Complexity

Asymptotic behavior :→

Complexity analysis is also a tool that allows us to explain how an algorithm behaves as the input grows larger. If we feed it a different input, how will the algorithm behave? If our algorithm takes 1 second to run for an input of size 1000, how will it behave if I double the input size? Will it run just as fast, half as fast or four times slower?

e.g. If we have $6n+4$ instructions to be executed in the program [where n is input] then

Asymptotic $O(n) = n$ [we ignore all constant terms]

if $f(n) = 109$ instructions (constant)

then $O(n) = 1$

It means by increasing or decreasing inputs won't impact Algorithm behaviour.

if $f(n) = n^n + n$

as $n^n >>> n$

$\therefore O(n) = n^n$

$f(n) = n^6 + 3n$ then $O(n) = n^6$

(1) If any program does not have any loop then $O(n) = 1$

(2) A single loop over n items yields $O(n) = n$

(3) A loop within a loop yields $O(n) = n^2$

(2)

(4) A loop within a loop within a loop yields $f(n)=n^3$.

(5) Given a series of for loops that are sequential, the slowest of them determines the asymptotic behavior of program. Two nested loops followed by a single loop is asymptotically the as the nested loop alone, because the nested loops dominate the simple loop.

1. $n^6+3n \in \Theta(n^6)$
2. $2^n+12 \in \Theta(2^n)$
3. $3^n+2^n \in \Theta(3^n)$
4. $n^n+n \in \Theta(n^n)$

$\Theta(1)$: Constant time algo.

$\Theta(n)$: linear

$\Theta(n^2)$: quadratic

$\Theta(\log(n))$: logarithmic

(6) Programs with a bigger Θ run slower than programs with a smaller Θ.

From the best to worst the rankings are.

$O(1)$, $O(\log n)$, $O(n)$, $O(n\log n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$

June 15 (III) (3.5)

Let $f(n)$ and $g(n)$ be asymptotically non-negative functions. ~~Which of the following is correct~~.

then $\Theta(f(n)+g(n)) = \max(f(n), g(n))$

Big-O notation :→ eg. $O(n^2)$ It means that our program is asymptotically no worse than n^2. It may be better than that or it may be the same as that. If our program is indeed $\Theta(n^2)$, we can say that it is $O(n^2)$. This gives us a good estimate of how fast our program runs.

$\Theta(n)$ basically gives worst case behavior of current algorithm.

$O(n)$ gives worst case behavior of current algo. and even we can make more changes in the algo. to make it worse.

So $O(n)$ can be worst than $\Theta(n)$ but can not be better than that as $O(n)$ is upper bound.

1. $\Theta(n)$ algo is $O(n)$

Sol:- True. Don't alter the program.

2. $\Theta(n)$ algo is $O(n^2)$

Sol:- True. n^2 is worse than n

3. $\Theta(n^2)$ algo is $O(n^3)$

Sol:- True. n^3 is worse than n^2

4. $\Theta(n)$ algo is $O(1)$

False. 1 is not worse than n.

5. $O(1)$ algo is $\Theta(1)$

True

6. $O(n)$ algo is $\Theta(1)$ Sol:- May or may not be true depending on algo. In general, false

Data Structure Operations

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)

Array 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)

Graph Operations

Node / Edge Management	Storage	Add Vertex	Add Edge	Remove Vertex	Remove Edge	Query
Adjacency list	$O(\lvert V\rvert+\lvert E\rvert)$	O(1)	O(1)	$O(\lvert V\rvert + \lvert E\rvert)$	$O(\lvert E\rvert)$	$O(\lvert V\rvert)$
Incidence list	$O(\lvert V\rvert+\lvert E\rvert)$	O(1)	O(1)	$O(\lvert E\rvert)$	$O(\lvert E\rvert)$	$O(\lvert E\rvert)$
Adjacency matrix	$O(\lvert V\rvert^2)$	$O(\lvert V\rvert^2)$	O(1)	$O(\lvert V\rvert^2)$	O(1)	O(1)
Incidence matrix	$O(\lvert V\rvert \cdot \lvert E\rvert)$	$O(\lvert V\rvert \cdot \lvert E\rvert)$	$O(\lvert V\rvert \cdot \lvert E\rvert)$	$O(\lvert V\rvert \cdot \lvert E\rvert)$	$O(\lvert V\rvert \cdot \lvert E\rvert)$	$O(\lvert E\rvert)$

