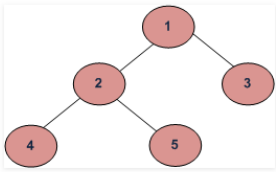
```java
public static void mergeSort(int[] a, int n) {
    if (n < 2) {
        return;
    }
    int mid = n / 2;
    int[] l = new int[mid];
    int[] r = new int[n - mid];

    for (int i = 0; i < mid; i++) {
        l[i] = a[i];
    }
    for (int i = mid; i < n; i++) {
        r[i - mid] = a[i];
    }
    mergeSort(l, mid);
    mergeSort(r, n - mid);

    merge(a, l, r, mid, n - mid);
}
```

```java
protected void rotateLeft(BinaryTreeNode<E> n) {
    if (n.getRight() == null) {
        return;
    }
    BinaryTreeNode<E> oldRight = n.getRight();
    n.setRight(oldRight.getLeft());
    if (n.getParent() == null) {
        root = oldRight;
    } else if (n.getParent().getLeft() == n) {
        n.getParent().setLeft(oldRight);
    } else {
        n.getParent().setRight(oldRight);
    }
    oldRight.setLeft(n);
}
```

Algorithm for Breadth-First Search

1. Take an arbitrary start vertex, mark it identified (color it light blue), and place it in a queue.
2. `while` the queue is not empty
3. Take a vertex, u, out of the queue and visit u.
4. `for` all vertices, v, adjacent to this vertex, u
5. `if` v has not been identified or visited
6. Mark it identified (color it light blue).
7. Insert vertex v into the queue.
8. We are now finished visiting u (color it dark blue).



Example Tree

Depth First Traversals:

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5

```java
// Java program for implementation of Bubble Sort
class BubbleSort
{
    void bubbleSort(int arr[])
    {
        int n = arr.length;
        for (int i = 0; i < n-1; i++)
            for (int j = 0; j < n-i-1; j++)
                if (arr[j] > arr[j+1])
                {
                    // swap temp and arr[i]
                    int temp = arr[j];
                    arr[j] = arr[j+1];
                    arr[j+1] = temp;
                }
    }
```

```java
public void preorderIter(TreeNode root) {

        if(root == null)
            return;

        Stack<TreeNode> stack = new Stack<TreeNode>();
        stack.push(root);

        while(!stack.empty()){

            TreeNode n = stack.pop();
            System.out.printf("%d ",n.data);


            if(n.right != null){
                stack.push(n.right);
            }
            if(n.left != null){
                stack.push(n.left);
            }

        }
```

```java
// Java program for implementation of Selection Sort
class SelectionSort
{
    void sort(int arr[])
    {
        int n = arr.length;

        // One by one move boundary of unsorted subarray
        for (int i = 0; i < n-1; i++)
        {
            // Find the minimum element in unsorted array
            int min_idx = i;
            for (int j = i+1; j < n; j++)
                if (arr[j] < arr[min_idx])
                    min_idx = j;

            // Swap the found minimum element with the first
            // element
            int temp = arr[min_idx];
            arr[min_idx] = arr[i];
            arr[i] = temp;
        }
    }
```

```java
public static void postorderIterative(TreeNode root)
{
    // create an empty stack and push root node
    Stack<TreeNode> stack = new Stack();
    stack.push(root);

    // create another stack to store post-order traversal
    Stack<Integer> out = new Stack();

    // run till stack is not empty
    while (!stack.empty())
    {
        // we pop a node from the stack and push the data to output stack
        TreeNode curr = stack.pop();
        out.push(curr.data);

        // push left and right child of popped node to the stack
        if (curr.left != null) {
            stack.push(curr.left);
        }

        if (curr.right != null) {
            stack.push(curr.right);
        }
```

```java
    private int partition(int arr[], int begin, int end) {
        int pivot = arr[end];
        int i = (begin-1);

        for (int j = begin; j < end; j++) {
            if (arr[j] <= pivot) {
                i++;

                int swapTemp = arr[i];
                arr[i] = arr[j];
                arr[j] = swapTemp;
            }
        }

        int swapTemp = arr[i+1];
        arr[i+1] = arr[end];
        arr[end] = swapTemp;

        return i+1;
    }
```

```java
public void quickSort(int arr[], int begin, int end) {
    if (begin < end) {
        int partitionIndex = partition(arr, begin, end);

        quickSort(arr, begin, partitionIndex-1);
        quickSort(arr, partitionIndex+1, end);
    }
```

```java
        public static void main(String args[])
        {
            // Creating a HashSet
            HashSet<String> set = new HashSet<String>();

            // Adding elements into HashSet using add()
            set.add("geeks");
            set.add("practice");
            set.add("contribute");
            set.add("ide");

            System.out.println("Original HashSet: "
                               + set);

            // Sorting HashSet using List
            List<String> list = new ArrayList<String>(set);
            Collections.sort(list);

            // Print the sorted elements of the HashSet
            System.out.println("HashSet elements "
                               + "in sorted order "
                               + "using List: "
                               + list);
        }
```

Algorithm for Depth-First Search

1. Mark the current vertex, u, visited (color it light blue), and enter it in the discovery order list
2. `for` each vertex, v, adjacent to the current vertex, u
3. `if` v has not been visited
4. Set parent of v to u.
5. Recursively apply this algorithm starting at v.
6. Mark u finished (color it dark blue) and enter u into the finish order list.