UNIT - 3.

Algorithms.

An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

An algorithm can also be described using a computer language. When this is done, only those instructions permitted in the language can be used. This often leads to a description of the algorithm that is complicated and difficult to understand. So instead of using a particular computer language to specify algorithms, a form of Pseudocode is used.

"Pseudocode" provides an intermediate step between an English language description of an algorithm and an implementation of this algorithm in a programming languages.

There are several properties that algorithm should satisfy

1. Input: An algorithm has input values from a specified set.
2. Output: From each set of input values an algorithm produces output values from a specified set. The output values are the solution to the problem.
3. Definiteness: The steps of an algorithm must be defined precisely.
4. Correctness: An algorithm should produce the correct output values for each set of input values.

5. Finiteness: An algorithm should produce the desired output after a finite number of steps for any input in the set.

6. Effectiveness: It must be possible to perform each step of an algorithm exactly and in a finite amount of time.

7. Generality: The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

Example 1.

Describe an algorithm for finding the maximum value in a finite sequence of integers.

Soln:- We perform the following steps.

1. Set the temporary maximum equal to the first integer in the sequence.

2. Compare the next integer in the sequence to the temporary maximum, & if it is larger than the temporary maximum, set the temporary maximum equal to this integer.

3. Repeat the previous step if there are more integers in the sequence.

4. Stop when there are no integers left in the sequence. The temporary maximum at this point is the largest integer in the sequence.

(5)

A Pseudocode description of the algorithm for finding maximum Element in a finite sequence follows.

Procedure max($a_1, a_2 \ldots a_n$: integers)

max := a_1

for i = 2 to n

if max < a_i then max := a_i

{ max is the largest element }.

Show that above algorithm for finding the maximum Element in a finite sequence of integers has all the Properties listed

Soln:- The input to Algorithm 1 is a sequence of integers. The output is the largest integer in the sequence. Each step of the algorithm is Precisely defined, because only assignments, a finite loop & Conditional stmts occur. To show that the algorithm is Correct, we must show that when the algorithm terminates, the Value of the Variable max Equals the maximum of the terms of Sequence. The algorithm uses a finite number of steps, because it terminates after all the integers in the sequence have been Examined.

The algorithm Can be Carried out in a finite amount of time because Each step is Either a Comparison or an assignment. Finally it is general because it Can be used to find the maximum of any finite sequence of integers.

Searching Algorithms

The Problem of locating an Element in an ordered list is Searching Problem.
For instance, a Program that checks the spelling of words searches for them in a dictionary, which is just an ordered list of words.
The general Searching Problem can be described as follows: Locate an Element x in a list of distinct Elements $a_1, a_2 \ldots a_n$, or determine that it is not in the list.
The solution to this search Problem is the location of the term in the list Equals x (ie, i is the solution if $x = a_i$) and is 0 if x is not in the list.

Linear Search (sequential search).

The linear Search algorithm begins by Comparing
x and a_1. When $x = a_1$, the solution is the
location of a_1, namely, 1. When $x \neq a_1$, Compare
x with a_2. If $x = a_2$, the solution is the
location of a_2, namely 2. When $x \neq a_2$, Compare
x with a_3. Continue this Process, Comparing x
successively with Each term of the list until
a match is found, where the solution is the
location of that term, unless no match occurs.
If the Entire list has been Searched without
locating x, the solution is 0.

(8)

Linear Search Algorithm

Procedure linear Search (x: integer, $a_1, a_2 \ldots a_n$: distinct integers)

$i := 1$

while ($i \leq n$ and $x \neq a_i$)

$i := i + 1$

if $i \leq n$ then location $= i$

Else location $= 0$

{location is the subscript of the term that Equals x, or is 0 if x is not found}.

(2) The Binary Search

This algorithm can be used when the list has terms occuring in order of increaing size. It Proceeds by Comparing the Element to be located to the middle term of the list. The list is then split into two Smaller Sublists of the Same size, or where one of these smaller lists has one fewer term than the other. The search Continues by restricting the search to the appropriate sublist based on the Comparison of the Element to be located & the middle term.

