

```
1 //Classical Binary Search
2 public class Solution {
3     public int binarySearch(int[] array, int target) {
4         if (array==null||array.length==0){
5             return -1;
6         }
7         int left = 0;
8         int right = array.length -1;
9
10        while( left <= right){ //注意边界条件, 这边是 left小于等于right
11            int mid = left + (right - left)/2;
12            if (array[mid]==target){
13                return mid;
14            } else if (array[mid]<target){
15                left = mid +1;
16            } else {
17                right = mid -1;
18            }
19        }
20        return -1;
21    }
22
23 }
```

```
1  public class Solution {
2      public int firstOccur(int[] array, int target) {
3          if(array==null||array.length==0){
4              return -1;
5          }
6          int left = 0;
7          int right = array.length-1;
8          while(left < right -1){
9              int mid = left + (right -left)/2;
10             if (array[mid]<=target){
11                 left = mid + 1;
12             } else {
13                 right = mid;
14             }
15         }
16         if (array[left]==target){
17             return left;
18         } else if (array[right]== target){
19             return right;
20         }
21         return -1;
22     }
23 }
24
```

```
1 public class Solution {
2     public int lastOccur(int[] array, int target) {
3         if (array==null||array.length==0){
4             return -1;
5         }
6         int left = 0;
7         int right = array.length -1;
8         while (left <right -1){
9             int mid = left +(right - left )/2;
10            if (array[mid]<= target){
11                left = mid;
12            } else {
13                right = mid;
14            }
15        }
16        if (array[right]== target){ //
17            return right;
18        } else if (array[left]== target){
19            return left;
20        }
21        return -1;
22    }
23 }
24
```

```
1 public class Solution {
2     public int closest(int[] array, int target) {
3         if(array==null||array.length==0){
4             return -1;
5         }
6         int left = 0;
7         int right = array.length-1;
8         while(left <right -1){
9             int mid = left + (right - left)/2;
10            if(array[mid]==target){
11                return mid;
12            } else if (array[mid] <target){
13                left = mid;
14            } else {
15                right = mid;
16            }
17        }
18        if(Math.abs(array[left] - target) <= Math.abs(array[right]- target)){
19            return left;
20        }
21        return right;
22    }
23 }
24
```

```
1  // convert the 2D array to 1D array and do binary search
2  public class Solution {
3      public int[] search(int[][] matrix, int target) {
4          if(matrix.length==0||matrix[0].length==0){
5              return new int[] {-1,-1};
6          }
7          int rows=matrix.length;
8          int cols= matrix[0].length;
9          int left = 0;
10         //convert the 2D array to 1D array with rows*cols elements
11         int right = rows*cols-1;
12         while(left <= right){
13             int mid = left + (right - left)/2;
14             // convert the postion in 1d array back to row and col in 2D array.
15             int row = mid /cols;
16             int col = mid % cols;
17             if(matrix[row][col]==target){
18                 return new int[]{row,col};
19             } else if (matrix[row][col] < target){
20                 left = mid +1;
21             } else {
22                 right = mid -1;
23             }
24         }
25         return new int[] {-1,-1};
26     }
27 }
```

