Asymptotic Notations II

RM.Periakaruppan

Associate Professor

Dept. of Applied Mathematics and Computational Sciences

PSG College of Technology

Asymptotic Notations Example 3

- Find upper bound for
- f(n) = 2 n^2 + 3 n + 4

Asymptotic Notations Example 3

- Find upper bound for
- f(n) = 2 n^2 + 3 n + 4

2 n^2 + 3 n + 4 <= 2 n^2 + 3 n^2 + 4 n^2

2 n^2 + 3 n + 4 <= 9 n^2 , n>=1

f(n) = O(n^2)

Asymptotic Notations Example 3

- Find lower bound for
- f(n) = 2 n^2 + 3 n + 4

Asymptotic Notations Example 3

- Find lower bound for
- f(n) = 2 n^2 + 3 n + 4

2 n^2 + 3 n + 4 >= 1 . n^2

2 n^2 + 3 n + 4 >= n^2 , n>=1

f(n) = Ω(n^2)

Asymptotic Notations Example 3

Find theta (tight) bound for

- f(n) = 2 n^2 + 3 n + 4

Asymptotic Notations Example 3

- Find theta bound for
- f(n) = 2 n^2 + 3 n + 4

n^2 <= 2 n^2 + 3 n + 4 <=9 n^2

f(n)= $\Theta(n^2)$

Asymptotic Notations Example 4

- Find upper bound for
- f(n) = n^2 log n + n

Asymptotic Notations Example 4

- Find upper bound for
- f(n) = n^2 log n + n

- n^2 log n + n <= n^2 log n + n^2 log n

  n^2 log n + n <= 2 n^2 log n , for all n >= 2

  f(n) = O(n^2 log n)

Asymptotic Notations Example 4

- Find lower bound for
- f(n) = n^2 log n + n

Asymptotic Notations Example 4

- Find lower bound for
- f(n) = n^2 log n + n

- n^2 log n + n >= n^2 log n for all n >= 1

f(n) = Ω(n^2 log n)

Asymptotic Notations Example 4

- Find theta bound for
- f(n) = n^2 log n + n

Asymptotic Notations Example 4

- Find theta bound for
- f(n) = n^2 log n + n

- n^2 log n <= n^2 log n + n <= 2 n^2 log n

  f(n) = Θ(n^2 log n)

Other asymptotic notations

- o notation (small Oh or little Oh)

We use o-notation to denote an upper bound that is not asymptotically tight.

We formally define o(g(n)) (“little-oh of g of n”) as the set

o(g(n))={f(n) : for any positive constant c>0, there exists a constant

n0 > 0 such that 0 <=f (n) < c g(n) for all n >= n0}

For example, 2n = o(n^2), but 2n^2 # o(n^2)

Other asymptotic notations

- ω notation (small Omega or little Omega)

We use o-notation to denote an lower bound that is not asymptotically tight.

We formally define ω(g(n)) (“little-omega of g of n”) as the set

o(g(n))={f(n) : for any positive constant c>0, there exists a constant

n0 > 0 such that 0 <= cg(n)<f(n) for all n >= n0}

For example, n^2/2 = ω(n), but n^2 /2 # ω(n^2)

Comparing functions

For the following, assume that f (n) and g(n) are asymptotically positive.

Transitivity:

f (n)= O(g(n)) and g(n) = O(h(n)) imply f (n) = O(h(n))

Similarly for Ω , Θ, o and ω notations.

Comparing functions

For the following, assume that f (n) and g(n) are asymptotically positive.

Reflexivity:

If f(n) is given then

f (n)= O(f(n)

For example if f(n)=n^3 then f(n)=O(n^3)

Similarly for Ω and Θ

Comparing functions

For the following, assume that f (n) and g(n) are asymptotically positive.

Symmetry

f (n)= Θ(g(n)) iff g(n)=Θ(f(n))

For example

If f(n) = n^2 and g(n) = n^2 then f(n) = Θ(n^2) and g(n) = Θ(n^2)

Comparing functions

For the following, assume that f (n) and g(n) are asymptotically positive.

Transpose Symmetry

f (n)= O(g(n)) iff g(n)=Ω(f(n))

f(n)= o(g(n)) iff g(n) = ω(f(n))

Analysis of Recursive function

- A function which call itself is called recursive function.
- Terminates when a base case is reached
- Each recursive call uses stack memory
- Used in solving problems based on divide and conquer strategy
- (eg) int fact(int n)
- { if (n==0 || n ==1) return 1; //base case
- else
- return n*fact(n-1); // recursive call to function fact
- }
- Mathematically can be expressed as recurrence relation
- T(n) = T(n-1) + a , n>1

  = 1 , n=0,1

Analysis of Recursive function

- Solving the following Recurrence Relation
- $T(n) = T(n-1) + a \quad , \ n>1$

  $= 1 \quad , \ n=0,1$

$T(n) = T(n-1) + a$

$= [T(n-2)+a] + a$

$= T(n-2)+2a$

$= T(n-3)+3a$

Analysis of Recursive function

•

$T(n) = T(n-k)+ka$ {k times}

When k=n

$T(n) = T(n-n)+na$

$= T(0) + na$

$= 1 + a\ n$

$= O(n)$

Worst, best and average case time complexity

- Based on the input instances of the problem, the best, worst and average case time complexities can be found.
- The input instances for which the algorithm takes maximum possible time is called worst case.
- The input instances for which the algorithm takes minimum possible time is called best case.
- The input instances for which the algorithm is neither behaves as best case nor worst case is called average case

Worst, best and average case time complexity Example Linear Search

- Worst Case happens when the element to be searched is not present or the last element in the array. Hence worst case time complexity is O(n) [or Θ(n) or Ω(n)].

- Best case happens when the element to be searched when the element to be searched is found in first position. Hence worst case time complexity is O(1) [or Θ(1) or Ω(1)].

-

Worst, best and average case time complexity Example Linear Search

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (n+1). Following is the value of average case time complexity.

Average case = (1+2+3+...n)/n

= [n(n+1)/2]/n

= n+1/2

= O(n) [or Θ(n) or Ω(n)]

•

Time complexities of all sorting algorithms

Algorithm	Data Structure	Time Complexity		
		Best	Average	Worst
Quicksort	Array	O(n log(n))	O(n log(n))	O(n^2)
Mergesort	Array	O(n log(n))	O(n log(n))	O(n log(n))
Heapsort	Array	O(n log(n))	O(n log(n))	O(n log(n))
Bubble Sort	Array	O(n)	O(n^2)	O(n^2)
Insertion Sort	Array	O(n)	O(n^2)	O(n^2)
Select Sort	Array	O(n^2)	O(n^2)	O(n^2)
Bucket Sort	Array	O(n+k)	O(n+k)	O(n^2)
Radix Sort	Array	O(nk)	O(nk)	O(nk)

Sample Questions

1. Suppose we have a O(n) time algorithm that finds median of an unsorted array. Now consider a QuickSort implementation where we first find median using the above algorithm, then use median as pivot. What will be the worst case time complexity of this modified QuickSort.
2. Which of the following sorting algorithms has the lowest worst-case complexity?**(A)** Merge Sort **(B)** Bubble Sort **(C)** Quick Sort
   (D) Selection Sort
3. Let s be a sorted array of n integers. Let t(n) denote the time taken for the most efficient algorithm to determined if there are two elements with sum less than 1000 in s. What is t(n)?