Example

To search for 19 in the list

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22

first split this list, which has 16 terms, into two smaller list with eight terms each.

1 2 3 5 6 7 8 10   12 13 15 16 18 19 20 22

Then compare 19 and the largest term in the first list. Because $10<19$, the search for 19 can be restricted to the list containing the 9th through the 16th terms of the original list.

Next split this list, which has eight terms, into the smaller lists of four terms each, namely

12 13 15 16   18 19 20 22.

Because $16<19$, the search is restricted to the second of these list, which contains the 13th through the 16th terms of the original list.

The list 18 19 20 22 is split into two lists, namely

18 19 | 20 22

Because 19 is not greater than the largest term of the first of these two lists, which is also 19, the search is restricted to the first list: 18 19, which contains the 13th & 14th terms of the original list. Next, this list of two terms is split into two lists of one term each: 18 and 19. Because $18<19$, the search is restricted to the second list: the list containing 14th term of list.

4

Binary Search Algorithm

Procedure binary search (x: integer, $a_1, a_2 \dots a_n$: increasing integers)

$i := 1$ { i is left endpoint of search interval }

$j := n$ { j is right endpoint of search interval }

while $i < j$

begin

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

if $x > a_m$ then $i = m+1$

Else $j = m$

End

if $x = a_i$ then location $= i$

Else location $= 0$

{ location is the subscript of term equal to x, or 0 if x is not found }.

Sorting

Sorting is putting these elements into a list in which the elements are in increasing order. For instance, sorting the list 7, 2, 1, 4, 5, 9 produces the list 1, 2, 4, 5, 7, 9.

Sorting the list d, h, c, a, f (using alphabetical order) produces the list a, c, d, f, h.

Bubble sort :-

The bubble sort is one of the simplest sorting algorithms but not one of the most efficient.
It puts a list into increasing order by successively comparing adjacent elements, interchanging them if they are in wrong order
To carry out bubble sort, we perform the basic operation, that is, interchanging a larger element with a smaller one following it, starting at the beginning of the list, for a full pass.
We iterate this procedure until the sort is compute.

Ex Use bubble sort to put 3, 2, 4, 1, 5 into increasing order

Soln The steps of this algorithm are illustrated in figure 1.

First Pass

3	2	2	2
2	3	3	3
4	4	4	1
1	1	1	4
5	5	5	5

(5)

Second Pass

2	2	2
3	3	1
1	1	3
4	4	4
$\boxed{5}$	$\boxed{5}$	$\boxed{5}$

Third Pass

2	1
1	2
3	3
$\boxed{4}$	$\boxed{4}$
$\boxed{5}$	$\boxed{5}$

Fourth Pass

1
2
$\boxed{3}$
$\boxed{4}$
$\boxed{5}$

↳ : an interchange

(: Pair in correct order

Begin by comparing the first 2 elements, 3 and 2. Because 3>2, interchange 3 and 2, producing the list 2.3.4.1.5. Because 3<4, continue by comparing 4 and 1. Because 4>1, interchange 1 and 4, producing the list 2.3.1.4.5. Because 4<5, the first pass is complete. The first pass guarantees that largest element 5 is in the correct position.

The second pass begins by comparing 2 and 3. Because these are in the correct order, 3 and 1 are compared. Because 3>1, these numbers are interchanged, producing 2,1,3,4,5. Because 3<4, these numbers are in correct order. It is not necessary to do any more comparisons for this pass because

5 is already in the correct Position.
The Second Pass guarantees that the two largest Element, 4 and 5, are in their Correct Positions.

The third Pass begins by Comparing 2 and 1. These are interchanged because $2>1$, Producing 1,2,3,4,5. Because $2<3$, these 2 Elements are in the Correct order. It is not necessary to do any more Comparisons for this Pass because 4 and 5 are already in the correct Positions. The third Pass guarantees that the three largest Elements, 3,4 and 5 are in their Correct Positions.

