Programming For Problem Solving

Unit - 5

Introduction to Algorithms

→ Algorithms for finding roots of a quadratic equations, finding maximum and minimum numbers of a given set, finding if a number is prime number, etc.

→ Basic searching in an array of elements (Linear and binary search techniques)

→ Basic algorithms to sort array of elements (Bubble, Insertion and Selection sort algorithms), Basic concept of order of complexity through the example programs.

Algorithm:- Algorithm is the step-by-step procedure for solving a problem.

Finding the roots of quadratic equation:-

Step-1:- Take input values a, b, c

Step-2:- Calculate $K = b^2 - 4ac$

Step-3:- If $(K > 0)$ then,
display "Roots are Real and
calculate root1 $-b + sqrt(2*a)$;
root2 $-b - sqrt(2*a)$;

Step-4:- else if $(K = 0)$ then
displayed the "Roots are equal"
calculate root1 = root2 = $\frac{-b}{2*a}$

Step-5:- else if $(K < 0)$ then
display Roots are imaginary and
calculate $r_1 = \frac{-b + d}{2*a}$, $r_2 = \frac{-b - d}{2*a}$

Step-6:- End the algorithm.

Algorithm for finding minimum and maximum numbers from a given set of elements (or) array of elements:-

The following steps are used to find maximum and minimum value of a set of elements.

Step-1 :- Start

Step-2 :- Input set = (val1, val2, ···· valn)

Step-3 :- Maximum = 0

Step-4 :- Minimum = 0

Step-5 :- Read (number) and we can compare

Step-6 :- If number > maximum the maximum = number

Step-7 :- If number < minimum then minimum = number.

Step-8 :- Repeat from step-IV as many times needed

Step-9 :- Print maximum and minimum

Step-10 :- Stop.

Algorithm for finding a number is prime number or not:-

The following is a algorithm for whether a given number is a prime number.

Step-1 :- Start

Step-2 :- Take a integer variable 'A'

Step-3 :- initalize the variable 'A'.

Step-4 :- If 'A' is divisible by any value from 2 to A/2 then it prints the given number is not a prime number.

Step-5 :- else it is a prime number.

Step-6 :- Stop.

Basing Searching in an array of elements:

Searching is a process of finding particular element in the list. If the elementary is present in the list then the elementary list process is called successful and process return the location of that element, otherwise the search is called unsuccessful and the process return element was not found.

There are two popular search methods that are widely used in order to search some item into list. they are:-

(1) Linear Search (2) Binary search.

Linear Search:-

→ Linear Search is the simplest search algorithm and often called sequential search.

→ In this type of searching, we simply transverse the list completely and match each element of list with the item whose location is to be found.

If the match found then location of item is returned otherwise the algorithm return NULL.

→ Linear search is mostly used to search an unordered list in which the items are not sorted.

→ The algorithm of Linear Search is given as follows:-

Syntax:- LINEAR_SEARCH (A, N, VAL)

Step-1:- [INITIALIZE] SET POS = -1

Step-2:- [INITIALIZE] SET I = 1

Step-3:- Repeat step 4 while I <= N

Step-4:- IF A[I] = VAL
SET POS = I
PRINT POS
Go to Step-6
SET I = I+1

Step-5:- IF POS = -1
PRINT "VALUE IS NOT PRESENTING ARRAY"

step-6:- EXIT

Example program:-

```
1. #include<stdio.h>
2. void main()
3. {
4.  int a[10]={10,20,30,4,2,0,14,13,50,9};
5.  int item,i,flag;
6.  printf("\n Enter Item which is to be searched\n");
7.  scanf("%d",&item);
8.  for(i=0;i<10;i++)
9.  {
10.   if(a[i]==item)
11.   {
12.    flag=i+1;
13.    break;
14.   }
15.   else
16.   flag=0;
17.  }
18.   if(flag!=0)
19.  {
20.   printf("\n Item found at location %d\n",flag);
21.  }
22.   else
23.  {
24.   printf("\nItem not found\n");
25.  }
26. }
```

Output:-

```
Enter item which is to be searched
20
Item not found
Enter item which is to be searched
23
Item found at location 2.
```

Binary Search:-

Binary search is the search technique which works efficiently on the sorted lists. Hence, in order to search an element into some list by using binary search technique, we must ensure that the list is sorted.

Binary search follows divide and conquer approach in which, the list is divided into two halves and the item is compared with the middle element of the list. If the match is found then, the location of middle element is returned otherwise, we search into either of the halves depending upon the result produced through the match.

