

USN

--	--	--	--	--	--	--	--	--	--	--



INTERNAL ASSESSMENT TEST – I (SOLUTION)

Sub :	Design and Analysis of Algorithm						Code:	21CS42	
Date:	05/ 07 / 2023	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	AInDS

1(a)	<p>Define time complexity and Space Complexity. Explain the following problem types: (a) Sequencing (b) Sorting (C) Combinatorial</p> <p>Solution: Time complexity: Time complexity is most commonly evaluated by considering the number of elementary steps required to complete the execution of an algorithm. or how many times the basic operation is executed. It is expressed as a function of time taken by an algorithm</p> <p>Space Complexity is a combination of Fixed Space and Variable or Dynamic space. Fixed/ Static Space Requirements are independent of the characteristics of inputs and outputs – Instruction space (space for code) – Space for simple variables, fixed-size structured variable, constants (space for data) Variable/ Dynamic Space Requirements depend on -Number, Size, Values of inputs and outputs associated with the algorithm Recursive stack space, formal parameters, local variables, return addresses Memory requirement when invoked creates an environment stack.</p> <p>Sequencing problems: Sequencing problems are concerned with an appropriate order (sequence) for a series of jobs to be done on a finite number of service facilities (like machines) in some well-defined technological order so as to optimize some efficiency measure such as total elapsed time or overall cost etc.</p> <p>Sorting is a very classic problem of reordering items (that can be compared, e.g., integers, floating-point numbers, strings, etc) of an array (or a list) in a certain order (increasing, non-decreasing (increasing or flat), decreasing, non-increasing (decreasing or flat), lexicographical, etc).</p> <p>Combinatorial Problems: These are problems that use permutations and combinations to reach a solution. However, number of combinatorial objects typically grows extremely fast with a problem's size, reaching unimaginable magnitudes even for moderate-sized instances. Also, there are no known algorithms for solving most such problems exactly in an acceptable amount of time.</p>	[05]	CO 1	L1
		2		
		3		

CI Signature

CCI Signature

HOD Signature

1(b)

Write an algorithm to find the maximum and minimum element using divide and conquer approach to the list [31, 22, 12, -7, 75, -6, 17, 47, 60].

[05]

CO
1

L3

Solution:

Algorithm: Max - MinDC(A [i, j])

// input: Array A with lower and upper indices i and j

// output: maximum and minimum elements of the array

Begin

if $(j - i) \leq 1$ then return $(\max(\text{numbers}[i], \text{numbers}[j]), \min(\text{numbers}[i], \text{numbers}[j]))$

else

mid = $\lfloor (i + j) / 2 \rfloor$

(max1, min1) := Max - MinDC(A[i, mid])

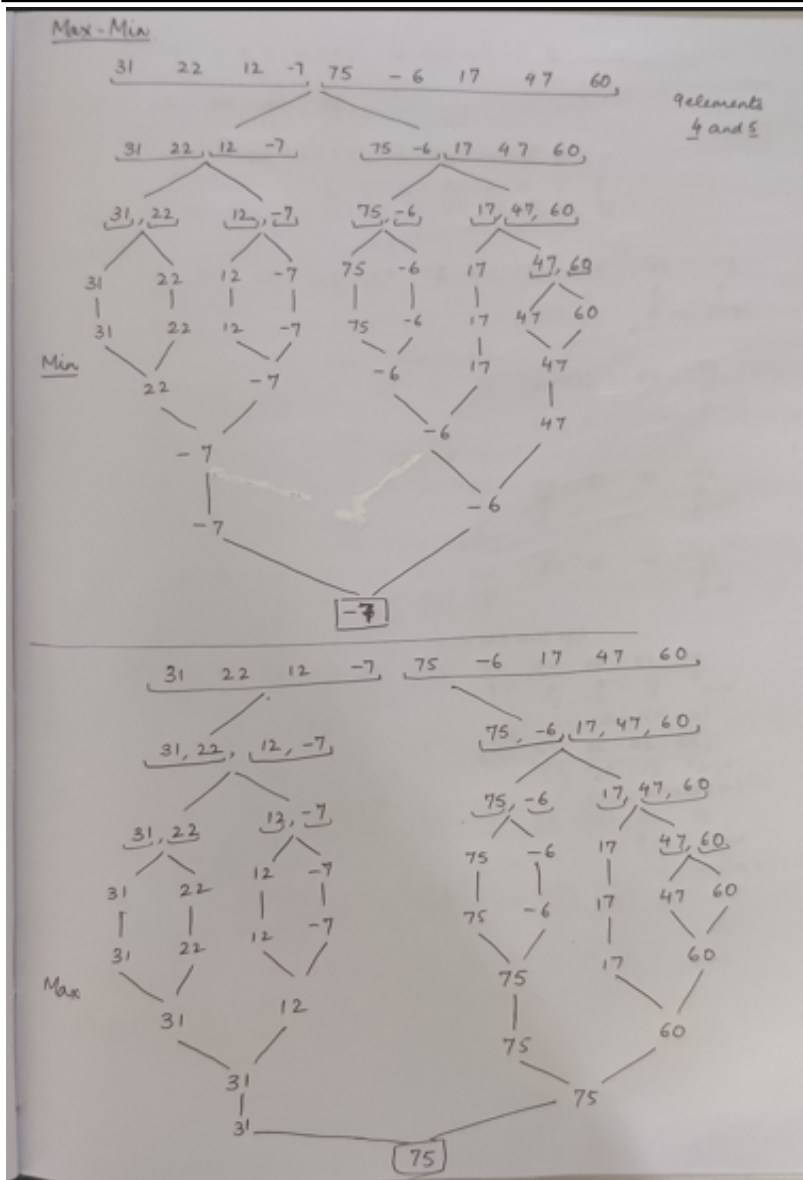
(max2, min2) := Max - MinDC(A[mid+1, j])