The fourth Pass Consists of one Comparison, namely the Comparison of 1 and 2. Because $1<2$, these Elements are in Correct Order.

Algorithm

Procedure bubblesort ($a_1, a_2 -- a_n$: real number with $n \geq$

for $i=1$ to $n-1$

for $j=1$ to $n-i$

if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}

{ $a_1 -- a_n$ is in increasing order}.

Scanned by CamScanner

(6)

Insertion sort :-

The insertion sort is a simple sorting algorithm. To sort a list with n elements, the insertion sort begins with the second element.

The insertion sort compares this second element with the first element and inserts it before the first element if it does not exceed the first element and after the first element if it exceeds the first element. At this point, the first two elements are in correct order. The third element is then compared with the first element, & if it is larger than the first element, it is compared with the second element; it is inserted into the correct position among the first 3 elements.

Ex: Use the insertion sort to put the elements of the list 3, 2, 4, 1, 5 in increasing order.

Soln:- The insertion sort first compares 2 and 3. Because 3 > 2, it places 2 in the first position, producing the list 2, 3, 4, 1, 5 - At this point, 2 and 3 are in the correct order. Next, it inserts the third element, 4, into the already sorted part of the list by making the comparison 4 > 2 and 4 > 3. Because 4 > 3, 4 is placed in third position. At this point the list is 2, 3, 4, 1, 5 and we know that the ordering of first 3 elements is correct. Next, we find the correct place for the fourth element, 1, among the already sorted elements, 2, 3, 4. Because 1 < 2, We obtain the list 1, 2, 3, 4, 5. Finally, we insert 5 into the correct position by successively comparing it to 1, 2, 3 and 4. Because 5 > 4, it goes at the end of the list, producing the correct order for the entire list.

(7)

The Growth of Functions.

The time required to solve a Problem depends on more than only the number of operations it uses. The time also depends on the hardware and Software used to run the Program that implements the algorithm.

If we change the hardware and software used to implement an algorithm, we can closely approximate the time required to solve a Problem of size n by multiplying the Previous time required by a constant.

Big-O notation.

Using big-O ~~nat~~ notation, we do not have to worry about the hardware & software used to implement an algorithm.

Furthermore, using big-O notation, we can assume that the different operations used in an algorithm take the same time, which simplifies the analysis considerably.

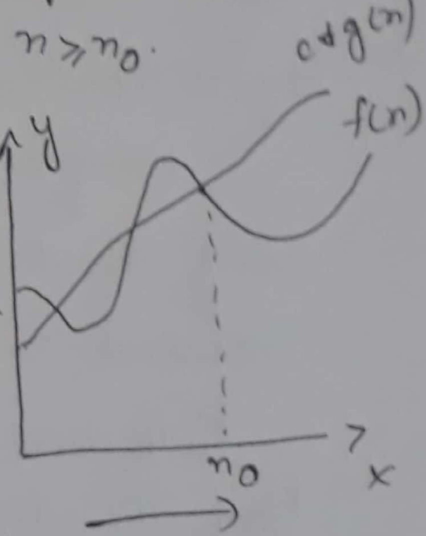
Big-O notation is used Extensively to Estimate the number of operations an algorithm uses as its input grows.

Using big-O notation, we can compare two algorithms to determine which is more Efficient as the size of input grows.

For instance, if we have two algorithms for solving a problem, one using $100n^2 + 17n + 4$ operations and the other using n^3 operations, big-O notation can help us see that the first algorithm uses far fewer operations when n is large, even though it uses more operations for small values of n, such as $n = 10$.

Definition

Let f and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that $f(x)$ is $O(g(x))$ if there are constants $C > 0$ and some non negative integers constant n_0, $f(n) \leq C * g(n)$ for all n, $n \geq n_0$.



