Syeda Reeha Quasar

14114902719

3C7

Aim

To implement Insertion sort and Merge sort techniques using array.

Experiment - 11

Data Structures

# **EXPERIMENT – 11**

**AIM:** To implement Insertion sort and Merge sort techniques using array.

# **THEORY**

#### **Insertion Sort**

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

**Algorithm**  
To sort an array of size n in ascending order:  
1: Iterate from arr[1] to arr[n] over the array.  
2: Compare the current element (key) to its predecessor.  
3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

**Example:**  
[](https://media.geeksforgeeks.org/wp-content/uploads/insertionsort.png)

#### **Merge Sort**

It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See the following C implementation for details.

**MergeSort(arr[], l, r)**

If r > l

**1.** Find the middle point to divide the array into two halves:

middle m = (l+r)/2

**2.** Call mergeSort for first half:

Call mergeSort(arr, l, m)

**3.** Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

**4.** Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

The following diagram from [wikipedia](http://en.wikipedia.org/wiki/File:Merge_sort_algorithm_diagram.svg" \t "_blank) shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.



### Merge Sort

## **Source code:**

#include <stdio.h> //lib

// function to sort the given array

void merge\_sort(int i, int j, int givenArr[]) {

if (j <= i) {

return; // the subsection is empty or a single element

}

int aux[100]; // temp arr for storage

int mid = (i + j) / 2;

// left sub-array is givenArr[i .. mid]

// right sub-array is givenArr[mid + 1 .. j]

merge\_sort(i, mid, givenArr); // sort the left sub-array recursively

merge\_sort(mid + 1, j, givenArr); // sort the right sub-array recursively

int pointer\_left = i; // pointer\_left points to the beginning of the left sub-array

int pointer\_right = mid + 1; // pointer\_right points to the beginning of the right sub-array

int k; // k is the loop counter

// we loop from i to j to fill each element of the final merged array

for (k = i; k <= j; k++) {

if (pointer\_left == mid + 1) { // left pointer has reached the limit

aux[k] = givenArr[pointer\_right];

pointer\_right++;

} else if (pointer\_right == j + 1) { // right pointer has reached the limit

aux[k] = givenArr[pointer\_left];

pointer\_left++;

} else if (givenArr[pointer\_left] < givenArr[pointer\_right]) { // pointer left points to smaller element

aux[k] = givenArr[pointer\_left];

pointer\_left++;

} else { // pointer right points to smaller element

aux[k] = givenArr[pointer\_right];

pointer\_right++;

}

}

for (k = i; k <= j; k++) { // copy the elements from aux[] to a[]

givenArr[k] = aux[k];

}

}

int main() {

int arr[100], n, i; // arr and variable declaration

// taking array size

printf("Enter number of elements in the array:\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

// taking arr elements input

for (i = 0; i < n; i++)

scanf("%d", &arr[i]);

merge\_sort(0, n - 1, arr); // merge sorting

printf("\n\nPrinting the sorted array:\n");

// printing sorted array

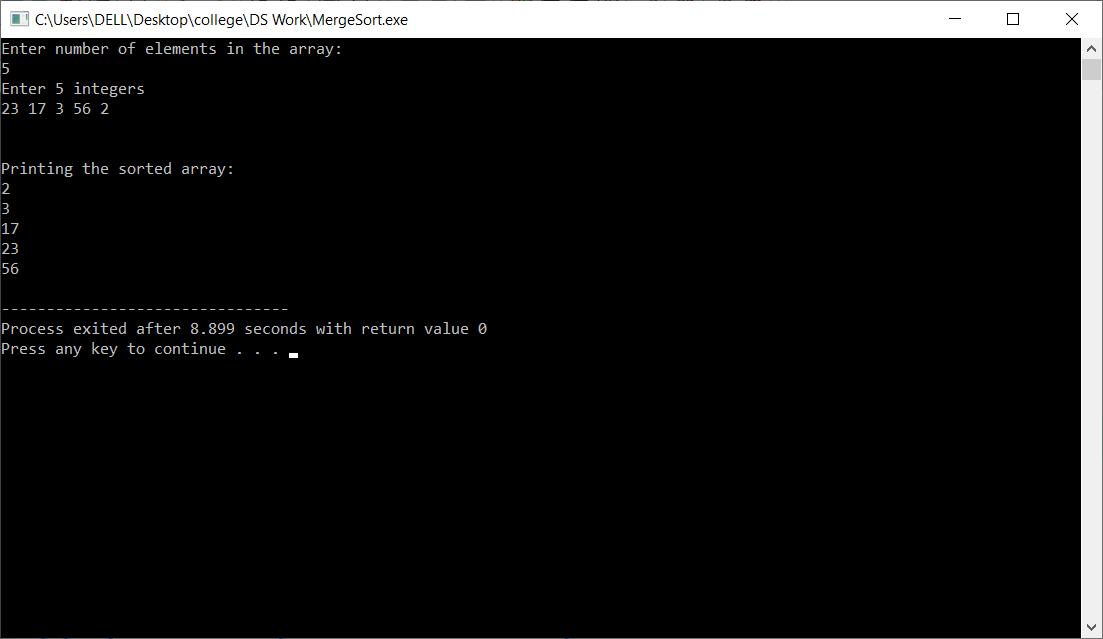
for (i = 0; i < n; i++)

printf("%d\n", arr[i]);

return 0;

}

**OUTPUT**



### Insertion Sort

## **Source code:**

# **VIVA VOICE**

1. Give any application of BST.

Ans.

1. It is used to efficiently store data in sorted form in order to access and search stored elements quickly
2. They can be used to represent arithmetic expressions
3. BST used in Unix kernels for managing a set of virtual memory areas (VMAs).
4. Whether a binary search tree is balanced or not?

