Chapter 1: Part II: Asymptotic Notations

Devesh C Jinwala , IIT Jammu, India

August 7, 2019

Design and Analysis of Algorithms
IIT Jammu, Jammu

The Big-Oh Notation

- def: for a given function g(n), we say that O(g(n)) = f(n) — if there exists positive constants c and n_0 such that, $0 \leq f(n) \leq cg(n)$, for all $n \geq n_0$

$f(n) = O(g(n)) \Rightarrow$

f(n)is dominated in the growth by g(n) i.e. f(n) is of the order at the most g(n) i.e. g(n) grows at least as fast as f(n)

The Big-Oh Notation

- def: for a given function g(n), we say that O(g(n)) = f(n) — if there exists positive constants c and n_0 such that, $0 \leq f(n) \leq cg(n)$, for all $n \geq n_0$
- The Big-oh defines an upper bound for a function within a constant factor i.e. except for a constant factor and a finite number of exceptions, f is bounded above by g.

$f(n) = O(g(n)) \Rightarrow$

f(n)is dominated in the growth by g(n) i.e. f(n) is of the order at the most g(n) i.e. g(n) grows at least as fast as f(n)

The Big-Oh Notation

- Can $f(n)$ grow faster than $g(n)$?

The Big-Oh Notation

- Can $f(n)$ grow faster than $g(n)$?
- Can $g(n)$ grow faster than $f(n)$?

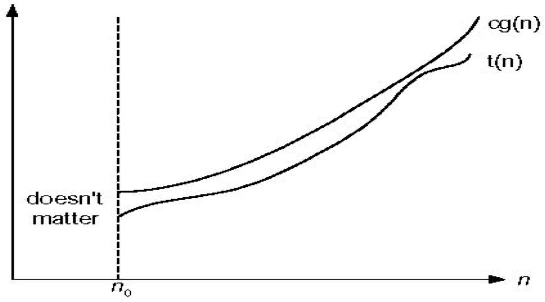


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

The Big-Oh Notation

- Can $f(n)$ grow faster than $g(n)$?
- Can $g(n)$ grow faster than $f(n)$?

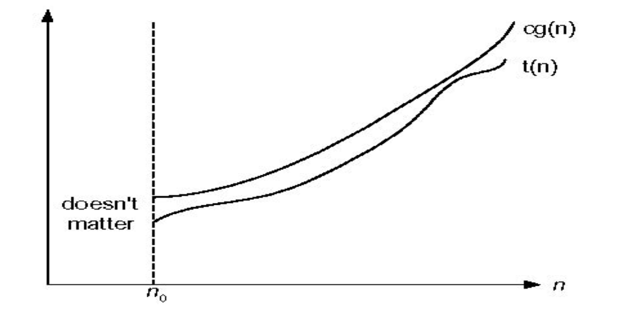


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

- What does the growth rate imply ?

The Big-Oh Notation Illustrations

Function	notation in O
$f(n) = 5n + 8$	$f(n) = O(?)$
$f(n) = n^2 + 3n - 8$	$f(n) = O(?)$
$F(n) = 12n^2 - 11$	$f(n) = O(?)$
$F(n) = 5*2^n + n^2$	$f(n) = O(?)$
$f(n) = 3n + 8$	$F(n) = O(n^2)$?
$f(n) = 5n + 8$	$f(n) = O(1)$?

The Big-Oh Notation

- allows us to keep track of the leading term while ignoring smaller terms

The Big-Oh Notation

- allows us to keep track of the leading term while ignoring smaller terms
- allows us to make concise statements that give approximations to the quantities to analyze.

The Big-Oh Notation

- allows us to keep track of the leading term while ignoring smaller terms

The Big-Oh Notation

- allows us to keep track of the leading term while ignoring smaller terms
- allows us to make concise statements that give approximations to the quantities to analyze.

Tutorial Problem No 1

- if $f(n) = O(g(n))$, what is the upper bound ?

Tutorial Problem No 1

- if f(n)= O(g(n)), what is the upper bound ?
- Do we specify how tight this upper bound is?

Tutorial Problem No 1

- if f(n)= O(g(n)), what is the upper bound ?
- Do we specify how tight this upper bound is?
- Consider that f(n)= O(n) & g(n)= O(n^2). Is f(n)= O(g(n)) saying the same as reverse i.e. g(n) = O(f(n))?