Heap Operations

Type	Time Complexity						
	Heapify	Find Max	Extract Max	Increase Key	Insert	Delete	Merge
Linked List (sorted)	-	O(1)	O(1)	O(n)	O(n)	O(1)	O(m+n)
Linked List (unsorted)	-	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
Binary Heap	O(n)	O(1)	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(m+n)
Binomial Heap	-	O(1)	O(log(n))	O(log(n))	O(1)	O(log(n))	O(log(n))
Fibonacci Heap	-	O(1)	O(log(n))	O(1)	O(1)	O(log(n))	O(1)

Bucket Sort :-> n : no. of keys.
k : range of each key (0 – k-1)

June 15 (III)

5

33) Which of the following is asymptotically smaller?

1) $\lg(\lg^* n)$ 2) $\lg^*(\log n)$ 3) $\log(\underline{Ln})$ 4) $\log^*(\underline{Ln})$

Sol ⇒ $\lg^* n$ (Inverse Ackermann function) is the number of times we can ~~take~~ log repeatedly until we get n inbetween $[0,1]$. This function can almost be considered constant for all practical purposes.

Option 1: lg (Constant)

Option 2 : Constant $[\underline{\lg^*(\log n)}]$

Option 3 : $\Theta(\log(\underline{Ln})) = \log(n^n) = n\log n$

Option 4 : $\Theta(\log^*(\underline{Ln}))$ is constant.

Option 3 is asymptotically biggest.
Option 2 & 4 is constant and option 1 is log (constant)

So, option 1 is asymptotically smaller.

Master Theorem :→

$$T(n) = a\,T(n/b) + f(n)$$

Case 1 : $T(n) = \cancel{\Theta(n\log}\ \Theta(n^{\log_b a})$

if $f(n) = O(n^{\log_b a - e})$

asymptotic upper bound

Case 2 : $T(n) = \Theta(n^{\log_b a} \lg^{k+1}(n))$

if $f(n) = \Theta(n^{\log_b a} \log^k n)$ with $k \geq 0$

Case 3 : $T(n) = \Theta(f(n))$

if $f(n) = \Omega(n^{\log_b a + e})$

asymptotic lower bound (big Omega)

where e is a +ve constant

$a \geq 1$ and $b > 1$ are constants and $f(n)$ is a +ve function.

Examples :→

(1) $T(n) = 2T(n/2) + 1$

$a = 2 \quad b = 2 \quad f(n) = 1 = n^0$

$n^{\log_b a} = n^{\log_2 2} = n^1$

$n^{1-e} = n^0$

Put $e = 1$

$n^{1-1} = n^0$

∴ Case 1 satisfies

∴ $T(n) = \Theta(n)$ Ans

7

(2) $T(n) = 2T(n/2) + n$

$a = 2 \quad b = 2 \quad f(n) = n^1$

$n^{\log_b a} = n^{\log_2 2} = n^1$

As $f(n) = n^{\log_b a}$

$\therefore$ Case 2 applies.

$\cancel{\Theta(n^{\log_b a} \log f(n))} = \Theta(n^{\log_b a} \log n)$

$= \Theta(n \log n)$

(3) $T(n) = 4T(n/2) + \sqrt{n}$

$a = 4 \quad b = 2 \quad f(n) = n^{1/2}$

$n^{\log_b a} = n^{\log_2 4} = n^{\log_2 2^2} = n^{2\log_2 2} = n^2$

$e = 1.5$ and $n^{\log_b a} > f(n)$

$\therefore T(n) = \Theta(n^2)$

(4) $T(n) = 2T(n/2) + n^2$

$a = 2 \quad b = 2 \quad f(n) = n^2$

$n^{\log_b a} = n^1$

$e = 1$ and $f(n) > n^{\log_b a}$

$\therefore \Theta(n^2)$

(8) (8)

(5) $T(n) = 2T(n/2) + n\log n$

$a = 2 \quad b = 2 \qquad f(n) = n\log n$

$n^{\log_b a} = n$

$f(n) = n^{\log_b a} \log n \qquad$ for $k = 1$

$\therefore$ Case 2 applies

$T(n) = \Theta(n\log^2(n))$ Ans

(6) $T(n) = 16T(n/4) + \lfloor n$

$a = 16 \quad b = 4$

$n^{\log_b a} = n^{\log_4 16} = n^{\log_4 4^2} = n^{2\log_4 4} = n^2$

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

$\epsilon = n-2$

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

$= \Theta(\lfloor n)$

$\lfloor n =$
$n\cdot(n-1)\cdot(n-2) \dots 1$
$= n^n$

(7) $T(n) = 0.5T(n/2) + 1/n$

$a < 1 \therefore$ Does not apply.