Syntax:- BINARY_SEARCH (A, lower_bond, upper_bond, VAL)

Binary search algorithm is given below:-

Step-1 :- [INITIALIZE] SET BEG = Lower_bound

END = upper_bond, POS = -1

Step-2 :- Repeat steps 3 and 4 while BEG <= END

Step-3 :- SET MID = (BEG + END)/2

step-4 :- IF A[MID] = VAL

SET POS = MID

PRINT POS

go to step.6

ELSE IF A[MID] > VAL

SET END = MID - 1

ELSE

SET BEG = MID + 1

[END OF IF]

[END OF LOOP]

step 5 :- IF POS = -1

PRINT "VALUE IS NOT PRESENT IN THE ARRAY"

[END OF IF]

step 6 :- EXIT

Example program :-

```
1. #include<stdio.h>
2. int binarysearch(int[], int, int, int);
3. void main()
4. {
5.    int arr[10] = {16, 19, 20, 23, 45, 56, 78, 90, 96, 100};
```

```
6.  int item, location=-1;
7.  printf("Enter the item which you want to search");
8.  scanf("%d", &item);
9.  location = binary search(arr, 0,9, item);
10. if(location != -1)
11. {
12.   printf("Item found at location %d", location);
13. }
14. else
15. {
16.   printf("Item not found");
17. }
18.}
19. int binary search(int a[], int beg, int end, int item)
20. {
21.   int mid;
22.   if(end >= beg)
23.   {
24.     mid = (beg + end)/2;
25.     if(a[mid] < item)
26.     {
27.       return mid + 1;
28.     }
        else if(a[mid] < item)
29.     {
30.     else return binary search(a, mid+1, end, item);
31.     }
32.
```

```
33. else
34. {
35. return binary search(a, beg, mid-1, item);
36. }
37.
38. }
39. return -1;
40. }
```

Output:-

Enter the item which you want to search
19.
item found at location 2.

Basic algorithms to sort array of elements:-

Basic algorithms to sort are of three types: they are:-

1) Bubble sort algorithm
2) Insertion sort algorithm
3) selection sort algorithm.

Bubble Sort:-

In Bubble sort, Each element of the array is compared with its adjacent element.
The algorithm processes the list in passes. A list

with n elements requires n-1 passes for sorting. consider an array A of n elements whose elements are to be sorted by bubble sort. The algorithm processes like following.

1. In Pass 1, A[0] is compared with A[1], A[1] is compared with A[2], A[2] is compared with A[3] and so on. At the end of pass 1, the largest element of the list is placed at the highest index of the list.

2. In pass 2, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of pass 2 the second largest element of the list is placed at the second highest index of the list.

3. In pass n-1, A[0] is compared with A[1], A[1] is compared with A[2] and so on. At the end of this pass. The smallest element of the list is placed at the first index of the list.

Algorithm:-

step-1:- Repeat step 2 For i=0 to N-1

step-2:- Repeat for J = i+1 to N-I

step-3:- IF A[J] > A[i]

SWAP A[J] and A[i]

[END OF INNER LOOP]

[END OF OUTER LOOP]

step-4:- EXIT

Example program:-

```
1. #include<stdio.h>
2. void main()
3. {
4.    int i,j,temp;
5.    int a[10]={10,9,7,101,23,44,12,7,8,34,23};
6.    for (i=0;i<10;i++)
7.    {
8.       for (j=i+1;j<10;j++)
9.       {
10.        if (a[j]>a[i])
11.        {
12.          temp=a[i];
13.          a[i]=a[j];
14.          a[j]=temp;
15.        }
16.      }
17.   }
18.   printf("Printing Sorted Element List...\n");
19.   for (i=0;i<10;i++)
20.   {
21.   printf("%d\n",a[i]);
22.   }
23. }
```

Output:- Printing Sorted Element List

7, 9, 10, 12, 23, 34, 34, 44, 78, 101

Insertion Sort:-

Insertion sort is the simple sorting algorithm which is commonly used in the daily lives while ordering a deck of cards. In this algorithm, we insert each element onto its proper place in the sorted array. This is less efficient that the order sort algorithms like quick sort, merge sort, etc.

Technique:-

consider an array A whose elements are to be sorted. Initially, A[0] is the only element on the sorted set. In pass 1, A[1] is placed at its proper index in the array.

In pass 2, A[2] is placed at its proper index in the array. Likewise, in pass n-1, A[n-1] is placed at its proper index into the array.

