```java
//Classical Binary Search
public class Solution {
  public int binarySearch(int[] array, int target) {
    if (array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length -1;

    while( left <= right){ //注意边界条件， 这边是 left小于等于right
      int mid = left + (right - left)/2;
      if (array[mid]==target){
        return mid;
      } else if (array[mid]<target){
        left = mid +1;
      } else {
        right = mid -1;
      }
    }
    return -1;
  }

}
```

```java
public class Solution {
  public int firstOccur(int[] array, int target) {
    if(array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length-1;
    while(left < right -1){
      int mid = left + (right -left)/2;
      if (array[mid]<=target){
        left = mid + 1;
      } else {
        right = mid;
      }
    }
    if (array[left]==target){
      return left;
    } else if (array[right]== target){
      return right;
    }
    return -1;
  }
}

```

```java
public class Solution {
  public int lastOccur(int[] array, int target) {
    if (array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length -1;
    while (left <right -1){
      int mid = left +(right - left )/2;
      if (array[mid]<= target){
        left = mid;
      } else {
        right = mid;
      }
    }
    if (array[right]== target){ //
      return right;
    } else if (array[left]== target){
      return left;
    }
    return -1;
  }
}

```

```java
public class Solution {
  public int closest(int[] array, int target) {
    if(array==null||array.length==0){
      return -1;
    }
    int left = 0;
    int right = array.length-1;
    while(left <right -1){
      int mid = left + (right - left)/2;
      if(array[mid]==target){
        return mid;
      } else if (array[mid] <target){
        left = mid;
      } else {
        right = mid;
      }
    }
    if(Math.abs(array[left] - target) <= Math.abs(array[right]- target)){
      return left;
    }
    return right;
  }
}

```

```java
// convert the 2D array to 1D array and do binary search
public class Solution {
  public int[] search(int[][] matrix, int target) {
    if(matrix.length==0||matrix[0].length==0){
      return new int[] {-1,-1};
    }
    int rows=matrix.length;
    int cols= matrix[0].length;
    int left = 0;
    //convert the 2D array to 1D array with rows*cols elements
    int right = rows*cols-1;
    while(left <= right){
      int mid = left + (right - left)/2;
      // convert the postion in 1d array back to row and col in 2D array.
      int row = mid /cols;
      int col = mid % cols;
      if(matrix[row][col]==target){
        return new int[]{row,col};
      } else if (matrix[row][col] <  target){
        left = mid +1;
      } else {
        right = mid -1;
      }
    }
    return new int[] {-1,-1};
  }
}
```

// 561. Find the Kth Element in The Matrix

```
Given a matrix, find the Kth index element.




example:

matrix:

1 3 4

5 6 7

8 9 10



k = 4 → return: 6
```

```java
public class Solution {
  public int findElement(int[][] matrix, int k) {
    int i=k/matrix[0].length; //row
    int j=k%matrix[0].length;//columns
    return matrix[i][j];
  }
}
```

```java
public class Solution {
  public long power(int a, int b) {
    if (a == 0) {
      return 0;
    }
    if (b == 0) {
      return 1;
    }
    long half = power(a, b / 2);
    if (b % 2 == 0) {
      return half * half;
    } else {
      return a * half * half;
    }
  }
}

```

```java
class Solution {
    public int maxInQueue(Queue<Integer> queue) {
    int max=queue.poll();
    while(!queue.isEmpty()){
      max=Math.max(max,queue.poll());
    }
    return max;
    }
}
```

```java
class Solution {
    public int sumOfStack(Deque<Integer> stack) {
    int sum=stack.poll();
    while(!stack.isEmpty()){
      sum +=stack.poll();
    }
    return sum;
    }
}
```

```java
public class Solution {
  private Deque<Integer> s1;
  private Deque<Integer> s2;
  public Solution() {
    s1 = new ArrayDeque<>();
    s2 = new ArrayDeque<>();
  }
  public Integer poll() {
    if (s1.isEmpty() && s2.isEmpty()) {
      return null;
    }
    shuffle(s1, s2);
    return s2.pollFirst();
  }

  public void offer(int element) {
    s1.offerFirst(element);
  }

  public Integer peek() {
    if (s1.isEmpty() && s2.isEmpty()) {
      return null;
    }
    shuffle(s1, s2);
    return s2.peekFirst();
  }

  public int size() {
    return s1.size() + s2.size();
  }

  public boolean isEmpty() {
    return s1.isEmpty() && s2.isEmpty();
  }
  public void shuffle(Deque<Integer> s1, Deque<Integer> s2) {
    if (s2.isEmpty()) {
      while (!s1.isEmpty()) {
        s2.offerFirst(s1.pollFirst());
      }
    }
  }
}