Tutorial Problem No 1

- if f(n)= O(g(n)), what is the upper bound ?
- Do we specify how tight this upper bound is?
- Consider that f(n)= O(n) & g(n)= O(n^2). Is f(n)= O(g(n)) saying the same as reverse i.e. g(n) = O(f(n))?
- The symbol = is not proper truly it is ϵ which should be used i.e. f(n) ϵ O(g(n)

The Big-Oh notation...

- When O notation bounds the worst case running time of an algorithm, by implication we also bound the running time of an algorithm on EVERY input.

Abuse

Technically, it is abuse to say that the running time of insertion sort is $O(n^2)$. Why?

The Big-Oh notation...

- When O notation bounds the worst case running time of an algorithm, by implication we also bound the running time of an algorithm on EVERY input.
- this is not so when using other notations i.e. the worst case $\theta(n^2)$ or θ(n) does not apply to every input.

Abuse

Technically, it is abuse to say that the running time of insertion sort is $O(n^2)$. Why?

The Big-Ω notation...

- the big-Ω notation is used to specify this aspect i.e. to define a lower bound for a function within a constant factor..

The Big-Ω notation...

- the big-Ω notation is used to specify this aspect i.e. to define a lower bound for a function within a constant factor..
- definition: the function f(n)= Ω (g(n)) is true iff there exists positive constants c and n_0 such that $f(n) >= c(g(n))$ for all n - n $\geq n_0$ i.e. $0 <= c(g(n)) <= f(n)$.

The Big-Ω notation...

- the big-Ω notation is used to specify this aspect i.e. to define a lower bound for a function within a constant factor..
- definition: the function f(n)= Ω (g(n)) is true iff there exists positive constants c and n_0 such that $f(n) >= c(g(n))$ for all n - n $\geq n_0$ i.e. $0 <= c(g(n)) <= f(n)$.
- i.e. except for a constant factor and a finite number of exceptions, f is bounded below by g.

The Big-Ω notation...

- the big-Ω notation is used to specify this aspect i.e. to define a lower bound for a function within a constant factor..
- definition: the function f(n)= Ω (g(n)) is true iff there exists positive constants c and n_0 such that $f(n) >= c(g(n))$ for all n - n $\geq$ n_0 i.e. $0 <= c(g(n)) <= f(n)$.
- i.e. except for a constant factor and a finite number of exceptions, f is bounded below by g.
- f(n) $= \Omega$(g(n)) implies that

The Big-Ω notation...

- the big-Ω notation is used to specify this aspect i.e. to define a lower bound for a function within a constant factor..
- definition: the function f(n)= Ω (g(n)) is true iff there exists positive constants c and n_0 such that $f(n) >= c(g(n))$ for all n - n $\geq$ n_0 i.e. $0 <= c(g(n)) <= f(n)$.
- i.e. except for a constant factor and a finite number of exceptions, f is bounded below by g.
- f(n) = Ω(g(n)) implies that

$\Omega(n) \Rightarrow$

f(n) always dominates the growth of g(n) i.e. f(n) is of the order at least g(n) i.e. g(n) grows at the most as fast as f(n).

The Big-Ω notation...

The Big Omega

- Can f(n) grow faster than g(n)?
- Can g(n) grow faster than f(n)?

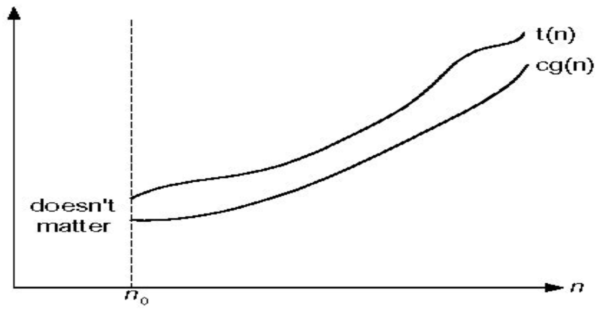


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

The Big-Oh Notation Illustrations

Function	notation in Ω
$f(n) = 3n + 8$	$f(n) = \Omega(?)$
$f(n) = n^2 + 3n - 8$	$f(n) = \Omega(?)$
$F(n) = 12n^2 - 11$	$f(n) = \Omega(?)$
$F(n) = 6*2^n + n^2$	$f(n) = \Omega(?)$
$f(n) = 3n + 8$	$f(n) = \Omega(n^2)$?
$f(n) = 5n + 8$	$f(n) = \Omega(1)$?

Tutorial Problem No 2

- Is it correct to say that $3n + 8 = \Omega(1)$?

Tutorial Problem No 2

- Is it correct to say that $3n + 8 = \Omega(1)$?
- There could be several functions g(n) for which f(n)= Ω (g(n)) - the function g(n) is only lower bound on n.