(8) $T(n) = 2T(n/2) + n/\log n$

Non polynomial diff. between $f(n)$ and $n^{\log_b a}$

Does not apply.

9 $T(n) = T(\sqrt{n}) + 1$, $T(1) = 0$ 9 ~~4~~

$T(2) = T(\sqrt{2}) + 1$

$= T(1.44) + 1$

$= 0 + 1 = 1$

$\cancel{T(64)}$

/5

$T(256) + 1 \leftarrow T((256)^2) = T(2^{16})$

/4

$T(16) + 1 \leftarrow T(256)$

/3

$T(4) + 1 \leftarrow T(16)$

/2

$\underset{=1}{T(2)} + 1 \leftarrow T(4)$

$T(2^{16}) = 5 \qquad n = 2^{16}$

$O(n) = 2^{16}$ X $\quad O(\sqrt{n}) = 256$ X

$O(\log n) = O(\log 2^{16}) = 16$ X

$O(\log \log n) = O(\log \log 2^{16}) = O(\log 16)$

$= O(\log 2^4) = 4$

≈ 5

$\therefore O(\log \log n)$ Ans

(10) $T(n) = 2T(\sqrt{n}) + \log n$

Put $n=1$

$T(1) = 2T(1) + \log 1$

$T(1) = 0$

$T(2) = 2T(1) + \log 2$

$= 1$

80

$2T(\cancel{256}) + \log n \leftarrow T((256^2))\ [n = 2^{16}]$

16

/32

$2T(16) + \log n \leftarrow \cancel{\log}\ T(256)\ (n = 2^8)$

8

/12

$2T(4) + \log n \leftarrow T(16) \quad (n = 2^4)$

4

/4

$2T(2) + \log n \leftarrow T(4)\ (n = 2^2)$

1 2

$T(2^{16}) = 80 \quad$ where $= n = 2^{16}$

$\Theta(n \log\log n) = \Theta(n \log\log 2^{16}) = \Theta(n \log 16)$

$= \Theta(n \log 2^4)$

$= \Theta(n \times 4)$

$= 4 \times 2^{16}$ X

$\Theta(\sqrt{n}\ \underbrace{\log\log n}_{=4}) = 2^8 \times 4$ X

10 ~~8~~

$$O\left(\underbrace{\log n}_{=16}\left(\underbrace{\log \log n}_{=4}\right)\right) = 64 \simeq 80 \geq \quad 11 \quad \cancel{6}$$

$$O\left((\log n)^2\right) = (16)^2 = 256 \quad X$$

$$\underline{\underline{\text{Ans}}} \quad O(\log n (\log \log n))$$

Dec 12 (III)

(14) Let $T(n)$ be the function defined by

$T(n)=1$ and $T(n)=2T(n/2)+\sqrt{n}$

which is true.

(a) $T(n)=O(\sqrt{n})$ (b) $T(n)=O(\log_2 n)$ (c) $T(n)=O(n)$ ✓

(d) $T(n)=O(n^2)$

Dec 12 (III)

(51) Heapsort : $O(n \log(n))$

(14) sol. ~~Let $n=2^m$~~

~~$T(2^m)=2T(2^{m-1})+2^{m/2}$~~

~~Let $m=2$~~

~~$T(2^2)=2\underbrace{T(2^1)}_{=1}+2^1$~~

~~$=2+2=4$~~

~~$m=3$~~

~~$T(2^3)=2\underbrace{T(2^2)}_{=4}+2^{3/2}$~~

~~$=8+2^{3/2}$~~

(14) sol.

as $a=2$ $b=2$ and $f(n)=n^{1/2}$

Hence we can apply Master's Theorem.