```
1 // 561. Find the Kth Element in The Matrix
2 ```
3 Given a matrix, find the Kth index element.
4
5
6
7
8
9 example:
10
11 matrix:
12
13 1 3 4
14
15 5 6 7
16
17 8 9 10
18
19
20
21 k = 4 → return: 6
22 ```
23 public class Solution {
24     public int findElement(int[][] matrix, int k) {
25         int i=k/matrix[0].length; //row
26         int j=k%matrix[0].length;//columns
27         return matrix[i][j];
28     }
29 }
30
```

```
1 public class Solution {
2     public long power(int a, int b) {
3         if (a == 0) {
4             return 0;
5         }
6         if (b == 0) {
7             return 1;
8         }
9         long half = power(a, b / 2);
10        if (b % 2 == 0) {
11            return half * half;
12        } else {
13            return a * half * half;
14        }
15    }
16 }
17
```

```
1  class Solution {
2      public int maxInQueue(Queue<Integer> queue) {
3          int max=queue.poll();
4          while(!queue.isEmpty()){
5              max=Math.max(max,queue.poll());
6          }
7          return max;
8      }
9  }
```



```
1  class Solution {
2      public int sumOfStack(Deque<Integer> stack) {
3          int sum=stack.poll();
4          while(!stack.isEmpty()){
5              sum +=stack.poll();
6          }
7          return sum;
8      }
9  }
```

```
1  public class Solution {
2      private Deque<Integer> s1;
3      private Deque<Integer> s2;
4      public Solution() {
5          s1 = new ArrayDeque<>();
6          s2 = new ArrayDeque<>();
7      }
8      public Integer poll() {
9          if (s1.isEmpty() && s2.isEmpty()) {
10             return null;
11         }
12         shuffle(s1, s2);
13         return s2.pollFirst();
14     }
15
16     public void offer(int element) {
17         s1.offerFirst(element);
18     }
19
20     public Integer peek() {
21         if (s1.isEmpty() && s2.isEmpty()) {
22             return null;
23         }
24         shuffle(s1, s2);
25         return s2.peekFirst();
26     }
27
28     public int size() {
29         return s1.size() + s2.size();
30     }
31
32     public boolean isEmpty() {
33         return s1.isEmpty() && s2.isEmpty();
34     }
35     public void shuffle(Deque<Integer> s1, Deque<Integer> s2) {
36         if (s2.isEmpty()) {
37             while (!s1.isEmpty()) {
38                 s2.offerFirst(s1.pollFirst());
39             }
40         }
41     }
42 }
43
```

```

1 //最小栈 easy 版本 双端队列
2 class MinStack {
3
4     /** initialize your data structure here. */
5     Deque<Integer> mStack;
6     Deque<Integer> minStack;
7     public MinStack() {
8         mStack = new LinkedList<Integer>();
9         minStack = new LinkedList<Integer>();
10        minStack.push(Integer.MAX_VALUE);
11
12    }
13
14    public void push(int x) {
15        mStack.push(x);
16        minStack.push(Math.min(minStack.peek(), x));
17    }
18
19    public void pop() {
20        mStack.pop();
21        minStack.pop();
22    }
23
24    public int top() {
25        return mStack.peek();
26    }
27
28    public int getMin() {
29        return minStack.peek();
30    }
31 }
32
33 /**
34  * Your MinStack object will be instantiated and called as such:
35  * MinStack obj = new MinStack();
36  * obj.push(x);
37  * obj.pop();
38  * int param_3 = obj.top();
39  * int param_4 = obj.getMin();
40  */
41
42
43 #
44 链接: https://leetcode-cn.com/problems/min-stack/solution/di-yi-bian-xian-ba-da-an-cha-o-hui-by-gavin-131/
45
46 // 最小栈 mid 版本 双端队列
47 public class Solution {
48     private Deque<Integer> stack;
49     private Deque<Integer> minStack;
50     public Solution() {
51         stack = new LinkedList<Integer>();
52         minStack = new LinkedList<Integer>();
53     }
54     public Integer min() {
55         if (minStack.isEmpty()) {
56             return -1;
57         }
58         return minStack.peekFirst();
59     }
60     public void push(int value) {
61         stack.offerFirst(value);
62         if (minStack.isEmpty() || value <= minStack.peekFirst()) {
63             minStack.offerFirst(value);
64         }
65     }
66     public Integer pop() {
67         if (stack.isEmpty()) {
68             return -1;
69         }
70         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 }
```

```
1  /**
2   * class ListNode {
3   *   public int value;
4   *   public ListNode next;
5   *   public ListNode(int value) {
6   *     this.value = value;
7   *     next = null;
8   *   }
9   * }
10 */
11 public class Solution {
12     public ListNode reverse(ListNode head) {
13         if (head == null || head.next == null) {
14             return head;
15         }
16         ListNode prev = null;
17         ListNode curr = head;
18         while (curr != null) {
19             ListNode next = curr.next;
20             curr.next = prev;
21             prev = curr;
22             curr = next;
23         }
24         return prev;
25     }
26 }
```

```
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
```

```
1  public class Solution {
2      public ListNode generate(int n) {