return $(\max(\max1, \max2), \min(\min1, \min2))$

End

2

3



	$n^2/2 \geq (n/2)$ $n^2 \geq n$ $n \geq 1$ <p>Therefore, $f(n) = \Omega(g(n))$ where $c=1/2$ and $n_0 = 1$</p>	2		
3(a)	<p>Write a merge-sort algorithm and analyze the time complexity and space complexity of it. Discuss the disadvantages of merge sort over quick sort.</p> <p>Solution:</p> <p>Algorithm Merge(B, C, A) <i>// Merges two sorted arrays into one sorted array</i> <i>// Input: Arrays B and C both sorted</i> <i>// Output: Sorted array A</i></p> <p>Begin</p> <p>$i = 0; j = 0; k = 0;$ <i>// pointer i, j, k track array B, C, A respectively</i> $m = \text{length}(B); n = \text{length}(C)$ while ($i \leq m$) and ($j \leq n$) do <i>// copy larger element of B or C to A</i> if ($B[i] < C[j]$) then $A[k] = B[i]$ $i++$</p> <p> else $A[k] = C[j]$ $j++$</p> <p> End if $k++;$ end while</p> <p>if ($i > m$) then <i>// copy remaining elements of C to A</i> while ($k \leq m+n$) do $A[k] = C[j];$ $k++; j++;$</p> <p> Else if ($j > n$) then <i>// copy remaining elements of B to A</i> while ($k \leq m+n$) do $A[k] = B[i]$ $k++; i ++;$</p> <p> Return (A);</p> <p>End</p> <p>Algorithm: Mergesort (A [first ... last])</p> <p><i>// Sorts array A[0..n-1] by recursive mergesort</i> <i>// Input: Array A[0..n-1] of unsorted elements, where first = 0 and last = n-1</i> <i>// Output: Array A[0.. n-1] sorted in non decreasing order</i></p> <p>Begin</p> <p>if ($\text{first} == \text{last}$) then return A[first]; else <i>// Find mid and split the list A</i> $\text{Mid} = \lfloor (\text{first} + \text{last}) / 2 \rfloor$ For $i = 0$ to mid <i>// copy elements of A[0.. mid] to intermediate array B</i> $B[i] = A[i]$ For $i = \text{mid} + 1$ to last $C[i] = A[i]$ <i>// copy elements of A[mid+1... n-1] to intermediate array C</i></p> <p> Mergesort(B[0...mid]) <i>// Call recursively</i> Mergesort(C[mid+1...n-1]) merge(B,C,A) <i>// Conquer and merge</i></p> <p>End</p>	[06]	CO 2	L2
		2		

Analysis:

- Input parameter: 'n' size array
- Basic Operation: Comparison and Merging
- Minimal Variation
- Merge Sort divides the array into two equally sized parts.

