorithms

Sorting problem:

- Input:

  A sequence (or array) of n numbers (a[1], a[2], …, a[n]).

- Desired output:

  A sequence (b[1], b[2], …, b[n]) of sorted numbers (in increasing order).

Example:

Input = (5, 17, -9, 76, 87, -57, 0).

Output = ?

Sorting problem:

- Input:

  A sequence (or array) of n numbers (a[1], a[2], …, a[n]).

- Desired output:

  A sequence (b[1], b[2], …, b[n]) of sorted numbers (in increasing order).

Example:

Input = (5, 17, -9, 76, 87, -57, 0).

Output = (-57, -9, 0, 5, 17, 76, 87).

Bubble sort:

Input (a[1], a[2], …, a[n]).

```
 for (i=n-1; i > 0; i--)
 {
   for (j=0; j < i; j++)
   {
     if (a[j] > a[j+1])
     swap a[j] and a[j+1];
    }
 }
```

Bubble sort

DEMO

Analysis of running time

T(n) = number of comparisons

i = n-1 ⇨ n -1 comparisons.

i = n-2 ⇨ n -2 comparisons.

…

i = 1 ⇨ 1 comparison.

```
Bubble sort:
Input (a[1], a[2], …, a[n]).
 for (i=n-1; i > 0; i--)
 {
    for (j=0; j < i; j++)
    {
      if (a[j] > a[j+1])
      swap a[j] and a[j+1];
     }
 }
```

$T(n) = (n-1) + (n-2) + \ldots + 1 < n^2$

Is this tight? Is also $T(n) = \Omega(n^2)$?

Analysis of running time

T(n) = number of comparisons

i = n-1 ⇨ n -1 comparisons.

i = n-2 ⇨ n -2 comparisons.

…

i = 1 ⇨ 1 comparison.

```
Bubble sort:
Input (a[1], a[2], …, a[n]).
 for (i=n-1; i > 0; i--)
 {
    for (j=0; j < i; j++)
    {
      if (a[j] > a[j+1])
      swap a[j] and a[j+1];
     }
 }
```

$T(n) = (n-1) + (n-2) + \ldots + 1 = n(n-1)/2 = \Theta(n^2)$

Space (also known as Memory)

We need to keep track of i, j

We need an extra element

to swap values of input array a.

Space = O(1)

```
Bubble sort:
Input (a[1], a[2], …, a[n]).
 for (i=n-1; i > 0; i--)
 {
    for (j=0; j < i; j++)
    {
      if (a[j] > a[j+1])
      swap a[j] and a[j+1];
     }
 }
```

Bubble sort takes quadratic time

Can we sort faster?

We now see two methods that can sort in linear time,

assuming that the numbers to sort are small.

Counting sort:

Assumption: all elements of the input array are integers in the range 0 to k.

The basic idea is to determine, for each x, the number of elements in the input array that are smaller than x.

This way we can put element x directly into its position.

Counting sort:

```
Countingsort (A[1..n])
   for (i=0; k ; i++)   C[i] = 0;  //Initializes C to 0

   for (j=1; n ; j++)  C[A[j]] =C[A[j]]+1;
       // C[i] =number of elements equal to i.

   for (i=1; k ; i++)  C[i] = C[i]+C[i-1] ;
          //C[i] = number of elements ≤ i.

   for (j=n; 1 ; j - -) {
         B[ C[ A[ j ] ] ] = A[ j ] ;  //Place A[j] at right location
         C[ A[ j ] ] = C[ A[ j ] ]-1; //Decrease for equal elements
   }
```

Analysis of running time

T(n) = number of operations

= O(k) + O(n) + O(k) + O(n)

= Θ(n + k).

If k = O(n) then T(n) = Θ(n)

```
Countingsort (A[1..n])
 for (i =0; i<k ; i++)
     C[i] = 0;
 for (j =1; j<n ; j++)
     C[A[j]] =C[A[j]] +1;
for (i =1; i<k ; i++)
     C[i] = C[i] +C[i-1] ;
for (j =n; j>1 ; j--)
     {
       B[ C[ A[ j ] ] ] = A[ j ] ;
       C[ A[ j ] ] = C[ A[ j ] ] -1;
      }
 }
```

Space

O(k) for C

Recall numbers in 0..k.

O(n) for B, where output is

Total space: O(n + k)

If k = O(n) then Θ(n)

```
Countingsort (A[1..n])
 for (i =0; i<k ; i++)
     C[i] = 0;
 for (j =1; j<n ; j++)
     C[A[j]] =C[A[j]] +1;
for (i =1; i<k ; i++)
     C[i] = C[i] +C[i-1] ;
for (j =n; j>1 ; j--)
     {
      B[ C[ A[ j ] ] ] = A[ j ] ;
      C[ A[ j ] ] = C[ A[ j ] ] -1;
     }
 }
```

Radix sort

Assumption: all elements of the input array are d-digit integers.

Idea: first sort the input array according to least significant digit of elements, then according to the next digit, … and finally according to the most significant digit.

It is essential to use a digit sorting algorithm that is stable: numbers with the same value appear in the output array in the same order as in the input array.

Fact: Counting sort is stable.

Radix sort

```
Radixsort(A[1..n]) {
  for (i =0; i<d ; i++)
    use a stable sorting algorithm to sort array A on digit i
}
```

We use Counting sort .

Radix sort

Radix sort DEMO.

Note: first sort by least significant bit, i.e.,
puts even number first, and then odd
(odd numbers are placed in temporary array)

Analysis of running time

```
Radixsort(A[1..n])  {
 for (i =0; i<d ; i++)
   use Counting sort to
   sort array A on digit i
 }
```

T(n) = number of operations

T(n) = d.(running time of Counting sort on n elements)

= Θ(d.n)

Example: To sort numbers in range 0.. n^{10}

T(n) = ?

(hint: think numbers in base n)

Analysis of running time

```
Radixsort(A[1..n])  {
 for (i =0; i<d ; i++)
   use Counting sort to
   sort array A on digit i
 }
```

T(n) = number of operations

T(n) = d.(running time of Counting sort on n elements)

= Θ(d.n)

Example: To sort numbers in range 0.. n^{10}

T(n) = Θ(10 n) = Θ(n)

While counting sort would take T(n) = ?

Analysis of running time

```
Radixsort(A[1..n])  {
 for (i =0; i<d ; i++)
   use Counting sort to
   sort array A on digit i
 }
```

T(n) = number of operations

T(n) = d.(running time of Counting sort on n elements)

= Θ(d.n)

Example: To sort numbers in range 0.. n^{10}

T(n) = Θ(10 n) = Θ(n)

While counting sort would take T(n) = Θ(n^{10})

Space

We need as much space as we did for Counting sort on each digit

```
Radixsort(A[1..n])
{
 for (i =0; i<d ; i++)
 use Counting sort to
 sort array A on digit i
 }
```

Can we sort faster than n^2 without extra assumptions?

Next we show how to sort in time O(n log (n)) arbitrary numbers

We introduce a new general paradigm