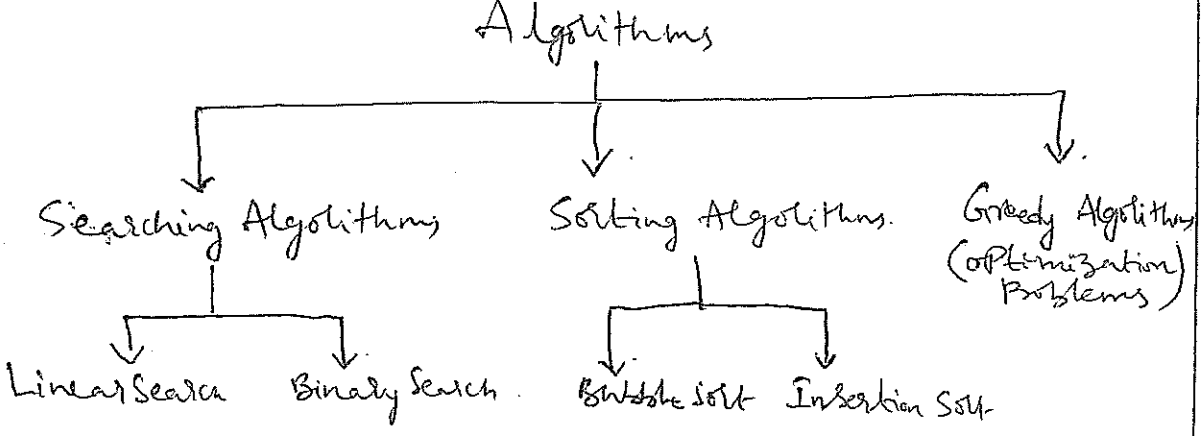
ALGORITHMS

Chapter-3
week - 6.

Algolithm:- An algolithm is a finite sequence of instructions for solving a problem.

PSEUDO CODE:- Pseudocode provides an intermediate step between an English language description of an algolithm and an implementation of this algolithm in a Programming language.



Linear search (or) Sequential search:-
(FOR LOCATING AN ELEMENT 'x' IN AN ORDERED LIST)

The solution to this search problem is the location of the term in the list that equals x

If the list of distinct elements is $\{a_1, a_2, \ldots a_n\}$ and to locate an element x in the list.

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

When $x = a_2$, the solution is the location of a_2, namely 2

When $x \neq a_2$, compare x with a_3.

Continue this process until a match is found, where the solution is the location of that term.

The solution is 0, if the entire list has been searched without locating x.

ALGORITHM LINEAR SEARCH

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}

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

NOTE:- For linear search, the ordered list need not be increasing order or decreasing order.

Example 1: To search for 6 in the list
1, 2, 4, 5, 6, 9

Sol Here $\{a_1, a_2, a_3, a_4, a_5, a_6\} = \{1, 2, 4, 5, 6, 9\}$

$\Rightarrow n = 6$ and $x = 6 = a_5$.

Now $i = 1$

$1 < 6$ and $x = a_5 \neq a_1$

$i = 1+1 = 2$

$2 < 6$ and $x = a_5 \neq a_2$.

$i = 2+1 = 3$.

$3 < 6$ and $x = a_5 \neq a_3$.

$i = 3+1 = 4$

$4 < 6$ and $x = a_5 \neq a_4$

$i = 4+1 = 5$

$5 < 6$ and $x = a_5 = a_5$ ✓

As $5 < 6$, then location := 5

return location := 5.

Example (2): To search for 7 in the list 1, 2, 4, 5, 6, 9

Sol Here $\{a_1, a_2, a_3, a_4, a_5, a_6\} = \{1, 2, 4, 5, 6, 9\}$.

and $n = 6$, $x = 7$ is not in list.

Now $i = 1$

$1 < 6$ and $x \neq a_1$

$i = 1+1 = 2$

$2 < 6$ and $x \neq a_2$

$i = 2+1 = 3$

$3 < 6$ and $x \neq a_3$

$i = 3+1 = 4$

$4 < 6$ and $x \neq a_4$

$i = 4+1 = 5$

$5 < 6$ and $x \neq a_5$

$i = 5+1 = 6$.

$6 = 6$ and $x \neq a_6$.

$i = 6+1 = 7$.

$7 \neq 6$, location := 0.

return 0.

BINARY SEARCH:- This method is used, when the list is in increasing order.