Time complexity depends on the *number of division stages*. Recursion Relation based on comparisons

$$C(n) = C(n/2) + C(n/2) + C_{\text{merge}}(n) \quad n > 1$$
$$= 0 \quad n = 1$$

For the worst case, 1. When all the elements are already sorted in descending order.

2. For merge; neither of two arrays becomes empty before the other one contains just one element (e.g. smaller elements may come from the alternating arrays). Therefore, in worst case $C_{\text{merge}}(n) = n - 1$,

Applying Masters Theorem:

$$C(n) = 2C(n/2) + n - 1$$

$$a=2; b=2^1; k=1$$

$$2 = 2^1;$$

According to Master's theorem when $a = b^k$ $C(n) \in \Theta(n^k \log n)$

Therefore, Worst case $C(n) = \Theta(n \log n)$

Best Case:

1. When elements are already sorted in ascending order.

2. For merge, one of the two array becomes empty before the other one.

Therefore, in best case

$C_{\text{merge}}(n) = n/2$. (Minimum number of comparisons will be $n/2$ when all elements of the first array are less than the elements of the second array).

The recurrence relation is $C(n) = 2C(n/2) + n/2$

$$a=2; b=2^1; k=1$$

$$2 = 2^1;$$

According to Master's theorem when $a = b^k$ $C(n) \in \Theta(n^k \log n)$

Therefore, Best case $C(n) = O(n \log n)$

Average Case: When elements are jumbled (neither ascending or descending).

Average case is again $\Theta(n \log n)$

In mergesort, all elements are copied into an auxiliary array of size n , where n is the number of elements in the unsorted array.

Hence, *Space Complexity* for merge sort is $O(n)$.

3

	<p>Disadvantages of mergesort</p> <ol style="list-style-type: none"> 1. It is not an inplace sorting algorithm. It <i>uses more memory space</i> to store the sub elements of the initial split list. 2. It works well for <i>large arrays and not for small arrays</i>. When size of list is very small, a large amount of time is wasted in recursion. ($n \leq 15$) 3. It uses a stack. So it requires extra space for storing activation records. 	1		
3(b)	<p>Define an algorithm. Explain the characteristics of an algorithm.</p> <p>Solution: Algorithm: It is a sequence of unambiguous instructions for solving problems. For obtaining a required output for any legitimate input in a finite amount of time. Characteristics:</p> <ol style="list-style-type: none"> 1. Input: 0 or more input values. 2. Output: Must generate some result. Function must do something, maybe just return void. 3. Definiteness: Every statement should be clear and unambiguous. 4. Finiteness: Should have limited steps. Must terminate at some point. Example: a web server has to stop service at some point. Can't just keep providing service endlessly. 5. Effectiveness: It should reach a solution. 	[04]	CO 1	L1
4(a)	<p>Write recursive code for Fibonacci series where time complexity is linear (instead of linear-exponential).</p> <p>Solution: The Fibonacci numbers: (used in predicting prices stocks and commodities) 0, 1, 1, 2, 3, 5, 8, 13, 21, ...</p> <p>The Fibonacci recurrence: $F(n) = F(n-1) + F(n-2)$ for $n > 1$ $F(0) = 0$ $F(1) = 1$</p> <p>Method of backward substitutions to solve recurrence fails to get an easily discernible pattern. Use Polynomial Reduction Method</p> <p>General 2nd order linear homogeneous recurrence with constant coefficients: $ax(n) + bx(n-1) + cx(n-2) = f(n)$</p> <p>Where, a, b, and c are real numbers with $a \neq 0$ and $x(n)$ is an unknown sequence. For homogeneous recurrence, $f(n) = 0$.</p>	[05]	CO 2	L1

Fibonacci Numbers (Recursive Method)

$$f(n) = f(n-1) + f(n-2)$$

$$\Rightarrow f(n) - f(n-1) - f(n-2) = 0 \quad \text{--- (1)}$$

Compare with $ax(n) + bx(n-1) + cx(n-2) = 0$

It's characteristic eqn. $ar^2 + br + c = 0$

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Wegelt; $r^2 - r - 1 = 0$ $a=1; b=-1; c=-1$

$$r_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{-(-1) \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2(1)}$$
$$= \frac{1 \pm \sqrt{5}}{2}$$

$$\therefore r_1 = \frac{1 + \sqrt{5}}{2}; \quad r_2 = \frac{1 - \sqrt{5}}{2}$$

Put in $x(n) = \alpha r_1^n + \beta r_2^n$

$$f(n) = \alpha \left(\frac{1 + \sqrt{5}}{2}\right)^n + \beta \left(\frac{1 - \sqrt{5}}{2}\right)^n \quad \text{--- (2)}$$