```

```java
//最小栈 easy 版本 双端队列
class MinStack {

    /** initialize your data structure here. */
    Deque<Integer> mStack;
    Deque<Integer> minStack;
    public MinStack() {
        mStack = new LinkedList<Integer>();
        minStack = new LinkedList<Integer>();
        minStack.push(Integer.MAX_VALUE);

    }

    public void push(int x) {
        mStack.push(x);
        minStack.push(Math.min(minStack.peek(),x));
    }

    public void pop() {
        mStack.pop();
        minStack.pop();
    }

    public int top() {
        return mStack.peek();
    }

    public int getMin() {
        return minStack.peek();
    }
}

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack obj = new MinStack();
 * obj.push(x);
 * obj.pop();
 * int param_3 = obj.top();
 * int param_4 = obj.getMin();
 */


#
链接：https://leetcode-cn.com/problems/min-stack/solution/di-yi-bian-xian-ba-da-an-chao-hui-by-gavin-131/

// 最小栈 mid 版本 双端队列
public class Solution {
  private Deque<Integer> stack;
  private Deque<Integer> minStack;
  public Solution() {
    stack = new LinkedList<Integer>();
    minStack = new LinkedList<Integer>();
  }
  public Integer min() {
    if (minStack.isEmpty()){
      return -1;
    }
    return minStack.peekFirst();
  }
  public void push(int value){
    stack.offerFirst(value);
    if(minStack.isEmpty()|| value <= minStack.peekFirst()){
      minStack.offerFirst(value);
    }
  }
  public Integer pop(){
    if(stack.isEmpty()){
      return -1;
    }
    Integer result = stack.pollFirst();
```

```
70        if (minStack.peekFirst().equals(result)){
71          minStack.pollFirst();
72        }
73        return result;
74      }
75      public Integer top(){
76        if(stack.isEmpty()){
77          return -1;
78        }
79        return stack.peekFirst();
80      }
81    }
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode reverse(ListNode head) {
    if (head == null || head.next == null) {
      return head;
    }
    ListNode prev = null;
    ListNode curr = head;
    while (curr != null) {
      ListNode next = curr.next;
      curr.next = prev;
      prev = curr;
      curr = next;
    }
    return prev;
  }
}
```

```
1
2  # shuffle stacks
3  class Solution {
4      public void shuffle(Deque<Integer> stack1, Deque<Integer> stack2) {
5        while (!stack1.isEmpty()) {
6          stack2.push(stack1.pop());
7        }
8      }
9  }
10
```

```java
public class Solution {
  public ListNode generate(int n) {
    ListNode head = new ListNode(0);
    ListNode cur = head;
    for (int i = 1 ; i < n ; i++){
      cur.next = new ListNode(i);
      cur = cur.next;
    }
    return head;
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public int count(ListNode head) {
    ListNode cur = head;
    int i =0;
    while (cur != null){
      i++;
      cur = cur.next;
    }
    return i ;
  }
}

```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode reverse(ListNode head) {
    if(head ==null || head.next == null){
      return head;
    }
    ListNode curr = reverse(head.next);
    head.next.next = head;
    head.next = null;
    return curr;
  }
}

```

```java
public class Solution {
  public int[] solve(int[] array) {
    if(array==null||array.length==0){
      return array;
    }
    for(int i =0;i<array.length-1;i++){//why array.length-1
      int min = i;
      for(int j = i+1;j<array.length;j++){
        if(array[min]>array[j]){
          min=j;
        }
      }
      swap(array,i,min);