3          ListNode head = new ListNode(0);
4          ListNode cur = head;
5          for (int i = 1 ; i < n ; i++){
6              cur.next = new ListNode(i);
7              cur = cur.next;
8          }
9          return head;
10     }
11 }
```

```
1  /**
2  * class ListNode {
3  *   public int value;
4  *   public ListNode next;
5  *   public ListNode(int value) {
6  *     this.value = value;
7  *     next = null;
8  *   }
9  * }
10 */
11 public class Solution {
12     public int count(ListNode head) {
13         ListNode cur = head;
14         int i = 0;
15         while (cur != null){
16             i++;
17             cur = cur.next;
18         }
19         return i ;
20     }
21 }
22
```



```
1  /**
2  * class ListNode {
3  *   public int value;
4  *   public ListNode next;
5  *   public ListNode(int value) {
6  *     this.value = value;
7  *     next = null;
8  *   }
9  * }
10 */
11 public class Solution {
12     public ListNode reverse(ListNode head) {
13         if(head ==null || head.next == null){
14             return head;
15         }
16         ListNode curr = reverse(head.next);
17         head.next.next = head;
18         head.next = null;
19         return curr;
20     }
21 }
22
```

```
1 public class Solution {
2     public int[] solve(int[] array) {
3         if(array==null||array.length==0){
4             return array;
5         }
6         for(int i =0;i<array.length-1;i++){//why array.length-1
7             int min = i;
8             for(int j = i+1;j<array.length;j++){
9                 if(array[min]>array[j]){
10                     min=j;
11                 }
12             }
13             swap(array,i,min);
14
15         }
16         return array;
17     }
18     public void swap(int[] array, int left ,int right){
19         int temp = array[left];
20         array[left]=array[right];
21         array[right]=temp;
22     }
23 }
24
```

```
1 public class Solution {
2     public int[] mergeSort(int[] array) {
3         if(array==null||array.length==0){
4             return array;
5         }
6         //新建一个helper function
7         int[] helper = new int [array.length];
8         mergeSort(array,helper,0,array.length-1);
9         return array;
10    }
11
12    //split
13    private void mergeSort(int[] array, int[] helper , int left , int right){
14        if(left >= right){
15            return;
16        }
17        int mid = left +(right-left)/2;
18        mergeSort(array, helper,left,mid);
19        mergeSort(array,helper,mid+1,right);
20        merge(array,helper,left,mid,right);
21    }
22    //merge
23    private void merge(int[] array, int[] helper, int left , int mid , int right){
24        for (int i = left ; i<= right; i++){
25            helper[i] = array[i];
26        }
27        int leftIndex = left ;
28        int rightIndex = mid +1;
29        while(leftIndex <= mid&&rightIndex <= right){
30            if(helper[leftIndex] <= helper[rightIndex]){
31                array[left++] = helper[leftIndex++];
32            } else{
33                array[left++]=helper[rightIndex++];
34            }
35        }
36        while(leftIndex <= mid){
37            array[left++] = helper[leftIndex++];
38        }
39    }
40 }
41
```

```
1 public class Solution {
2     public int[] quickSort(int[] array) {
3         if (array == null || array.length <= 1) {
4             return array;
5         }
6         quickSort(array, 0, array.length - 1);
7         return array;
8     }
9     private void quickSort(int[] array, int left, int right) {
10        if (left >= right) {
11            return;
12        }
13        int pivotIndex = partition(array, left, right);
14        quickSort(array, left, pivotIndex - 1);
15        quickSort(array, pivotIndex + 1, right);
16    }
17    private int partition(int[] array, int left, int right) {
18        int randomNum = left + findRandom(right - left);
19        int L = left;
20        int R = right - 1;
21        swap(array, randomNum, right);
22        while (L <= R) {
23            if(array[L] <= array[right]) {
24                L++;
25            } else {
26                swap(array, L, R--);
27            }
28        }
29        swap(array, L, right);
30        return L;
31    }
32    private void swap(int[] array, int a, int b) {
33        int temp = array[a];
34        array[a] = array[b];
35        array[b] = temp;
36    }
37    private int findRandom(int range) {
38        Random rand = new Random();
39        return rand.nextInt(range + 1);
40    }
41 }
42
```

```
1 public class Solution {
2     public int[] rainbowSort(int[] array) {
3         if (array.length == 0) {
4             return array;
5         }
6         int i = 0;
7         int j = 0;
8         int k = array.length - 1;
9         while (j <= k) {
10             if (array[j] == 0) {
11                 j++;
12             } else if (array[j] == 1) {
13                 swap(array, j, k--);
14             } else {
15                 swap(array, i++, j++);
16             }
17         }
18         return array;