Initial conditions; $f(0) = 0; f(1) = 1$

$$f(0) = \alpha \left(\frac{1 + \sqrt{5}}{2}\right)^0 + \beta \left(\frac{1 - \sqrt{5}}{2}\right)^0 = 0$$

$$\Rightarrow \alpha + \beta = 0 \quad \boxed{\beta = -\alpha}$$

$$f(1) = \alpha \left(\frac{1 + \sqrt{5}}{2}\right)^1 + \beta \left(\frac{1 - \sqrt{5}}{2}\right)^1 = 1 \quad \beta$$

Put $\beta = -\alpha$

$$f(1) = \alpha \left(\frac{1 + \sqrt{5}}{2}\right) - \alpha \left(\frac{1 - \sqrt{5}}{2}\right) = 1 \quad (\text{Solve})$$

$$\boxed{\alpha = \frac{1}{\sqrt{5}}}$$

$$f(n) = \frac{1}{\sqrt{5}} \left[\frac{1 + \sqrt{5}}{2}\right]^n - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2}\right)^n$$

$$= \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2}\right)^n - \left(\frac{1 - \sqrt{5}}{2}\right)^n \right]$$

Golden Ratio; $\phi = \frac{1 + \sqrt{5}}{2}; \quad \hat{\phi} = \frac{1 - \sqrt{5}}{2}$

$$f(n) = \frac{1}{\sqrt{5}} [\phi^n - \hat{\phi}^n]$$

// Compute the nth Fibonacci number recursively using its definition

Algorithm Fib(n)

//Input: A non negative integer n

//Output: The nth Fibonacci number

Begin

if $n \leq 1$ return n;

else return Fib(n-1) + Fib(n-2)

End

1

2

	<p>Analysis: Input Parameter: 'n' the number Basic Operation: Addition Depends on 'n' – no variations Algorithm recursive relation: $\text{Fib}(n) = \text{Fib}(n-1) + \text{Fib}(n-2) \quad \text{for } n > 1$ $\text{Fib}(0) = 0$ $\text{Fib}(1) = 1$ Let number of additions performed to calculate F(n) be A(n). Based on Basic Operation: $A(n) = A(n-1) + A(n-2) + 1 \quad \text{for } n > 1, \text{ // Additions for } n-1, n-2 \text{ and } +1$ <i>addition to calculate the sum</i> $A(0) = 0 \text{ and } A(1) = 0$ The recurrence relation is $A(n) - A(n-1) - A(n-2) = 1$</p> <p><i>Solve</i> $A(n) = A(n-1) + A(n-2) + 1$ $A(n) - A(n-1) - A(n-2) = 1$ $\Rightarrow (A(n)+1) - (A(n-1)+1) - (A(n-2)+1) = 0$ $B(n) = A(n)+1$ $B(n) - B(n-1) - B(n-2) = 0$ $B(0) = 1 ; B(1) = 1$</p> <p><i>Applying same solution as for f(n)</i> $B(n) = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)$</p> $B(n) = A(n)+1$ $A(n) = B(n-1) = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n) - 1$ <p><i>For large values of n, subtracting 1 becomes negligible and $\hat{\phi}^n$ is inverse of ϕ^n, so negligible.</i></p> $A(n) \approx \frac{1}{\sqrt{5}} \phi^n$ $A(n) \in \Theta(\phi^n) \quad \text{Exponential growth}$	1½				
4(b)	<p>Consider functions function_1 and function_2 expressed in pseudo code as follows:</p> <table border="0" style="width: 100%;"> <tr> <td style="width: 50%; vertical-align: top;"> <p>Function_1 While n > 1 do for i = 1 to n do x = x+1; end for n = floor(n/2) end_while</p> </td> <td style="width: 50%; vertical-align: top;"> <p>Function_2 for i = 1 to 100*n do x = x+1; end_for</p> </td> </tr> </table>	<p>Function_1 While n > 1 do for i = 1 to n do x = x+1; end for n = floor(n/2) end_while</p>	<p>Function_2 for i = 1 to 100*n do x = x+1; end_for</p>	[05]	CO 1	L2
<p>Function_1 While n > 1 do for i = 1 to n do x = x+1; end for n = floor(n/2) end_while</p>	<p>Function_2 for i = 1 to 100*n do x = x+1; end_for</p>					

Let $f_1(n)$ and $f_2(n)$ denote the number of times the statement “ $x=x+1$ ” is executed in Function_1 and Function_2, respectively. Define the relation between $f_1(n)$ and $f_2(n)$ in terms of asymptotic notations.