    }
    return array;
  }
  public void swap(int[] array, int left ,int right){
    int temp = array[left];
    array[left]=array[right];
    array[right]=temp;
  }
}

```

```java
public class Solution {
  public int[] mergeSort(int[] array) {
    if(array==null||array.length==0){
      return array;
    }
    //新建一个helper function
    int[] helper = new int [array.length];
    mergeSort(array,helper,0,array.length-1);
    return array;
  }

  //split
  private void mergeSort(int[] array, int[] helper , int left , int right){
    if(left >= right){
      return;
    }
    int mid = left +(right-left)/2;
    mergeSort(array, helper,left,mid);
    mergeSort(array,helper,mid+1,right);
    merge(array,helper,left,mid,right);
  }
  //merge
  private void merge(int[] array, int[] helper, int left , int mid , int right){
    for (int i = left ; i<= right; i++){
      helper[i] = array[i];
    }
    int leftIndex = left ;
    int rightIndex = mid +1;
    while(leftIndex <= mid&&rightIndex <= right){
      if(helper[leftIndex] <= helper[rightIndex]){
        array[left++] = helper[leftIndex++];
      } else{
        array[left++]=helper[rightIndex++];
      }
    }
    while(leftIndex <= mid){
      array[left++] = helper[leftIndex++];
    }
  }
}

```

```java
public class Solution {
  public int[] quickSort(int[] array) {
    if (array == null || array.length <= 1) {
      return array;
    }
    quickSort(array, 0, array.length - 1);
    return array;
  }
  private void quickSort(int[] array, int left, int right) {
    if (left >= right) {
      return;
    }
    int pivotIndex = patition(array, left, right);
    quickSort(array, left, pivotIndex - 1);
    quickSort(array, pivotIndex + 1, right);
  }
  private int patition(int[] array, int left, int right) {
    int randomNum = left + findRandom(right - left);
    int L = left;
    int R = right - 1;
    swap(array, randomNum, right);
    while (L <= R) {
      if(array[L] <= array[right]) {
        L++;
      } else {
        swap(array, L, R--);
      }
    }
    swap(array, L, right);
    return L;
  }
  private void swap(int[] array, int a, int b) {
    int temp = array[a];
    array[a] = array[b];
    array[b] = temp;
  }
  private int findRandom(int range) {
    Random rand = new Random();
    return rand.nextInt(range + 1);
  }
}

```

```java
public class Solution {
  public int[] rainbowSort(int[] array) {
    if (array.length == 0) {
      return array;
    }
    int i = 0;
    int j = 0;
    int k = array.length - 1;
    while (j <= k) {
      if (array[j] == 0) {
        j++;
      } else if (array[j] == 1) {
        swap(array, j, k--);
      } else {
        swap(array, i++, j++);
      }
    }
    return array;
  }
  private void swap(int[] array, int i, int j) {
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
  }
}

```

```java
public class Solution {
  public int minIndex(int[] array, int i) {
    int min = i;
    for(int j =i+1;j<array.length;j++){
      if(array[min]>array[j]){
        min=j;
      }
    }
    return min;
  }
}
```

```java
public class Solution {
  public int[] merge(int[] array1, int[] array2) {
    // 开拓新的空间，储存合并好的数组
    int[] result = new int[array1.length +array2.length];
    int i =0,j=0,k=0;
    while(i<array1.length&&j<array2.length){
      if(array1[i]<array2[j]){
        result[k]=array1[i];
        i++;
      } else{
        result[k]=array2[j];
        j++;
      }
      k++;
    }
    while(i<array1.length){
      result[k]=array1[i];
      i++;
      k++;
    }
    while(j<array2.length){
      result[k]=array2[j];
      j++;
      k++;
    }
    return result;
  }
}

```

```java
class Solution {
    public int random(int a, int b) {
                Random rand = new Random();
                return a + rand.nextInt(b - a + 1);
    }
}
```

