13
09
2019

ASSIGNMENT 01

Q:1 Write a program to insert a new element in the given unsorted array at K^{th} Position.

Ans:

```
#include <stdio.h>
#include <conio.h>

void main()
{ int array[50], i, size, num, pos;
  printf("Enter no of elements to be entered : ");
  scanf("%d", &size);
  printf("Enter the elements : ");
  for (i=0; i<size; i++)
  { scanf("%d", &array[i]);
  printf("Enter the element to insert : ");
  scanf("%d", &num);
  printf("Enter the elements Position : ");
  scanf("%d", &pos);
  if (pos > size+1 || pos <= 0)
  { printf("Incorrect Position");
  }
  else
  { for (i=size; i>=pos; i++)
    { array[i] = array[i-1];
    }
    array[pos-1] = num;
    size++;
    printf("Array element after Insertion : ");
    for (i=0; i<size; i++)
    { printf("%d ", array[i]);
    }
  } getch();
}
```

Q:2 What do you understand by complexity of a Algorithm ? Explain Big OH, Omega and Theta notations with examples.

Ans: The complexity of an algorithm is a way to classify how efficient an algorithm is compared to alternative ones. the focus is on how execution time increases with the data set to be processed. The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structure.

* **BIG OH (O)**

Big Oh notation specially describes worst case scenario. It represents the upper bound running time complexity of an algorithm.

$$f(n) \leq cg(n)$$
$$\text{then } f(n) = O(g(n))$$
$$\text{for all } n \geq n_0 \text{ , } c>0$$

* **OMEGA (Ω) :**

It specially describes best case scenario. It represents the lower bound running time complexity of an algorithm. so if we represent a complexity of an algorithm in Omega notation, it means that the algorithm cannot be completed in less time than this. It would atleast take the time represent by Omega Notation or it can take more time when not in best case.

$$f(n) \geq cg(n) \text{ then}$$
$$f(n) = \Omega\, cg(n)$$
$$\text{for all } n \geq n_0 \text{ and}$$
$$c>0$$

* **THETA (Θ) :**

This notation describes both upper band & lower band so we can say that it defines exact asymtotic notation. In the real case scenario the algorithm not always run on best & worst cases the average running time lies in worst and best case.

$$c_1 (g(n)) \leq f(n) \leq c_2 (g(n))$$

for all $n \geq 0$ & $c > 0$

Q:3. Write a Algorithm FIND (DATA, N, LOC1, LOC2) which finds the location LOC1 of the largest element and the location LOC2 of the second largest element in an array DATA with n>1 elements.

Ans: **FIND** (DATA, N, LOC1, LOC2)

step 1 : declare I, First, second, temp.
step 2 : set First = DATA [1], Second = DATA [2], LOC1 = 1, LOC2 = 2 ;
step 3 : if (First < Second)
then Temp = First ; First = Second ; Second = Temp
step 4 : LOC1 = 2 ; LOC2 = 1
step 5 : Repeat step 6 for I = 3 to N
step 6 : if (First < DATA [I])
then set second = First , First = DATA [I] ;
set LOC2 = LOC1 , LOC1 = I ;
else if Second < DATA [I]
then set Second = DATA [I] and LOC2 = I
step 7 : write First = LOC1 , Second : LOC2
step 8 : Exit

Q:4 What is Data structure ? Differentiate data and information. Also explain types of Data structure with example.

Ans: Data structure is a logical or mathematical way to organise the data in memory.

DATA	INFORMATION
- Data can be any character, number, images, text.	organised and presented in a context to make it useful.
- Data alone can never be significant	It is always important by itself.
- It is based on records, observations etc.	It is based on analysis of data.

* TYPES OF DATA STRUCTURES:

i) **ARRAY** : It consists of a collection of elements, each identified by a common variable name & an index. A block of continous memory is assigned for an array. It can be 1-D or 2-D or multidimensional.

ii) **LINKED LIST** : It consists of group of nodes together which represent a sequence. It consists of 2 main parts : the data and a pointer pointing to the next node in the list. The start is marked by head pointer & the end is denoted by NULL pointer.

iii) **TREE** : It has a set of node which have data & pointer points to the children node of the node. The constraint is the no reference is duplicated and no node points to the root.

iv) **GRAPH** : It is a collection of a set of variables vertices V & a set of edges E. Each edge is a pair (V, W) where V and W are element of V i.e. they are vertices.

