Page No.
Date

Assignment No. 5

* Implement trees and various traversing techniques.

Implementing Trees :

Generally the binary tree creation is application dependent. Creation of an empty binary search tree means creating a tree with zero nodes.

For a binary tree, the declaration required is of following type -

```
type def struct btype
{
    struct btype * left ;
    int info ;
    struct btype * right ;
} bnode ;
bnode * root ;
```

Once the data structure for binary tree is declared, you can create an empty tree using that structure by declaring it as null. i.e. you need a statement like

```
bnode * tree ;
    tree = (bnode *) NULL ;
```

The above declaration initializes the tree structure to zero by declaring the external pointer tree, containing address of root node, as null.

Page No.
Date

Various Traversing Techniques :

The most commonly used operations on a tree is the tree traversal. Traversal means visiting or processing all the nodes of the tree once and only once. As a binary tree is by definition a tree containing root, left subtree and right subtree so during traversal we can process the root first or left or right subtree first.
Based on the order in which the root node is processed, the tree traversal can be classified in three types :-

- Preorder Traversal
- In order Traversal
- Post order Traversal

i) Preorder Traversal -

The preorder traversal of a binary tree may be defined recursively as

- Process the root node
- Traverse the left subtree in Preorder
- Traverse the right subtree in Preorder

ii) Inorder Traversal -

The inorder traversal of a binary tree may be defined recursively as

- Traverse the left subtree in inorder
- Process the root node

- Traverse the right subtree in order.

iii) **Post Order Traversal** -

The post order traversal of a binary tree may be defined recursively as

- Traverse the left subtree in postorder
- Traverse the right subtree in post order
- Process the root code

Page No.
Date

Assignment No. 6

* Implement various searching and sorting algorithms and to compare them for checking efficiency.

⇒ **SEARCHING :**

Finding one particular piece of information among large amount of data is known as searching.

TYPES OF SEARCH :

1. Internal search techniques
   - Sequential Search or Linear Search
   - Binary Search
   - Indexed Sequential Search
   - Interpolation Search
   - Tree Search
2. External search techniques

→ Here we will discuss only Binary search and Linear search.

- Linear Search or Sequential Search

```
/* Program Seqeential Search */
# include <stdio.h>
 main ()
 {
```

```
int rolls[100]; /* table of marks */
int flg, x, i;
/* Read data table in memory */
for (i=1; i<=100; i++)
{ printf ("\n Enter Marks :");
  scanf ("%d", &rolls[i]);
}
printf ("\n Enter the marks to search ");
scanf ("%d", &X);
flg = 0;
for(i=1; i<=n; i++)
{
  if (rolls[i] == x)
  {
    printf ("\n The element is found at %d
            position ", i);
    flg = 1;
    break;
  }
}
if (flg == 0)
  printf ("\n The element is not present in
    the table ");
}
```

- Binary Search

```
/* Program for BINARY SEARCH using array */

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

```
void main()
{
  int a[100];
  int ead, mid, start, last;
  int i, j, n, t, elem;
  int flag = 0;
  printf("\n Enter number of elements in array");
  scanf("%d", &n);
  for(i=0; i<=n-1; i++)
  {
    printf("\n Enter %d th element", i);
    scanf("%d", &a[i]);
  }
  /* sort the list in ascending order */
  for(i=0; i<=n-2; i++)
  {
    for(j=i+1; j<=n-1; j++)
    {
      if(a[i] > a[j])
      {
        t = a[i];
        a[i] = a[j];
        a[j] = t;
      }
    }
  }
  printf("\n Enter the element to be searched");
  scanf("%d", &elem);
  mid = 0;
  start = 0;
  last = n-1;
```

```
/* start searching the element */
while ((start <= last) && (elem != a[mid]))
{
  mid = ((start + last) / 2);
  if (elem == a[mid])
  {
    printf ("\n The search is successfull");
    ead = mid;
    printf ("\n the element is found at
%d th position ", ead + 1);
    flag = 1;
  }
  if (elem < a[mid])
    last = mid - 1;
  else
    start = mid + 1;
}
  if (flag == 0)
{
  printf ("\n Search unsuccessfull. Element
not found in list.");
  printf ("\n");
}
```

SORTING :

Arranging a data table in ascending or descending order of key is known as sorting.

Page No.
Date

TYPES OF SORTING :

1. Internal Sorting
   - Selection Sort
   - Exchange Selection Sort
   - Insertion Sort
   - Bubble Sort
   - Quick Sort
   - Heap Sort
   - Tree Sort
   - Shell Sort
   - Merge Sort
   - Radix Sort
   - Tournament Sort

2. External Sorting

- SELECTION SORT

```
/* Program for Simple Selection Sort */
                /SELSORT.C */

#include <stdio.h>
#include <conio.h>
#define maxn 200
/* maximum size of array */

main()
{
  int n, j, i, k, min, max;
```

```
int a[maxn], b[maxn];
/* a is unsorted list and b is sorted list */
printf("\n Enter the elements in the list");
scanf("%d", &max);
/* read array elements */
for( i=0; i<max; i++)
{
    printf("\n enter %d th element : ", (i+1));
    scanf("%d", &a[i]);
}
/* start sorting the list */

m=max;
for(j=0; j<max; j++)
{
    min=99999;
    /* find smallest element */
    for(i=0; i<m; i++)
    {
        if (a[i]<min)
        { min = a[i];
          k=i;
        }
    }
    b[j] = min;
    /* remove the element entered in sorted list */

    for(i=k; i<m-1; i++)
      a[i] = a[i+1];
    for(i=m; i<max; i++)
      a[i]=0;
```

```
    -m;
}
/* print the sorted list */

for (i=0; i<=max; i++)
    printf("\n %d \t %d", i, b[i]);
}
```

- INSERTION SORT

```
/* Program for Insertion Sort */
/* INSSORT.C */

# include <stdio.h>
# include <conio.h>
# define maxsize 200
/* declared maximum size for the array */

main ()
{
    int i, j, k, max;
    int a[maxsize];

    printf ("\n enter number of elements in the
            list");
    scanf ("%d", &max);

    /* read array elements */

    for (i=0; i<max; i++)
    {
```

```
printf("\n enter %d the element :", (i+1));
scanf("%d", &a[i]);
}

/* start insertion sort */

for(i=1; i<max; i++)
{
temp = a[j]; j=i-1;
while((temp<a[j] && (j>=0))
{
a[j+1] > a[j];
j--;
}
a[j+1] = temp;
}
printf("Sorted numbers are *\n");
for(i=0; i<n; i++)
printf("%d", a[i]);
}
```

- CHART OF COMPLEXITY OF ALGORITHMS

Searching Algorithms

Algorithm	Worst case	Average case	Best case
Linear Search	O(n)	$\frac{n+1}{2}$	
Binary Search	O(log n)		

Page No.
Date

Sorting Algorithms

Algorithm	Worst Case	Average Case	Best case
Bubble Sort	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = dn^2$
Quick Sort	$\frac{n(n+3)}{2} = O(n^2)$	$4n \log n = O(n \log n)$	$O(n \log n)$
Heap Sort	$3n \log n = O(n \log n)$	$3n \log n = O(n \log n)$	$O(n \log n)$
Insertion Sort	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{4} = O(n^2)$	$O(n)$
Selecting Sort	$\frac{n(n-1)}{2} = O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$	$O(n^2)$
Merge Sort	$n \log n = O(n \log n)$	$n \log n = O(n \log n)$	$O(n \log n)$
Radix Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$