```java
class Solution {
    public void partition(int[] array, int pivotIndex) {
    int pivot = array[pivotIndex];
    swap(array, pivotIndex ,array.length-1);
    int leftBound = 0 ;
    int rightBound = array.length-2;
    while(leftBound <= rightBound){
      if(array[leftBound] < pivot){
        leftBound++;
      }else{
        swap(array,leftBound,rightBound);
        rightBound --;
      }
    }
    swap(array,leftBound,array.length-1);
  }
  private void swap(int[] array, int left , int right){
    int temp = array[left];
    array[left] = array[right];
    array[right]=temp;
  }
}
```

```java
public class Solution {
  public boolean hasCycle(ListNode head) {
    if(head == null || head.next ==null){
      return false;
    }
    ListNode slow = head;
    ListNode fast =head.next;
    while(fast != null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
      if (slow == fast){
        return true;
      }
    }
    return false;
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode insert(ListNode head, int value) {
    ListNode newNode = new ListNode(value);
    //1.determine if the inserted node is before head.
    if(head == null || head.value >= value){
      newNode.next = head;
      return newNode;
    }
    //2. insert the new node to the right postion.
    //using the previous node to traverse the linked list
    // the insert postion of the new node should be between prev and prev.next
    ListNode prev = head;
    while (prev.next != null && prev.next.value < value){
      prev = prev.next;
    }
    newNode.next = prev.next;
    prev.next = newNode;
    return head;
  }
}

```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode merge(ListNode one, ListNode two) {
    if(one == null) {
      return two;
    }
    if (two == null) {
      return one;
    }
    ListNode dummy = new ListNode(0);
    ListNode curr = dummy;
    while (one != null && two != null) {
      if (one.value < two.value) {
        curr.next = one;
        one = one.next;
        curr = curr.next;
      } else {
        curr.next = two;
        two = two.next;
        curr = curr.next;
      }
    }
    if (one != null) {
      curr.next = one;
    } else {
      curr.next = two;
    }
    return dummy.next;
  }
}

```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode partition(ListNode head, int target) {
    if(head== null){
      return null;
    }
    ListNode fakeHeadSmall = new ListNode(0);
    ListNode fakeHeadLarge = new ListNode(0);
    ListNode smallCurr = fakeHeadSmall;
    ListNode largeCurr = fakeHeadLarge;
    ListNode current = head;
    while(current != null){
      if(current.value < target){
        smallCurr.next = current;
        smallCurr= current;
      } else {
        largeCurr.next = current;
        largeCurr = current;
      }
      current = current.next;
    }
    largeCurr.next = null;
    smallCurr.next = fakeHeadLarge.next;
    return fakeHeadSmall.next;
  }
}

```

```java
public class Solution {
  public ListNode findMiddleNode(ListNode head) {
    if (head == null || head.next ==null){
      return head;
    }
    ListNode fast = head;
    ListNode slow = head;
    while (fast != null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
    }
    return slow; //针对奇数节点中间值的情况，slow 节点会落在中间点上
  }
}
```

```java
/**
 * class ListNode {
 *   public int value;
 *   public ListNode next;
 *   public ListNode(int value) {
 *     this.value = value;
 *     next = null;
 *   }
 * }
 */
public class Solution {
  public ListNode findMiddleNode(ListNode head) {
    if(head ==null || head.next==null){
      return head;
    }
    ListNode slow = head;
    ListNode fast = head.next;
    while(fast!= null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
    }
    return slow;
  }
}

```

```java
public class Solution {
  public ListNode findMiddleNode(ListNode head) {
    if (head== null || head.next ==null){
      return head;
    }
    ListNode slow = head;
    ListNode fast = head.next;
    while(fast!= null && fast.next != null){
      slow = slow.next;
      fast = fast.next.next;
    }
    return slow.next;
  }
}

```

```java
public class Solution {
  public ListNode insertNode(ListNode head, int target) {
    ListNode curr = head;
    ListNode newHead = new ListNode (target);
    while (curr.next != null){
      if(target >= curr.value && target <= curr.next.value){
        ListNode temp = curr.next;
        curr.next = newHead;
        newHead.next = temp;
        return head;
      }
      curr = curr.next;
    }
    return head;
  }
}
```