Q:5 Explain different Operations can be performed on Data structure :

Ans: **INSERTION** : Insertion means addition of a new data element in a data structure. In this we insert an element at any position.

DELETION : It means removal of data elements from a data structure if it is found.

SEARCHING : Searching involves searching for the specified data element in data structure.

TRAVERSAL : Traversal of a data structure means processing all the data elements present in it.

SORTING : Arranging data elements of a data structure in a specified order.

MERGING : Combining elements of two similar data structure to form a new data structure of the same type.

Q:6 Write a Algorithm to multiply 2 matrices of size N * N each. Also prove that complexity of this algorithm is $O(N^3)$

Ans:

Multiplication (A, B, N, N)

step 1 : Read A and B.
step 2 : Declare C[N][N], i, j, k
step 3 : Repeat step 4 to 7 for i = 0 to n-1
step 4 : Repeat step 5 to 7 for j = 0 to n-1
step 5 : set C[I][j] = 0
step 6 : Repeat step 7 for k = 0 to n-1
step 7 : set C[i][j] = C[i][j] + A[i][j] + A[i][k] * B[k][j] ;
step 8 : write C[][]
step 9 : exit.

- **COMPLEXITY** :

Total Time : $C_1 + C_2 + C_3(n+1) + C_4(n(n+1)) + C_5 n^2 + C_6 n^2(n+1) + C_7 n^3 + C_8 + C_9$

⇒ putting 1 at C_1 to C_9

⇒ $1 + 1 + (n+1) + n^2 + n + n^2 + n^3 + n^2 + n^3 + 1 + 1$

⇒ $2n^3 + 3n^2 + 2n + 5$

Hence, Time Complexity of 2 Matrix Multiplication is proved $O(N^3)$.

13
09
2019

ASSIGNMENT 02

Q:1 What do you mean by searching ? Explain following searching methods :

I) LINEAR SEARCH II) BINARY SEARCH

Ans: Searching is the process of finding a given value position in a list of values. It decides whether a search key is present in the data or not.

I) **LINEAR SEARCH** : It is the most basic type of searching algorithm. A linear search sequentially moves through our collection looping for a matching value ---n complexity term this is an O(n) search, the time taken to search the list at the same time rate as the list.

- Time Complexity :

$$f(n) = 1\left(\frac{1}{n}\right) + 2\left(\frac{1}{n}\right) + \cdots n\left(\frac{1}{n}\right) \cdot 0 = (1+2\cdots+n)\frac{1}{n}$$

$$\Rightarrow \frac{n(n+1)}{2} \cdot \frac{1}{n} \Rightarrow \frac{n+1}{2} \Rightarrow \underline{\underline{O(n)}}.$$

II) **BINARY SEARCH** : It is also known as half interval search, logarithmic search, or binary chop, is a search algorithm that finds the position of a target value within a sorted array. It compares the target value to the middle element in the array.

Q.2 Write a program to perform selection sort method. Also explain with an example.

Ans:

```
#include <stdio.h>
#include <conio.h>

void main()
{ int i, j, temp, min, pos, n, a[10];
  printf("Enter no. of elements in Array : ");
  scanf("%d", &n);
  printf("Enter the Numbers : ");
  for(i=0; i<n; i++)
  { scanf("%d", &a[i]); }
  for(i=0; i<n-1; i++)
  { min = a[i];
    pos = i;
    for(j=i+1; j<=n-1; j++)
    { if(min > a[j])
      { min = a[j];
        pos = j;
      } }
    temp = a[pos];
    a[pos] = a[i];
    a[i] = temp;
  }
  printf("Array after sorting is : ");
  for(i=0; i<n; i++)
  { printf("%d", a[i]);
  }
  getch();
}
```

Q:3 Write a Algorithm to find out the sum of elements in given array where size of array is N. Prove that time complexity of this algorithm is O(N).