19     }
20     private void swap(int[] array, int i, int j) {
21         int temp = array[i];
22         array[i] = array[j];
23         array[j] = temp;
24     }
25 }
26
```

```
1 public class Solution {  
2     public int minIndex(int[] array, int i) {  
3         int min = i;  
4         for(int j =i+1;j<array.length;j++){  
5             if(array[min]>array[j]){  
6                 min=j;  
7             }  
8         }  
9         return min;  
10    }  
11 }
```

```
1 public class Solution {
2     public int[] merge(int[] array1, int[] array2) {
3         // 开拓新的空间, 储存合并好的数组
4         int[] result = new int[array1.length + array2.length];
5         int i = 0, j = 0, k = 0;
6         while(i < array1.length && j < array2.length) {
7             if(array1[i] < array2[j]) {
8                 result[k] = array1[i];
9                 i++;
10            } else {
11                result[k] = array2[j];
12                j++;
13            }
14            k++;
15        }
16        while(i < array1.length) {
17            result[k] = array1[i];
18            i++;
19            k++;
20        }
21        while(j < array2.length) {
22            result[k] = array2[j];
23            j++;
24            k++;
25        }
26        return result;
27    }
28 }
29
```

```
1  class Solution {
2      public int random(int a, int b) {
3          Random rand = new Random();
4          return a + rand.nextInt(b - a + 1);
5      }
6  }
```



```
1  class Solution {
2      public void partition(int[] array, int pivotIndex) {
3          int pivot = array[pivotIndex];
4          swap(array, pivotIndex ,array.length-1);
5          int leftBound = 0 ;
6          int rightBound = array.length-2;
7          while(leftBound <= rightBound){
8              if(array[leftBound] < pivot){
9                  leftBound++;
10             }else{
11                 swap(array,leftBound,rightBound);
12                 rightBound --;
13             }
14         }
15         swap(array,leftBound,array.length-1);
16     }
17     private void swap(int[] array, int left , int right){
18         int temp = array[left];
19         array[left] = array[right];
20         array[right]=temp;
21     }
22 }
```

```
1  public class Solution {
2      public boolean hasCycle(ListNode head) {
3          if(head == null || head.next ==null){
4              return false;
5          }
6          ListNode slow = head;
7          ListNode fast =head.next;
8          while(fast != null && fast.next != null){
9              slow = slow.next;
10             fast = fast.next.next;
11             if (slow == fast){
12                 return true;
13             }
14         }
15         return false;
16     }
17 }
```

```
1  /**
2   * class ListNode {
3   *   public int value;
4   *   public ListNode next;
5   *   public ListNode(int value) {
6   *     this.value = value;
7   *     next = null;
8   *   }
9   * }
10 */
11 public class Solution {
12     public ListNode insert(ListNode head, int value) {
13         ListNode newNode = new ListNode(value);
14         //1.determine if the inserted node is before head.
15         if(head == null || head.value >= value){
16             newNode.next = head;
17             return newNode;
18         }
19         //2. insert the new node to the right postion.
20         //using the previous node to traverse the linked list
21         // the insert postion of the new node should be between prev and prev.next
22         ListNode prev = head;
23         while (prev.next != null && prev.next.value < value){
24             prev = prev.next;
25         }
26         newNode.next = prev.next;
27         prev.next = newNode;
28         return head;
29     }
30 }
31
```

```
1  /**
2   * class ListNode {
3   *   public int value;
4   *   public ListNode next;
5   *   public ListNode(int value) {
6   *     this.value = value;
7   *     next = null;
8   *   }
9   * }
10 */
11 public class Solution {
12     public ListNode merge(ListNode one, ListNode two) {
13         if(one == null) {
14             return two;
15         }
16         if (two == null) {
17             return one;
18         }
19         ListNode dummy = new ListNode(0);
20         ListNode curr = dummy;
21         while (one != null && two != null) {
22             if (one.value < two.value) {
23                 curr.next = one;
24                 one = one.next;
25                 curr = curr.next;
26             } else {
27                 curr.next = two;
28                 two = two.next;
29                 curr = curr.next;
30             }
31         }
32         if (one != null) {
33             curr.next = one;
34         } else {
35             curr.next = two;
36         }
37         return dummy.next;
38     }
39 }
40
```

```

1  /**
2  * class ListNode {
3  *     public int value;
4  *     public ListNode next;
5  *     public ListNode(int value) {
6  *         this.value = value;
7  *         next = null;
8  *     }
9  * }
10 */
11 public class Solution {