To search for the integer x in the list $\{a_1, a_2, \ldots a_n\}$ which is in increasing order i.e., $a_1 < a_2 < \cdots < a_n$

Split the list into two smaller list at $m = \lfloor \frac{n+1}{2} \rfloor$ to get two smaller lists $\{a_1, a_2, \ldots a_m\}$ and $\{a_{m+1}, a_{m+2}, \ldots a_n\}$.

Begin by comparing x with the middle term a_m.

If $x > a_m$, then search for x in the list $\{a_{m+1}, a_{m+2}, \ldots a_n\}$

otherwise $x \leq a_m$ search for x in the list $\{a_1, a_2, \ldots a_m\}$.

Now the search will continue in the restricted list either in $\{a_1, a_2, \ldots a_m\}$ or in $\{a_{m+1}, a_{m+2}, \ldots a_n\}$.

Repeat the procedure for the new restricted list until a list with one term is obtained.

ALGORITHM : BINARY SEARCH

Procedure binary search $\{x$: integer, $a_1, a_2, \ldots, a_n$: increasing integers$\}$

```
i = 1   { left endpoint of search interval }
j = n.  { Right endpoint of search interval }
    While i < j
        m := ⌊ (i+j)/2 ⌋
        If x > a_m then i = m+1
        else j = m
If x = a_i then location = i
else location = 0
return location.
```

Example ① Search 19 in the list

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

$\{a_1, a_2, a_3, a_4, a_5, \ldots a_{16}\} = \{1, 2, 3, 5, 6, 7, \ldots 22\}$

So Here $n = 16$ and $x = 19$

$i = 1$

$j = 16$

As $1 < 16$

$m = \lfloor \frac{1+16}{2} \rfloor = \lfloor \frac{17}{2} \rfloor = \lfloor 8.5 \rfloor = 8$

As $19 > a_m (= a_8 = 10)$ then $i = m+1$

$= 8+1$

$= 9$.

$i = 9$

$j = 16$.

9 to 16 means $\{a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}\} = \{12, 13, 15, 16, 18, 19, 20, 22\}$

As $9 < 16$

$m = \lfloor \frac{9+16}{2} \rfloor = \lfloor \frac{25}{2} \rfloor = \lfloor 12.5 \rfloor = 12$.

As $19 > a_m (= a_{12} = 16)$ then $i = m+1$

$= 12+1$

$= 13$

$i = 13$.

$j = 16$

13 to 16 means $\{a_{13}, a_{14}, a_{15}, a_{16}\} = \{18, 19, 20, 22\}$

As $13 < 16$

$m = \lfloor \frac{13+16}{2} \rfloor = \lfloor \frac{29}{2} \rfloor = \lfloor 14.5 \rfloor = 14$

As $19 \neq a_m (= a_{14} = 19)$

$j = m$

$\Rightarrow j = 14$.

$i = 13$

$j = 14$

13 to 14 means $\{a_{13}, a_{14}\} = \{18, 19\}$

As $13 < 14$ $m = \lfloor \frac{13+14}{2} \rfloor = \lfloor \frac{27}{2} \rfloor = \lfloor 13.5 \rfloor = 13$.

As $19 > a_m (= a_{13} = 18)$ then $i = m+1$
$= 13+1$
$= 14$

$i = 14$
$j = 14.$

$14 \not< 14$ $(i \not< j)$

As $x (= 19) = a_i (= a_{14} = 19)$ then location $= i (= 14)$

Return location $(= 14)$.

SORTING:- Putting elements into a list where the elements are in increasing order.

BUBBLE SORT:- Bubble sort puts a list into increasing order by successively comparing adjacent elements, interchanging them if they are in the wrong order, starting at the beginning of the list, for a full pass. We iterate this procedure until the sort is complete.

ALGORITHM BUBBLE SORT

Procedure bubble sort $\{a_1, a_2, \ldots a_n$: Real numbers with $n \geq 2\}$
For $i = 1$ to $n-1$ $\longrightarrow$ First pass to $(n-1)$ pass.
{For $j = 1$ to $n-i$
if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}.

$\{a_1, a_2, \ldots, a_n\}$ is in increasing order.

Example. Using bubble Sort Put the list 3,2,4,1,5 into increasing order.

Sol: $\{a_1, a_2, a_3, a_4, a_5\} = \{3, 2, 4, 1, 5\}$

Here $n = 5$.