Tutorial Problem No 2

- Is it correct to say that $3n + 8 = \Omega(1)$?
- There could be several functions g(n) for which f(n)= Ω (g(n)) - the function g(n) is only lower bound on n.
- hence, for f(n)=Ω(g(n) to be meaningful, g(n) should be as large a function of n as possible.

Tutorial Problem No 2

- Is it correct to say that $3n + 8 = \Omega(1)$?
- There could be several functions g(n) for which f(n)= Ω (g(n)) - the function g(n) is only lower bound on n.
- hence, for f(n)=Ω(g(n) to be meaningful, g(n) should be as large a function of n as possible.
- Given two choices viz. 3n+3=Ω(n) and 3n+3=Ω(1), which one shall we choose ?

Tutorial Problem No 2

- Is it correct to say that $3n + 8 = \Omega(1)$?
- There could be several functions g(n) for which f(n)= Ω (g(n)) - the function g(n) is only lower bound on n.
- hence, for f(n)=Ω(g(n) to be meaningful, g(n) should be as large a function of n as possible.
- Given two choices viz. 3n+3=Ω(n) and 3n+3=Ω(1), which one shall we choose ?
- if f(n)= $O(\Omega(n))$, what is the lower bound ?

Tutorial Problem No 2

- Is it correct to say that $3n + 8 = \Omega(1)$?
- There could be several functions g(n) for which f(n)= Ω (g(n)) - the function g(n) is only lower bound on n.
- hence, for f(n)=Ω(g(n) to be meaningful, g(n) should be as large a function of n as possible.
- Given two choices viz. 3n+3=Ω(n) and 3n+3=Ω(1), which one shall we choose ?
- if f(n)= $O(\Omega(n))$, what is the lower bound ?
- Do we specify how tight this lower bound is?

Tutorial Problem No 2...

- When we say $f(n)=\Omega(g(n))$, does it mean that $g(n)=O(f(n))$?

Tutorial Problem No 2...

- When we say f(n)=Ω(g(n)), does it mean that g(n)=O(f(n))?
- Would the growth of f(n) dominate the growth of (g(n)) ?

Tutorial Problem No 2...

- When we say f(n)=Ω(g(n)), does it mean that g(n)=O(f(n))?
- Would the growth of f(n) dominate the growth of (g(n)) ?
- Consider f(n)=n^2 & g(n)=2n^2

Tutorial Problem No 2...

- When we say f(n)=Ω(g(n)), does it mean that g(n)=O(f(n))?
- Would the growth of f(n) dominate the growth of (g(n)) ?
- Consider f(n)=n^2 & g(n)=2n^2

When we use Ω-notation to describe a lower bound,

Tutorial Problem No 2...

- When we say f(n)=Ω(g(n)), does it mean that g(n)=O(f(n))?
- Would the growth of f(n) dominate the growth of (g(n)) ?
- Consider f(n)=n^2 & g(n)=2n^2

When we use Ω-notation to describe a lower bound,

- we are also implying a lower bound on the running time of the algorithm on arbitrary inputs as well.

Tutorial Problem No 2...

- When we say f(n)=Ω(g(n)), does it mean that g(n)=O(f(n))?
- Would the growth of f(n) dominate the growth of (g(n)) ?
- Consider f(n)=n^2 & g(n)=2n^2

When we use Ω-notation to describe a lower bound,

- we are also implying a lower bound on the running time of the algorithm on arbitrary inputs as well.
- the bounds $O(n^2)$ and Ω(n) are as tight bounds as possible.

Tutorial Problem No 2...

- When we say f(n)=Ω(g(n)), does it mean that g(n)=O(f(n))?
- Would the growth of f(n) dominate the growth of (g(n)) ?
- Consider f(n)=n^2 & g(n)=2n^2

When we use Ω-notation to describe a lower bound,

- we are also implying a lower bound on the running time of the algorithm on arbitrary inputs as well.
- the bounds $O(n^2)$ and Ω(n) are as tight bounds as possible.
- Can we say that the running time of insertion sort is $\Omega(n^2)$?

Tutorial Problem No 2...

- When we say f(n)=Ω(g(n)), does it mean that g(n)=O(f(n))?
- Would the growth of f(n) dominate the growth of (g(n)) ?
- Consider f(n)=n^2 & g(n)=2n^2

When we use Ω-notation to describe a lower bound,

- we are also implying a lower bound on the running time of the algorithm on arbitrary inputs as well.
- the bounds $O(n^2)$ and Ω(n) are as tight bounds as possible.
- Can we say that the running time of insertion sort is $\Omega(n^2)$?
- Can we say that the running time of insertion sort is O(n)?