Ans.

A non-empty binary tree is height-balanced if:

1. Its left subtree is height-balanced.
2. Its right subtree is height-balanced.
3. The difference between heights of left & right subtree is not greater than 1.
4. Which traversal will generate a ascending order list of nodes in BST?

Ans.

Inorder traversal of BST prints it in ascending order. The only trick is to modify recursion termination condition in standard Inorder Tree Traversal.

1. what is a binary tree?

Ans.

A normal tree has no restrictions on the number of children each node can have. Binary trees, on the other hand, can have at most two children for each parent. Every node contains a 'left' reference, a 'right' reference, and a data element. The topmost node in the tree is called the root node. Nodes with children are parent nodes, and the child nodes contain references to their parents. A node with no children is called a leaf node. Thus, each node in a binary tree can have either 0, 1 or 2 children

1. What are different types of binary trees?

Ans.

There are three different types of binary trees that will be discussed in this lesson:

* Full binary tree: Every node other than leaf nodes has 2 child nodes.
* Complete binary tree: All levels are filled except possibly the last one, and all nodes are filled in as far left as possible.
* Perfect binary tree: All nodes have two children and all leaves are at the same level.

|  |
| --- |
| Types of Binary Trees |
| ***Figure 2: Types of Binary Trees*** |

1. Which of the following is false about a binary search tree?  
   a) The left child is always lesser than its parent  
   b) The right child is always greater than its parent  
   c) The left and right sub-trees should also be binary search trees  
   d) In order sequence gives decreasing order of elements

Ans. Answer: d  
Explanation: In order sequence of binary search trees will always give ascending order of elements. Remaining all are true regarding binary search trees.

1. **What is the speciality about the inorder traversal of a binary search tree?  
   a) It traverses in a non increasing order  
   b) It traverses in an increasing order  
   c) It traverses in a random fashion  
   d) It traverses based on priority of the node**

Ans. Answer: b  
Explanation: As a binary search tree consists of elements lesser than the node to the left and the ones greater than the node to the right, an inorder traversal will give the elements in an increasing order.

1. **What does the following piece of code do?**

**public void func(Tree root)**

**{**

**func(root.left());**

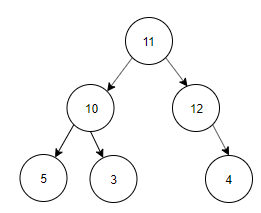
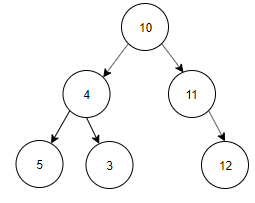
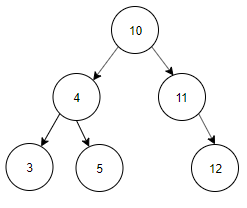
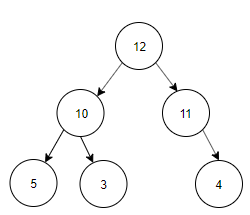
**func(root.right());**

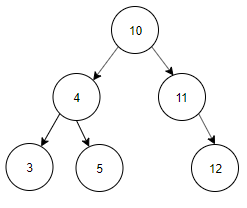
**System.out.println(root.data());**

**}**

1. **Preorder traversal**
2. **Inorder traversal**
3. **Postorder traversal**
4. **Level order traversal**

Ans. Answer: c  
Explanation: In a postorder traversal, first the left child is visited, then the right child and finally the parent.

1. **Construct a binary search tree with the below information.  
   The preorder traversal of a binary search tree 10, 4, 3, 5, 11, 12.**a) [](https://www.sanfoundry.com/wp-content/uploads/2017/08/data-structure-questions-answers-binary-search-tree-q11a.png)  
   b) [](https://www.sanfoundry.com/wp-content/uploads/2017/08/data-structure-questions-answers-binary-search-tree-q11b.png)  
   c) [](https://www.sanfoundry.com/wp-content/uploads/2017/08/data-structure-questions-answers-binary-search-tree-q11c.png)  
   d) [](https://www.sanfoundry.com/wp-content/uploads/2017/08/data-structure-questions-answers-binary-search-tree-q11d.png)

Ans. Answer: c  
Explanation: Preorder Traversal is 10, 4, 3, 5, 11, 12. Inorder Traversal of Binary search tree is equal to ascending order of the nodes of the Tree. Inorder Traversal is 3, 4, 5, 10, 11, 12. The tree constructed using Preorder and Inorder traversal is  
[](https://www.sanfoundry.com/wp-content/uploads/2017/08/data-structure-questions-answers-binary-search-tree-q11c.png)

1. **What are the applications of binary trees?**

Ans.

* Binary Search Tree - Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
* Binary Space Partition - Used in almost every 3D video game to determine what objects need to be rendered.
* Binary Tries - Used in almost every high-bandwidth router for storing router-tables.
* Hash Trees - used in p2p programs and specialized image-signatures in which a hash needs to be verified, but the whole file is not available.
* Heaps - Used in implementing efficient priority-queues, which in turn are used for scheduling processes in many operating systems, Quality-of-Service in routers, and A\* (path-finding algorithm used in AI applications, including robotics and video games). Also used in heap-sort.
* Huffman Coding Tree (Chip Uni) - used in compression algorithms, such as those used by the .jpeg and .mp3 file-formats.
* GGM Trees - Used in cryptographic applications to generate a tree of pseudo-random numbers.
* Syntax Tree - Constructed by compilers and (implicitly) calculators to parse expressions.
* Treap - Randomized data structure used in wireless networking and memory allocation.
* T-tree - Though most databases use some form of B-tree to store data on the drive, databases which keep all (most) their data in memory often use T-trees to do so.