For $i = 1$ to $n-1 (= 5-1 = 4)$

$i=1$ For $j = 1$ to $n-i (= 5-1 = 4)$

As $a_1 > a_2$ interchange a_1 and a_2 $= \{2, 3, 4, 1, 5\}$.
$= \{a_1, a_2, a_3, a_4, a_5\}$

Now $a_2 \not> a_3$.

As $a_3 > a_4$ interchange a_3 and a_4 $= \{2, 3, 1, 4, 5\}$
$= \{a_1, a_2, a_3, a_4, a_5\}$.

Now $a_4 \not> a_5$ and nothing is there to be compared after first Pass we get $\{a_1, a_2, a_3, a_4, a_5\} = \{2, 3, 1, 4, 5\}$.

$i=2$ For $j = 1$ to $n-i (= 5-2 = 3)$.

Now $a_1 \not> a_2$

As $a_2 > a_3$ interchange a_2 and a_3 $= \{2, 1, 3, 4, 5\}$
$= \{a_1, a_2, a_3, a_4, a_5\}$

And $a_3 \not> a_4$

$i=3$ For $j = 1$ to $n-i (= 5-3 = 2)$.

As $a_1 > a_2$ interchange a_1 and a_2 $= \{1, 2, 3, 4, 5\}$
$= \{a_1, a_2, a_3, a_4, a_5\}$

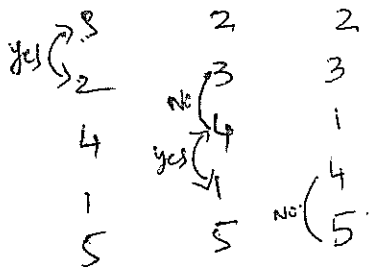
And $a_2 \not> a_3$

$i=4$ For $j = 1$ to $n-i (= 5-4 = 1)$

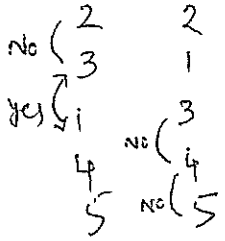
As $a_1 \not> a_2$ and no more j's are available.

$\{a_1, a_2, a_3, a_4, a_5\} = \{1, 2, 3, 4, 5\}$ is in increasing order.

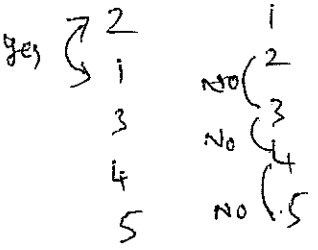
$i=1$ (First Pass)



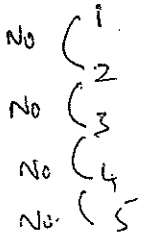
$i=2$ (Second Pass).



$i=3$ (Third Pass)



$i=4$ (Fourth Pass)



INSERTION SORT :- To sort a list of n elements, the insertion sort begins with the second element and compares this second element with first element and inserts it before the first element if it is not bigger than first element and after the first element if it is bigger than first element. At this point, the first two elements are in the correct order.

The third element is then compared with the first element, and if it bigger than the first element, then the third element is compared with second element. It is inserted into the correct position among the first three elements.

The algorithm continues until the last element is placed in the correct position relative to the already sorted list of the first (n-1) elements.

ALGORITHM INSERTION SORT

Procedure insertion sort $(a_1, a_2, \ldots)$ an : real numbers with $n \geq 2$)

For $j = 2$ to n.

$i = 1$

While $a_j > a_i$

$i = i+1$

$m = a_j$

for $k = 0$ to $j - i - 1$

$a_{j-k} = a_{j-k-1}$

$a_i = m$

$\{a_1, a_2, \ldots a_n\}$ is in increasing order.

Example use insertion sort and put the elements of the list 3, 2, 4, 1, 5 in increasing order.

Sol The list is 3, 2, 4, 1, 5
The insertion sort first compares 2 and 3.

As $3 > 2$, it places 2 in first position and the list becomes 2, 3, 4, 1, 5.

Next it inserts the 3rd element, 4, into the already sorted part of the list i.e., 2, 3.

As $4 > 2$ and $4 > 3$, 4 remains in the third position and the list becomes 2, 3, 4, 1, 5

Next it inserts the 4th element, 1, into the already sorted part of the list i.e., 2, 3, 4

As $1 < 2$, it places 1 in first position and the list becomes 1, 2, 3, 4, 5.

Next it inserts the 5th element, 5, into the already sorted part of the list i.e., 1, 2, 3, 4.