The Big-Θ notation...

- Neither the big-O notation nor the big-Ω notation describe the asymptotically tight bounds.
- Θ-notation to express tighter bounds - used to specify the exact order of growth of functions.
- def: we say that f(n)=Θ((g)n) iff there exists positive constants c_1 and c_2 and a number n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$
- f(n)=Θ((g)n) iff f(n)=O(g(n)) and f(n)=Ω(g(n))

The Big-Θ notation...

- Neither the big-O notation nor the big-Ω notation describe the asymptotically tight bounds.
- Θ-notation to express tighter bounds - used to specify the exact order of growth of functions.
- def: we say that f(n)=Θ((g)n) iff there exists positive constants c_1 and c_2 and a number n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$
- f(n)=Θ((g)n) iff f(n)=O(g(n)) and f(n)=Ω(g(n))

The Big-Θ Notation...

Given f(n)=Θg(n)

- Can $f(n)$ grow faster than $g(n)$?

The Big-Θ Notation...

Given f(n)=Θg(n)

- Can $f(n)$ grow faster than $g(n)$?
- Can $g(n)$ grow faster than $f(n)$?

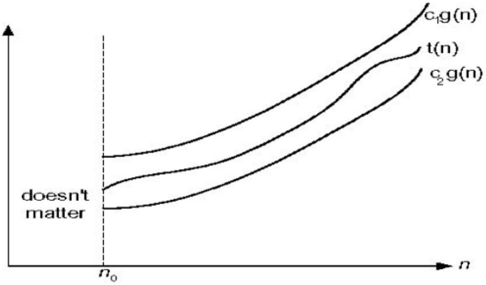


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

The Big-Θ Notation...

Given f(n)=Θg(n)

- Can $f(n)$ grow faster than $g(n)$?
- Can $g(n)$ grow faster than $f(n)$?

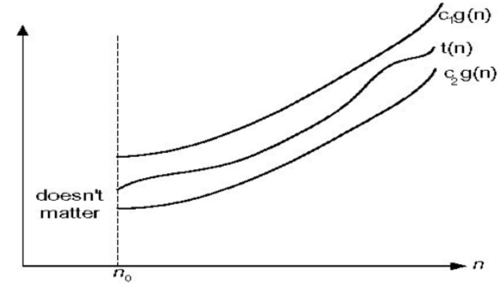


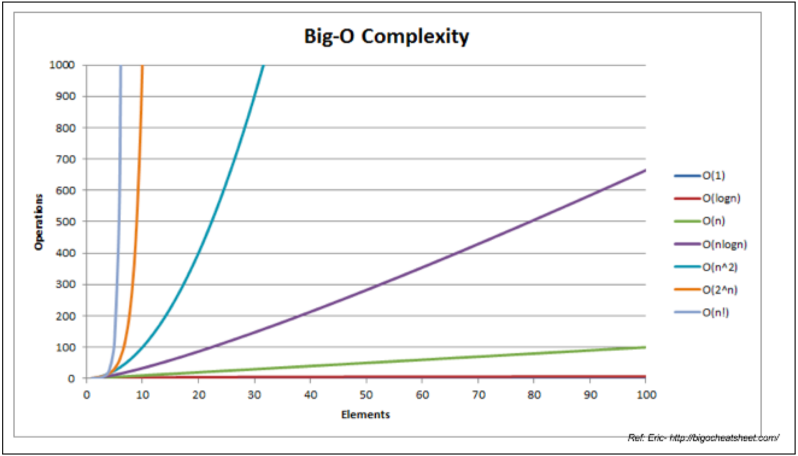
Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

- What does the growth rate imply ?

The Big-Θ Notation...

Function	notation in θ
$f(n) = 3n + 8$	$f(n) = \theta(?)$
$f(n) = 10n^2 + 3n - 8$	$f(n) = \theta(?)$
$F(n) = 12n^2 - 11$	$f(n) = \theta(?)$
$F(n) = 6*2^n + n^2$	$f(n) = \theta(2^n)$?
$F(n) = 6*2^n + n^2$	$f(n) = \theta(n^2)$?
$f(n) = 3n + 8$	$f(n) = \theta(n^2)$?
$f(n) = 5n + 8$	$f(n) = \theta(1)$?