$n^{\log_b a}=n^{\log_2 2}=n^1$

As ~~$n^1 > n$~~ as $1 > 1/2$

$\therefore$ $O(n)$ Ans

Sol. of June 13 (III)

12 $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$

$n = 2^m$

$T(2^m) = 2T(2^{m/2}) + \log 2^m$

$T(2^m) = 2T(2^{m/2}) + m$

For $m=0$

$T(1) = 2T(1) + 0$

$\Rightarrow T(1) = 0$

For $m=1$

$T(2^1) = 2T(\lfloor 2^{1/2} \rfloor) + 1$

$= 2T(\lfloor 1.44 \rfloor) + 1$

$= 2\underbrace{T(1)}_{=0} + 1$

$T(2) = 1$

$m=2$

$T(2^2) = 2\underbrace{T(2^1)}_{=1} + 2$

$= 4$

$m=4$

$T(2^4) = 2T(2^2) + 4$

$= 12$

$m=8$

$T(2^8) = 2T(2^4) + 8$

$= 32$

$n = 2^8$ [large enough]

$T(64)$

$T(32)$

$T(16)$

$T(8)$

$2T(4) + 4 \leftarrow T(16)$

$2T(2) + 2 \leftarrow T(4)$

$O(n \log\log\log n) = n \log\log\underbrace{\log 2^8}_{} = n \log\underbrace{\log 8}_{}$

$= n \log_2 3 = 2^8 \times$ greater than 1

which is much larger than 32

Basically, we can reject all the options where we have n in multiplication as $n = 2^8$ and ans shd be only 22

So, check $O(\log\log n) = \log \underbrace{\log 2^8}$
$= \log 8 = 3$ too small

$O(\log n \log\log n) = \log n \log \underbrace{\log 2^8}$
$= \log n \underbrace{\log 8}$
$= \log_2 {}^8 \times 3$
$= 8 \times 3 = 24 \simeq 22$

∴ Ans → D

DEC13(III)

34) ______ Comparisons are necessary in the worst case to find both the max. and min. of n numbers.

a) $2n-2$ b) $n + \lfloor \lg n \rfloor - 2$ (c) $\lfloor \frac{3n}{2} \rfloor - 2$

(d) $2\lg n - 2$

DEC13(II)

35 Big O estimate for

$f(x) = (x+1)\log(x^2+1) + 3x^2$

$= x\log(x^2+1) + \log(x^2+1) + 3x^2$

Biggest term is x^2

∴ $O(x^2)$ Ans

June (13) III

12) The solution of recurrence relation

$T(n) = 2T(\text{floor}(\sqrt{n})) + \log n$

a) $O(n \log\log\log n)$ (b) $O(n \log\log n)$

c) $O(\log\log n)$ d) $O(\log n \log\log n)$

13)

19) ~~An imo~~ graphics.

30), 31

June 12 (III)

(31) The upper bound of computing time of m coloring decision problem is

a) $O(nm)$ b) $O(n^m)$ c) $O(nm^n)$ d) $O(n^m m^n)$

Dec 13 (III)

(38) Assuming there are n keys and each key is in the range [0, m-1]. The run time of bucket sort is

(a) $O(n)$ (b) $O(n \lg n)$ (c) $O(n \lg m)$ (d) $O(n+m)$

Matrix Multiplication (Dec 13 (III)) [n x n matrices]

c[i][j] = c[i][j] + a[i][k] * b[k][j]

In general, as we have 3 (nested) loops. So. Complexity $O(n^3)$. But there are some improvement :—

Strassen algo : $O(n^{2.807355})$

(35) Coppersmith - Winograd algo : $O(n^{2.375477})$

(2014) Francois Le Gall algo : $O(n^{2.3728639})$ (Most efficient).

June 14 (III)

63) The solution of the recurrence relation of

$$T(n) = 3T\left(\text{floor}\left(\frac{n}{4}\right)\right) + \underline{n}$$

(a) $O(n^2)$ (b) $O(n \lg n)$ (c) $O(n)$ (d) $O(\lg n)$

(Master Theorem)

(66) $a = 3$ $b = 4$ $f(n) = n^1$

$$n^{\log_b a} = n^{\log_4 3} \Rightarrow n^{(\text{less than } 1)} \quad \text{as } \log_4 4 = 1$$

$$\therefore O(n)$$