As 5>1, 5>2, 5>3, 5>4, it stays at the end of the list, producing the order 1, 2, 3, 4, 5.

GREEDY ALGORITHMS (OPTIMIZATION PROBLEMS)

Algorithms that makes what seems to be the "best" choice at each step are called greedy algorithms.

ALGORITHM Greedy change-making Algorithm

Procedure change ($c_1, c_2, \ldots, c_r$: values of denominations of coins, where $c_1 > c_2 > \cdots > c_r$; n : a positive integer)

For $i = 1$ to r

$d_i = 0$ { d_i counts the coins of denomination c_i }.

while $n \geq c_i$

$d_i = d_i + 1$

$n = n - c_i$

{ d_i is the number of coins of denomination c_i in the change }.

Example: Make a change for 67 cents. Using quarters, dimes, nickels and pennies. Using fewest coins possible.

Solution

Here $n = 67$

$c_1 = 25$, $c_2 = 10$, $c_3 = 5$, $c_4 = 1$, means types of coins

number of coins $r = 4$.

For $i = 1$ to 4

$d_1 = 0$.

As $67 > c_1 (= 25)$

$d_1 = d_1 + 1 = 0 + 1 = 1$

$n = n - c_i = 67 - 25 = 42$

As $42 > c_1 (= 25)$

$d_1 = d_1 + 1 = 1 + 1 = 2$

$n = n - c_i = 42 - 25 = 17$.

Now $n \not> c_1$

$d_2 = 0$

As $17 > c_2 (= 10)$

$d_2 = d_2 + 1$

$= 0 + 1 = 1$

$n = n - c_2 = 17 - 10 = 7$

Now $n \not> c_2$.

$d_3 = 0$

As $7 > c_3 (= 5)$

$d_3 = d_3 + 1 = 0 + 1 = 1$

$n = n - c_3 = 7 - 5 = 2$.

Now $n \not> c_3$.

$d_4 = 0$

As $2 > c_4 (= 1)$

$d_4 = d_4 + 1 = 0 + 1 = 1$

$n = n - c_4 = 2 - 1 = 1$.

As $i = 1$

$d_4 = d_4 + 1$

$= 1 + 1$

$= 2$

$n = n - c_4 = 1 - 1 = 0$

Now $n \not> c_4$.

{ $c_1 - 2$ $(= d_1)$
$c_2 - 1$ $(= d_2)$
$c_3 - 1$ $(= d_3)$
$c_4 - 2$ $(= d_4)$ }

The Growth of Functions

Big-O Notation: Estimates 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 the input grows.

Let $f, g : \mathbb{Z}/\mathbb{R} \to \mathbb{R}$. We say that $f(x)$ is $O(g(x))$ if there exists witnesses, the constants C and K such that $|f(x)| \leq C|g(x)|$ whenever $x > K$.

Mean that $f(x)$ grows slower than $g(x)$ (or) $g(x)$ dominates $f(x)$.
Big-O gives an upper bound on the growth of a function $f(x)$.

Example ① Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$.

Sol: Let $x > 1 \Rightarrow 1 < x$.
$\Rightarrow x < x^2$.

Also $x > 1 \Rightarrow 1 < x$
$\Rightarrow 1 < x^2$

Adds and $g(x) = x^2$

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2(x^2) + x^2$$

$$\Rightarrow |x^2 + 2x + 1| \leq 4|x^2|$$

$$\Rightarrow |f(x)| \leq C|g(x)| \text{ where } C = 4 \text{ and } K = 1.$$

Examples

② $7x^2$ is $O(x^3)$ for $C = 1$ and $K = 7$

③ x^2 is not $O(x)$

④ x^3 is not $O(7x^2)$

(5) $f(x) = a_n x^n + a_{n-1} x^{n-1} + a_2 x^{n-2} + \cdots + a_1 x + a_0$, i.e., $f(x)$ is a polynomial of degree n. Then $f(x)$ is $O(x^n)$ for $k=1$, $c = |a_n| + |a_{n-1}| + \cdots + |a_0|$

(6) $1 + 2 + 3 + \cdots + n$ is $O(n^2)$. (For $c=1$, $k=1$)