```java
public class Solution {
  public ListNode insertNode(ListNode head, int target) {
    ListNode curr = head;
    ListNode newNode = new ListNode(target);
    if(curr == null){
      return newNode;
    }
    while (curr.next != null){
      curr = curr.next;
    }
    curr.next = newNode;
    return head;

  }
}
```

```java


class Solution {
    public ListNode middleNode(ListNode head) {
        //边界条件不用忘记处理了
        if(head==null || head.next==null) {
            return head;
        }
        //定义慢指针，快指针
        ListNode low = head;
        ListNode fast = head.next;
        while(fast!=null && fast.next!=null) {
            //慢指针每次走一步，快指针每次走两步
            low = low.next;
            fast = fast.next.next;
        }
        //根据快指针是否为空判断边界条件
        if(fast!=null) {
            return low.next;
        }
        return low;
    }
}
```

```java
/**
 * public class TreeNode {
 *   public int key;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int key) {
 *     this.key = key;
 *   }
 * }
 */
public class Solution {
  public boolean isBalanced(TreeNode root) {
    if (root == null) {
      return true;
    }
    if (Math.abs(getHeight(root.left) - getHeight(root.right)) > 1) {
      return false;
    }
    return isBalanced(root.right) && isBalanced(root.left);
  }
  private int getHeight(TreeNode root) {
    if (root == null) {
      return 0;
    }
    int L = getHeight(root.left);
    int R = getHeight(root.right);
    return Math.max(L, R) + 1;
  }
}

```

```java
/**
 * public class TreeNode {
 *   public int key;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int key) {
 *     this.key = key;
 *   }
 * }
 */
public class Solution {
  public int findHeight(TreeNode root) {
    if(root == null){
      return 0;

    }
    int leftHeight = findHeight(root.left);
    int rightHeight = findHeight(root.right);
    return Math.max(findHeight(root.left),findHeight(root.right))+1;

  }
}

```

```java
/**
 * public class TreeNode {
 *   public int key;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int key) {
 *     this.key = key;
 *   }
 * }
 */
public class Solution {
  public int countNodes(TreeNode root) {
    if (root == null) {
      return 0;
    }
    int left = countNodes(root.left);
    int right = countNodes(root.right);
    return countNodes(root.left) + countNodes(root.right) + 1;
  }
}

```

```java
/**
 * public class TreeNode {
 *   public int key;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int key) {
 *     this.key = key;
 *   }
 * }
 */
public class Solution {
  public List<Integer> inOrder(TreeNode root) {
    List<Integer> res = new ArrayList<>();
    helper(root,res);
    return res;
  }
  private void helper(TreeNode root ,List<Integer> res){
    if (root == null){
      return;
    }
    helper(root.left,res);
    res.add(root.key);
    helper(root.right,res);
  }
}
```

```java
/**
 * public class TreeNode {
 *   public int key;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int key) {
 *     this.key = key;
 *   }
 * }
 */
public class Solution {
  public List<Integer> preOrder(TreeNode root) {
    List<Integer> res = new ArrayList<>();
    helper(root,res);
    return res;
  }
  private void helper(TreeNode root, List<Integer> res){
    if (root == null ){
      return;
    }
    res.add(root.key);
    helper(root.left,res);
    helper(root.right,res);
  }
}

```

```java
/**
 * public class TreeNode {
 *   public int key;
 *   public TreeNode left;
 *   public TreeNode right;
 *   public TreeNode(int key) {
 *     this.key = key;
 *   }
 * }
 */
public class Solution {
  public List<Integer> postOrder(TreeNode root) {
    List<Integer> res = new ArrayList<>();
    helper(root, res);
    return res;
  }
  private void helper(TreeNode root, List<Integer> res) {
    if (root == null) {
      return;
    }
    helper(root.left, res);
    helper(root.right, res);
    res.add(root.key);
  }
}

```