For all values of $n > n_0$ function 'f' is atmost c times the function 'g'.

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

Read as "f of n is big oh of g of n".

(8)

→ Constant function

1. $f(n) = 16$

2. $f(n) = 27$

1Ans $f(n) \leq 16 * 1$, where $c = 16$ and $n_0 = 0$

2Ans $f(n) \leq 27 * 1$, where $C = 27$ and $n_0 = 0$.

→ Linear function

1. $f(n) = 3n + 5$.

2 $f(n) = 2n + 3$.

1 Ans. $f(n) = 3n + 5$

$3n + 5 \leq 4n$.

$n = 1$

$c = 4$.

$8 \leq 4$ ✗

$n = 2$

$11 \leq 8$ ✗

$n = 3$

$14 \leq 12$

$n = 4$

$n = 5$

$20 \leq 20$

$n \geq n_0$

$n \geq 5$

$C = 4$, $n_0 = 5$.

2 Ans $f(n) = 2n + 3$

$2n + 3 \leq 3n$

$c = 3$.

Let $n = 1$

$2n + 3 \leq 3n$

$5 \leq 3$ ✗

Let $n = 2$

$7 \leq 6$ ✗

Let $n = 3$

$9 \leq 9$. – $n \geq n_0 \Rightarrow n \geq 3$

$\therefore n_0 = 3$

Let $n = 4$

$11 \leq 12$ ✓

Let $n = 5$

$13 \leq 15$

So $f(n) = O(n)$

Quadratic function

1. $f(n) = 27n^2+16n$

2. $f(n) = 27n^2+16$

1Ans $f(n) = 27n^2+16n$

$27n^2+16n \leq 28n^2$

$C = 28$.

Let $n = 1$

$27+16 \leq 28$ ×

Let $n = 2$

$27\times4+16 \leq 28\times4$ ×

Let $n = 16$

$\therefore C = 28$. $n_0 = 16$.

So $f(n) = O(n^2)$.

2Ans $f(n) = 27n^2+16$

$C = 28$. $n_0 = 16$

$\therefore f(n) = O(n^2)$.

Example 1

Show that $f(n) = n^2+2n+1$ is $O(n^2)$

$n^2+2n+1 < 2n^2$ $\quad$ $f(n) \le c * g(n)$.

$\therefore c = 2$

$n = 1$

$1+2+1 < 2$

$4 < 2$. ×

$n = 2$.

$4+4+1 < 2(2)^2$

$9 < 2 * 4$

$9 < 8$ ×

Let $n = 3$

$9+6+1 < 2(3)^2$

$16 < 2 * 9$

$16 < 18$.

$\therefore n_0 = 3$ $\quad$ $\therefore c = 2, n_0 = 3$

So $f(n) = O(n^2)$.

The big-O symbol is sometimes called a Landau symbol after the German mathematician Edmund Landau.

Example 2.

Show that $7n^2$ is $O(n^3)$.

$7n^2 < 6n^2$

$c = 6$.

Let $n = 1$

$7 < 6$ ×

Let $n = 2$.

$7(2)^2 < 6(2)^2$

$7 \times 4 < 6 \times 4$

$28 < 24$

Let $n = 3$

$7(3)^2 < 6(3)^2$

$7 * 9 < 6 \times 9$

$63 < 54$.

$7x^2 < x^3$

$c = 1$

$n = 1$

$7 \leq 1$ ✗

$n = 2$

$7(2)^2 \leq (2)^3$

$7 \times 4 < 8$

$28 < 8$

$n = 3$

$7(3)^2 \leq (3)^3$

$7 \times 9 < 27$

$63 < 27$

Let $n = 4$

$7(4)^2 \leq (4)^3$

$7 * 16 < 64$

$112 < 64$

$\begin{array}{r} 16 \\ \times 4 \\ \hline 64 \end{array}$ $\begin{array}{r} 16 \\ \times 7 \\ \hline 112 \end{array}$

Let $n = 7$

$7(7)^2 \leq (7)^3$

$7 * 49 < 7 \times 7 \times 7$

$343 < 343$

$\begin{array}{r} 49 \\ \times 7 \\ \hline 343 \end{array}$

Let $n = 8$

$7(8)^2 < (8)^3$

$7 \times 8 \times 8 \leq 8 \times 8 \times 8$

$\therefore$ $n = \cancel{8}$, $c = 1$.