Ans:

```
Array - Sum ( A , N)
// A is array & N is no of elements

step 1 : declare i , sum
step 2 : set sum = 0
step 3 : Repeat step 4 for i = 0 to n-1
step 4 : set sum = sum + a[i]
step 5 : write A[]
step 6 : exit
```

- TIME COMPLEXITY :

» $c_1 + c_2 + c_3(n+1) + c_4(n) + c_5 + c_6$

» $1 + 1 + 1(n+1) + 1(n) + 1 + 1$

» $1 + 1 + n + 1 + n + 1 + 1$

» $2n + 5$

Q:4 Write an Algorithm of Bubble sort Method. Also prove that complexity of this algorithm is $O(N^2)$ when N is number of element in array.

Ans:

```
Bubble - Sort ( A , N)
// here A is array and N is no of elements

step 1 : declare i , j , temp , n ;
step 2 : Repeat step 3 to 4 for i = 0 to n-1
step 3 : Repeat step 4 for j = 0 to j < n-1-i
step 4 : if a[j] > a[j+1]
             then temp = a[j]
                  a[j] = a[j+1]
                  a[j+1] = temp
step 5 : write a[]          step 6 : exit
```

- TIME COMPLEXITY :

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

$\Rightarrow \frac{n(n-1)}{2} \Rightarrow \frac{n^2}{2} + O(n)$

$\Rightarrow n + O(n)$

$\Rightarrow \underline{\underline{O(n^2)}}$ Proved.

Q:5 Write an algorithm to perform Insertion sort Algorithm. Also prove that time complexity of this algorithm in Best Case $O(n)$ and in worst case $O(n^2)$. where n is number of elements in array.

Ans:

```
Insertion_sort (a, n)
// here A is array & n is no of elements in array.
step 1 : declare i, j, key
step 2 : Repeat step 3 to 6 for i=0 to i<n
step 3 : set key = a[i]
         j = i - 1
step 4 : Repeat step 5 while j>=0 && a[j]>key
step 5 : set a[j+1] = a[j]
         j = j - 1
step 6 : set a[j+1] = key
step 7 : write a[]
step 8 : exit
```

BEST CASE :

$f(n) = C_1 + C_2(n) + C_3(n-1) + C_4 \sum_{i=1}^{n-1} ti + C_5 \sum_{i=1}^{n-1} ti - 1 +$

$C_6(n-1) + C_7$

$\Rightarrow C_1 + C_2 n + C_3(n-1) + C_4(n-1) + C_6(n-1) + C_7$

$\Rightarrow 1 + n + n - 1 + n - 1 + n - 1 + 1$

$\Rightarrow 4n - 1 \Rightarrow \underline{\underline{O(n)}}$

WORST CASE :

$$C_1 + C_2 + C_3(n-1) + C_4\left[\frac{n(n+1)}{2} - 1\right] + C_5\left[\left(\frac{n(n+1)}{2} - 1\right) - 1\right] + C_6(n-1) + C_7$$

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

$$3n + \frac{n^2+n-2}{2} + \frac{n^2+n-2}{2} - 1$$

$$\frac{6n + n^2 + n - 2 + n^2 + n - 2 - 2}{2} \quad \frac{6n + 2n^2 + 2n - 6}{2}$$

$$\frac{2n^2 + 8n - 6}{2} \quad O(n^2)$$

Q:6 Find the Complexity of the below program :

```
function (int n)
{ if (n == 1)
  return ;
  for ( int i = 1 ; i <= n ; i++)
  { for (int j = 1 ; j <= n ; j++)
    { printf (" * "); }
  }
}
```

Ans : TIME COMPLEXITY : ?

- ALGORITHM :

```
function (N)
// here function is a function with N argument.
step 1 : declare i , j
step 2 : if n == 1
         then return
step 3 : Repeat step 4 & 5 for i = 1 to i <= n
step 4 : Repeat step 5 for j = 1 to j <= n
step 5 : write *
```

- TIME COMPLEXITY :

» $C_1 + C_2 + C_3(n+1) + C_4(n+1) + C_5$

» $1 + 1 + 1(n+1) + 1(n+1) + 1$

» $1 + 1 + n + 1 + n + 1 + 1$

» $2n + 5$ » $\underline{\underline{O(n)}}$