The Asymptotic Classes



Complexity of Data Structures

Ref: Eric- http://bigocheatsheet.co

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

Complexities of Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

Ref: Eric- http://bigocheatsheet.com/

Lower Bound on Sorting

- To sort n elements, one needs information about the sequence of these elements
  i.e. given two elements, the relative order could be

Lower Bound on Sorting

- To sort n elements, one needs information about the sequence of these elements
  i.e. given two elements, the relative order could be
- Decision tree at a higher level of abstraction

Lower Bound on Sorting

- To sort n elements, one needs information about the sequence of these elements
  i.e. given two elements, the relative order could be
- Decision tree at a higher level of abstraction
- For an input sequence of three elements, what would be the total number of 2-element comparisons ?

Lower Bound on Sorting

- To sort n elements, one needs information about the sequence of these elements
  i.e. given two elements, the relative order could be
- Decision tree at a higher level of abstraction
- For an input sequence of three elements, what would be the total number of 2-element comparisons ?
- Draw a decision tree showing the number of comparisons of any n-distinct elements - say for n $=3$.

Lower Bound on Sorting

- To sort n elements, one needs information about the sequence of these elements
  i.e. given two elements, the relative order could be
- Decision tree at a higher level of abstraction
- For an input sequence of three elements, what would be the total number of 2-element comparisons ?
- Draw a decision tree showing the number of comparisons of any n-distinct elements - say for n =3.
  - represent each internal node by a_i:a_j in the range $1 \leq i \leq n$

Lower Bound on Sorting

- To sort n elements, one needs information about the sequence of these elements
  i.e. given two elements, the relative order could be
- Decision tree at a higher level of abstraction
- For an input sequence of three elements, what would be the total number of 2-element comparisons ?
- Draw a decision tree showing the number of comparisons of any n-distinct elements - say for n =3.
  - represent each internal node by a_i:a_j in the range $1 \leq i \leq n$
  - denote each leaf by permutation ($\pi(1)$, $\pi(2)$, $\pi(3)$, $\pi(4)$.. $\pi(n)$)

Lower Bound on Sorting

- How does the execution of a sorting algorithm relate with this decision tree?

Lower Bound on Sorting

- How does the execution of a sorting algorithm relate with this decision tree?
- What does the length of the longest path from the root of the tree to any of its leave represent ?

Lower Bound on Sorting

- How does the execution of a sorting algorithm relate with this decision tree?
- What does the length of the longest path from the root of the tree to any of its leave represent ?
- How does it relate to the minimum number of comparisons required under the decision tree model ?

Lower Bound on Sorting

- How does the execution of a sorting algorithm relate with this decision tree?
- What does the length of the longest path from the root of the tree to any of its leave represent ?
- How does it relate to the minimum number of comparisons required under the decision tree model ?
- Can we find out a lower bound on the height of the decision tree under consideration ?

Lower Bound on Sorting

- How does the execution of a sorting algorithm relate with this decision tree?
- What does the length of the longest path from the root of the tree to any of its leave represent ?
- How does it relate to the minimum number of comparisons required under the decision tree model ?
- Can we find out a lower bound on the height of the decision tree under consideration ?
- How does it relate to the running time of any comparison based sort ?

Lower Bound on Sorting...

Theorem

Any decision tree that sorts n-elements has height $\Omega(nlgn)$

Corollary

Heapsort and Mergesort are asymptotically optimal comparison sorts.

The Counting Sort

- Let input instance A=<7 1 3 1 2 4 5 7 2 4 3>

A

1	2	3	4	5	6	7	8	9	10	11
7	1	3	1	2	4	5	7	2	4	3

- `C[A[j]] = C[A[j]] + 1`

C

1	2	3	4	5	6	7
2	2	2	2	1	0	2

- `C[i] = C[i] + C[i-1]`

1	2	3	4	5	6	7
2	4	6	8	9	0	11

- `B[C[A[j]]] = A[j]`

The Counting Sort ...

Algorithm CountingSort(A[i],n)

```
for i=1 to k
        do C[j] =0
for j = 1 to length[A]
        do C[A[j]] = C[A[j]] + 1
for i=2 to k
        do C[i] = C[i] + C[i-1]
for j=length[A] downto 1
        do B[C[A[j]]] = A[j]
                C[A[j]] = C[A[j]] - 1
```

What is the time complexity of this sort ?

The Interger Sorting Algorithms

- Integer sorting
  - sorting a collection of data values by numeric keys, each of which is an integer.
  - the ability to perform integer arithmetic on the keys allows integer sorting algorithms to be faster than comparison sorting algorithms in many cases, depending on the details of which operations are allowed in the model of computing and how large the integers to be sorted are.
  - the classical integer sorting algorithms of bucket sort, counting sort, and radix sort

Blank

Blank

Blank