To insert an element A[K] to its proper index, we must compare it with all other elements i.e. A[K-1], A[K-2], and so on until we find an element A[j] such that, A[j] <= A[K]

All the elements from A[K-1] to A[j] need to be shifted and A[K] will be moved to A[j+1].

Alogorithm:-

step-1 :- Repeat steps 2 to 5 for K=1 to N-1

step-2 :- SET TEMP = ARR[K]

step-3 :- SET J = K-1

step-4 :- Repeat while TEMP<=ARR[J]

SET ARR[J+1] = ARR[J]

SET J = J-1

[END OF INNER LOOP]

step-5 :- SET ARR[J+1]=TEMP

[END OF LOOP]

step-6 :- EXIT.

Program:-

```
1. #include<stdio.h>
2. void main()
3. {
4.    int i,j,k,temp;
5.    int a[10]={10,9,7,101,23,44,12,7,8,34,23};
6.    printf("\n printing sorted elements ....\n");
7.    for (k=1;k<10;k++)
8.    {
9.       temp=a[k];
10.      j=k-1;
11.      while (j>0 && temp<=a[j])
12.      {
13.         a[j+1]=a[j];
```

```
14.  j=j-1;
15. }
16.  a[j+I]=temp;
17. }
18.  for(i=0; i<10;i++)
19.  {
20    printf("\n%d\n"a[i]);
21.  }
22. }
```

Output:-

Printing sorted Elements...

7
9
10
12
23
23
34
44
78
101.

Selection Sort:-

In selection sort, the smallest value among the unsorted elements of the array is selected in every pass and inserted to its appropriate position into the array.

First, find the smallest value among element of the array and place it on the first position. Then, find the second smallest element of the array and place it on the second position. The process continues until we get the sorted array.

The array with n elements is sorted by using n-1 pass of selection sort algorithm.

→ In 1st pass, smallest element of the array is to be found along with its index pos. then, swap A[0] and A[pos]. Thus A[0] and A[1] are sorted, we now have n-1 elements which are to be sorted.

→ In 2nd pas, position pos of the smallest element present in the sub-array A[n-1] is found. Then, swap, A[1] and A[pos]. Thus A[0] and A[1] are sorted, we now left with n-2 unsorted elements.

→ In n-1 th pass, position pos of the smaller element between A[n-1] and A[n-2] is to be found. Then, swap, A[pos] and A[n-1].

Therefore, by following the above explained process, the elements A[0], A[1], A[2], . . ., A[n-1] are sorted.

Ex:-

consider the following array with 6 elements. sort the elements of the array by using selection sort.

A = {10, 2, 3, 90, 43, 56}

pass	POS	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
1	1	2	10	3	90	43	56
2	2	2	3	10	90	43	56
3	2	2	3	10	90	43	56
4	4	2	3	10	43	90	56
5	5	2	3	10	43	56	90

Algorithms:-

SELECTION SORT(ARR,N)

step1:- Repeat steps 2 and 3 for K=1 to N-1

step 2:- CALL SMALLEST (ARR, K, N, POS)

[END OF LOOP]

step-3:- SWAP A[K] with ARR[POS]

[END OF LOOP]

step-4:- EXIT

SMALLEST (ARR K,N, POS)

step-1:- [INITIALIZE] SET SMALL = ARR[k]

step-2:- [INITALIZE] SET POS = K

step-3:- REPEAT for J = K+1 to N-1.

IF SMALL > ARR[J]

SET SMALL = ARR[J]

SET POS = J

[END OF IF]

[END OF LOOP]

Step:-4 :- RETURN POS

Example program:-

```
1.  #include<stdio.h>
2.  int smallest (int [],int,int);
3.  void main ()
4.  {
5.      int a[10]= {10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
6.      int i,j,k, pos, temp;
7.      for (i=0; i<10; i++)
8.      {
9.        pos = smallest (a, 10, i);
10.       temp = a[i];
11.       a[i]= a[pos];
12.       a[pos]= temp;
13.     }
14.     printf("\n printing sorted elements...\n");
15.     for (i=0; i<10; i++)
16.     {
17.       printf("%d\n", a[i]);
18.     }
19.  }
20.  int smallest (int a[], int n, int i)
21.  {
```

```
22.  int small, pos, j;
23.  small = a [i];
24.  pos = i;
25.  for(j = i+1; j<10; j++)
26.  {
27.     if (a[j] < small)
28.     {
29.       small = a [j];
30.       pos = j;
30.    }
31.  }
32.  return pos;
33. }
```

Output:-

printing sorted elements...

7
9
10
12
23
23
34
44
78
101.