(7) $n!$ is $O(n^n)$ for $c=1$, $k=1$

(8) $\log n!$ is $O(n \log n)$ for $c=1$ and $k=1$

(9) n is $O(2^n)$ for $c=1$, $k=1$

(10) $\log n$ is $O(n)$ for $c=1$, $k=1$

(11) n^c is $O(n^d)$ for $d > c > 1$

(12) n^d is not $O(n^c)$ for $d > c > 1$

(13) n^d is $O(b^n)$ for $b > 1$ and d positive

(14) b^n is not $O(n^d)$

(15) b^n is $O(c^n)$ for $c > b > 1$

(16) c^n is not $O(b^n)$ for $c > b > 1$

(17) If $f_1(x)$ is $O(g_1(x))$ and $f_2(x)$ is $O(g_2(x))$ then

(i) $(f_1 + f_2)(x)$ is $O(\max(|g_1(x)|, |g_2(x)|))$

(ii) $(f_1 f_2)(x)$ is $O(g_1(x)\, g_2(x))$

(18) If $f_1(x)$ is $O(g(x))$ and $f_2(x)$ is $O(g(x))$ then $(f_1 + f_2)(x)$ is $O(g(x))$.

Big - OMEGA Notation :-

Big - Ω gives an lower bound on the growth of a function $f(x)$.

Let $f, g : \mathbb{Z}/\mathbb{R} \to \mathbb{R}$. We say that $f(x)$ is $\Omega g(x)$ if there exists constants c and k such that

$$|f(x)| \geq c\,|g(x)| \quad \text{whenever } x > k.$$

NOTE If $f(x)$ is $\Omega g(x)$, then $g(x)$ is $O f(x)$

Example Show that $f(x) = 8x^3 + 5x^2 + 7$ is $\Omega(x^3)$

Solution Here $f(x) = 8x^3 + 5x^2 + 7$, $g(x) = x^3$.

Let $x > 0$

$\Rightarrow x^2 > 0$.

Now $8x^3 + 5x^2 + 7 \geq 8x^3$.

$\Rightarrow |f(x)| \geq c\,|g(x)|$ where $c = 8$.

$\Rightarrow 8x^3 + 5x^2 + 7$ is $\Omega(x^3)$

(OR) we can also say that x^3 is $O(8x^3 + 5x^2 + 7)$

BIG - THETA Notation

Let $f, g : \mathbb{Z}/\mathbb{R} \to \mathbb{R}$. If $f(x)$ is $O(g(x))$ and $f(x)$ is $\Omega(g(x))$ then $f(x)$ is $\Theta(g(x))$

$f(x)$ is $\Theta(g(x))$ means.

$$c_1\,|g(x)| \leq f(x) \leq c_2\,|g(x)| \quad \text{whenever } x > k.$$

and c_1, c_2 are constants.

Example. Show that $3x^2 + 8x \log x$ is $\Theta(x^2)$.

Sol We have to show that $3x^2 + 8x \log x$ is $O(x^2)$
and $3x^2 + 8x \log x$ is $\Omega(x^2)$

Let $x > 1$
$\Rightarrow \log x \leq x$.

$\because 0 \leq 3x^2 + 8x \log x \leq 3x^2 + 8x(x)$
$\leq 11x^2$

$\Rightarrow f(x) \leq c\, g(x)$ where $c = 11$ and $k = 1$

$\Rightarrow f(x)$ is $O(x^2)$.

When $x > 1$
$x^2 \leq 3x^2 + 8x \log x$.
$\Rightarrow x^2$ is $O(3x^2 + 8x \log x)$
$\Rightarrow 3x^2 + 8x \log x$ is $\Omega(x^2)$
Consequently, $3x^2 + 8x \log x$ is $\Theta(x^2)$.

COMPLEXITY OF ALGORITHMS:

Computational complexity

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.
Operations → Comparison, addition, multiplication, division of integers or any other basic operations.

Examples

1) Finding the Maximum Element in a Finite sequence

Procedure. max { $a_1, a_2, \ldots, a_n$: Integers }

Max = a_1

for $i = 2$ to n.

if $max < a_i$ then $max = a_i$ }

Return max.

This algorithm as time Complexity $\theta(n)$.

WORST-CASE Complexity:- Largest number of operations needed to solve the given problem.

AVERAGE-CASE Complexity:- Average number of operations needed to solve the given problem.

Examples

1) Linear search algorithm → Worst Case → $\theta(n)$
2) Binary search algorithm → Worst case → $\theta(\log n)$
3) Linear search algorithm → Average Case → $\theta(n)$
4) Bubble sort algorithm → Worst-Case → $\theta(n^2)$
5) Insertion sort algorithm → Worst Case → $\theta(n^2)$

—