Solution:

Function-1
 while $n > 1$ do _____ $\log n$
 {
 for $i = 1$ to n do _____ $n \cdot \log n$
 $x = x + 1$; _____ $n \cdot \log n$.
 $n = \text{floor}(n/2)$;
 }
 when $\frac{a}{2^k} \leq 1$ loop terminates
 $a \leq 2^k$
 $n \leq 2^k$ As $n = a$ (assumed)
 $k = \log_2 n$
 $\therefore f_1(n) = n \log n \in O(n \log n)$ — ①

Function-2
 for $i = 1$ to $100 \times n$ do _____ $100n$
 $x = x + 1$; _____ $100n$ times
 $\therefore f_2(n) = 200n$
 $= O(n)$ — ②

From ① and ②
 $f_1(n) \geq f_2(n)$ for large values of n
 $\therefore f_1(n) \in \Omega(f_2(n))$

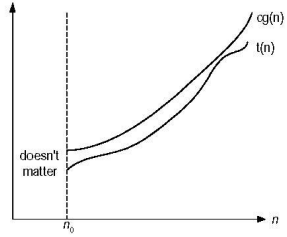
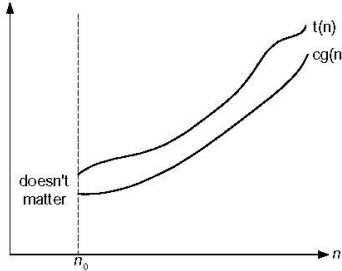
Trace the algorithm

n	i	x
a	1	$x+1$
	2	$x+2$
	3	$x+3$
\vdots		
	a	$x+a$
	$a/2$	$x+a+1$
$a/2$	1	$x+a+1$
	2	$x+a+2$
	3	$x+a+3$
\vdots		
	$a/2$	$x+a+a/2$
	$a/4$	$x+a+a/2+1$
$a/4$	1	$x+a+a/2+1$
	2	$x+a+a/2+2$
	\vdots	
\vdots		
	$a/4$	$x+a+a/2+a/4$
	\vdots	
\vdots		
	$\frac{a}{2^k}$	$x+a+a/2+\dots+1$
	2	\dots
\vdots		
	$\frac{a}{2^k}$	$x+a+a/2+a/4+\dots+a/2$
	2	\dots

2
1
1/2
1/2

5. Apply both merge sort and quick sort algorithm to sort the characters VTUBELAGAVI.
 There are two ways to do it.
 1. Consider alphabets as such and apply quicksort/ mergesort
 2. convert the alphabets into ASCII code and sort the numbers thus obtained.

[10
] C
O2 L2

<p>6(a)</p>	<p>Discuss asymptotic notation with examples.</p> <p>Solution: The word Asymptotic means approaching a value or curve arbitrarily closely (i.e., as some sort of limit is taken).</p> <p>Used to analyze the performance of an algorithm for some large data set. It is a mathematical representation of time complexity.</p> <p>Types of Asymptotic Notations:</p> <ol style="list-style-type: none"> Big Oh: $f(n)$ is in $O(g(n))$, denoted $f(n) \in O(g(n))$, if order of growth of $f(n) \leq$ order of growth of $g(n)$ (within constant multiple), i.e., there exist positive constant c and non-negative integer n_0 such that $f(n) \leq c g(n)$ for every $n \geq n_0$ <p>Example: Given $f(n) = 3n+2$, prove that $f(n) = O(n)$</p> <p>$3n+2=O(n)$ as $3n+2 \leq 4n$ for all $n \geq 2$, $c = 4$ and $n_0 = 2$, $g(n) = n$.</p> <p>$3n+2=O(n)$ as $3n+2 \leq 5n$ for all $n \geq 1$, $c = 5$ and $n_0 = 1$, $g(n) = n$.</p>  <p>Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$</p> <ol style="list-style-type: none"> Big Omega: A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is bounded below by some constant multiple of $g(n)$ for all large n, i.e., <u>if there exist some positive constant c and some nonnegative integer n_0 such that $t(n) \geq cg(n)$ for all $n \geq n_0$</u> <p>Example: $3n^2 + n \in \Omega(n)$</p> <p>Here, $t(n) = 3n^2 + n$; and $g(n)=n$</p> $3n^2 + n \geq c \cdot n \quad \forall n \geq n_0$ $\geq 3n + n$ $\geq 4n$ <p>If $n = 1$, $c=4$, $3n^2 + n \geq 4n \quad \forall n \geq 1$</p> <p>Hence, $3n^2 + n \in \Omega(n)$</p>  <p>Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$</p> <ol style="list-style-type: none"> Theta: A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large n, i.e., <u>if there exist some positive constant c_1 and c_2 and some nonnegative integer n_0 such that</u> 	<p>[06]</p> <p>1</p> <p>5</p>	<p>CO 1</p>	<p>L1</p>
-------------	--	-------------------------------	-----------------	-----------