Theorem 1:

Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 + f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|))$.

Theorem 2.

Suppose that $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$. Then $(f_1 f_2)(x)$ is $O(g_1(x)\, g_2(x))$.

2. Show that $f(n) = 4n^2 - 64n + 288 = \Omega(n^2)$

Soln $\quad f(n) \geq C \times g(n)$

$$4n^2 - 64n + 288 \geq n^2$$

Let $n = 1$

$$4 - 64 + 288 \geq 1$$

$$= 228 \geq 1$$

$\therefore \; 4n^2 - 64n + 288 = \Omega(n^2)$

Hence Proved.

Big omega and Big Theta Notation.

Omega notation (Best case)

The lower bound for the function 'f' is Provided by the omega notation (Ω).

The function $f(n) = \Omega(g(n))$ iff there Exist Positive Constant C and n_0 such that

$$f(n) \geq C * g(n) \text{ for all } n, n \geq n_0.$$

1. Let $f(n) = 3n+2$. PT $f(n) = \Omega g(n)$

Soln

$$f(n) \geq C * g(n)$$

Let $g(n) = 2n$

$$3n+2 \geq 2n.$$

Let $n = 1$

$$5 \geq 2.$$

$\therefore c = 2$

$n_0 = 1$

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

2. Let $f(n) = 4n+6$

$$4n+6 \geq 3n$$

Let $n = 1$

$$10 \geq 3$$

$\therefore c = 3$

$n_0 = 1.$

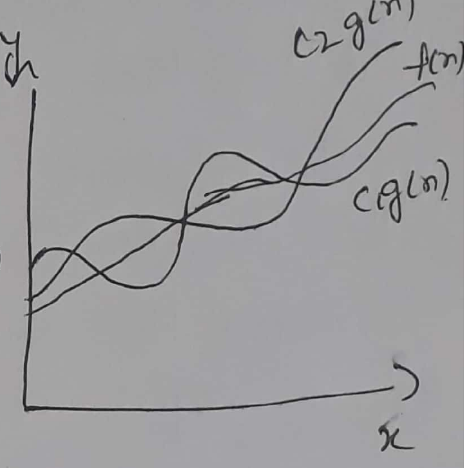
(11)

Theta Notation (Average case)

The function $f(n) = \theta(g(n))$ (read as 'f of n is theta of g of n') iff there exists positive constants c_1, c_2 and n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n, n \geq n_0.$$

The lower bound & upper bound for the function 'f' is provided by theta notation.



1. The function $3n+2$

$2n \leq 3n+2 \leq 4n$

$\therefore$ Let $n=1$ $\quad 2 \leq 5 \leq 4$ ✗

Let $n=2$ $\quad 4 \leq 8 \leq 8$ ✓

Let $n=3$ $\quad 6 \leq 11 \leq 12$ ✓

$\therefore c_1 = 2, \ c_2 = 4, \ n_0 = 2.$

$$\boxed{1 < \log n < \sqrt{n} < n < n\log n < n^2 < n^3 < \dots < 2^n < 3^n < n^n}$$

Growth of Functions.

Complexity of Algorithms.

-> Time Complexity
-> Space complexity.

Time Complexity

The time complexity of an algorithm can be expressed in terms of the number of operations used by the algorithm when the input has a particular size. The operations used to measure time complexity can be the comparison of integers, the addition of integers, the multiplication of integers, the division of integers, or any other basic operations.

Time complexity is described in terms of the number of operations required instead of actual computer time because of the difference in time needed for different computers to perform basic operations.