Example: $n(n-1)/2 \in \Theta(n^2)$ $c_2 g(n) \leq t(n) \leq c_1 g(n)$ for all $n \geq n_0$

Here, $t(n) = n(n-1)/2$; and $g(n) = n^2$

$$c_2 \cdot n^2 \leq n(n-1)/2 \leq c_1 \cdot n^2 \quad \forall n \geq n_0$$

Equation 1: $c_2 \cdot n^2 \leq n(n-1)/2$

$$n(n-1)/2 \geq c_2 \cdot n^2$$

$$n^2/2 - n/2 \geq n^2/2 - \frac{1}{2} \cdot n^2$$

$$\geq 1/4n^2$$

If $n = 2$, $c = 1/4$, $n(n-1)/2 \geq 1/4n^2$

Hence, $n(n-1)/2 \in \Omega(n^2) \quad \forall n \geq 2$

Equation 2: $n(n-1)/2 \leq c_1 \cdot n^2 \quad \forall n \geq n_0$

$$n^2/2 - n/2 \leq \frac{1}{2} n^2$$

If $n = 1$, $c = 1/2$, $n(n-1)/2 \leq \frac{1}{2} n^2$

Hence, $n(n-1)/2 \in O(n^2) \quad \forall n \geq 1$

From 1 and 2, $c_2 \cdot n^2 \leq n(n-1)/2$ and $n(n-1)/2 \leq c_1 \cdot n^2$

Hence, $n(n-1)/2 \in \Theta(n^2)$

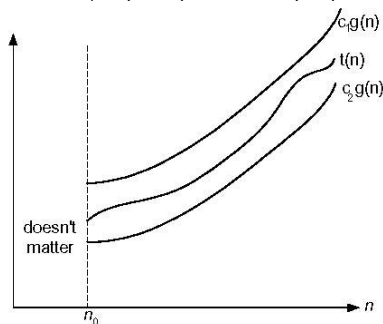


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

4. Little oh : A function $f(n)$ is in $o(g(n))$, denoted $f(n) \in o(g(n))$, if order of growth of $f(n) <$ order of growth of $g(n)$ (within constant multiple), i.e., there exist positive constant c and non-negative integer n_0 such that

$$f(n) < c g(n) \text{ for every } n \geq n_0$$

5. little omega: A function $t(n)$ is said to be in $\omega(g(n))$, denoted $t(n) \in \omega(g(n))$, if $t(n)$ is bounded below by some constant multiple of $g(n)$ for all large n , i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) > c g(n) \text{ for all } n \geq n_0$$

6(b) List out the advantages and disadvantages of Divide and Conquer technique.

Solution:

Advantages:

- Easily solve large problems faster than other algorithms.
- It solves simple sub-problems within the cache memory instead of accessing the slower main memory.
- It supports parallelism because sub-problems are solved independently.
- Hence, the algorithm made using this approach runs on the multiprocessor system

[04]

CO
2

L1

	<p>Disadvantages:</p> <ul style="list-style-type: none">● It uses recursion therefore sometimes memory management is a very difficult task.● An explicit stack may overuse the space.● It may crash the system if the recursion is carried out rigorously (require Stack greater than the stack present in the CPU).			
--	--	--	--	--

Answer any 5 full questions