12     public ListNode partition(ListNode head, int target) {
13         if(head== null){
14             return null;
15         }
16         ListNode fakeHeadSmall = new ListNode(0);
17         ListNode fakeHeadLarge = new ListNode(0);
18         ListNode smallCurr = fakeHeadSmall;
19         ListNode largeCurr = fakeHeadLarge;
20         ListNode current = head;
21         while(current != null){
22             if(current.value < target){
23                 smallCurr.next = current;
24                 smallCurr= current;
25             } else {
26                 largeCurr.next = current;
27                 largeCurr = current;
28             }
29             current = current.next;
30         }
31         largeCurr.next = null;
32         smallCurr.next = fakeHeadLarge.next;
33         return fakeHeadSmall.next;
34     }
35 }
36

```

```
1  public class Solution {
2      public ListNode findMiddleNode(ListNode head) {
3          if (head == null || head.next == null){
4              return head;
5          }
6          ListNode fast = head;
7          ListNode slow = head;
8          while (fast != null && fast.next != null){
9              slow = slow.next;
10             fast = fast.next.next;
11         }
12         return slow; //针对奇数节点中间值的情况，slow 节点会落在中间点上
13     }
14 }
```

```
1  /**
2  * class ListNode {
3  *   public int value;
4  *   public ListNode next;
5  *   public ListNode(int value) {
6  *     this.value = value;
7  *     next = null;
8  *   }
9  * }
10 */
11 public class Solution {
12     public ListNode findMiddleNode(ListNode head) {
13         if(head ==null || head.next==null){
14             return head;
15         }
16         ListNode slow = head;
17         ListNode fast = head.next;
18         while(fast!= null && fast.next != null){
19             slow = slow.next;
20             fast = fast.next.next;
21         }
22         return slow;
23     }
24 }
25
```

```
1  public class Solution {
2      public ListNode findMiddleNode(ListNode head) {
3          if (head== null || head.next ==null){
4              return head;
5          }
6          ListNode slow = head;
7          ListNode fast = head.next;
8          while(fast!= null && fast.next != null){
9              slow = slow.next;
10             fast = fast.next.next;
11         }
12         return slow.next;
13     }
14 }
15
```



```
1  public class Solution {
2      public ListNode insertNode(ListNode head, int target) {
3          ListNode curr = head;
4          ListNode newHead = new ListNode (target);
5          while (curr.next != null){
6              if(target >= curr.value && target <= curr.next.value){
7                  ListNode temp = curr.next;
8                  curr.next = newHead;
9                  newHead.next = temp;
10                 return head;
11             }
12             curr = curr.next;
13         }
14         return head;
15     }
16 }
```

```
1  public class Solution {
2      public ListNode insertNode(ListNode head, int target) {
3          ListNode curr = head;
4          ListNode newNode = new ListNode(target);
5          if(curr == null){
6              return newNode;
7          }
8          while (curr.next != null){
9              curr = curr.next;
10         }
11         curr.next = newNode;
12         return head;
13     }
14 }
15 }
```

```
1
2
3
4
5 class Solution {
6     public ListNode middleNode(ListNode head) {
7         //边界条件不用忘记处理了
8         if(head==null || head.next==null) {
9             return head;
10        }
11        //定义慢指针，快指针
12        ListNode low = head;
13        ListNode fast = head.next;
14        while(fast!=null && fast.next!=null) {
15            //慢指针每次走一步，快指针每次走两步
16            low = low.next;
17            fast = fast.next.next;
18        }
19        //根据快指针是否为空判断边界条件
20        if(fast!=null) {
21            return low.next;
22        }
23        return low;
24    }
25 }
```

```
1  /**
2   * public class TreeNode {
3   *     public int key;
4   *     public TreeNode left;
5   *     public TreeNode right;
6   *     public TreeNode(int key) {
7   *         this.key = key;
8   *     }
9   * }
10 */
11 public class Solution {
12     public boolean isBalanced(TreeNode root) {
13         if (root == null) {
14             return true;
15         }
16         if (Math.abs(getHeight(root.left) - getHeight(root.right)) > 1) {
17             return false;
18         }
19         return isBalanced(root.right) && isBalanced(root.left);
20     }
21     private int getHeight(TreeNode root) {
22         if (root == null) {
23             return 0;
24         }
25         int L = getHeight(root.left);
26         int R = getHeight(root.right);
27         return Math.max(L, R) + 1;
28     }
29 }
30
```

```
1  /**
2   * public class TreeNode {
3   *     public int key;
4   *     public TreeNode left;
5   *     public TreeNode right;
6   *     public TreeNode(int key) {
7   *         this.key = key;
8   *     }
9   * }
10 */
11 public class Solution {
12     public int findHeight(TreeNode root) {
13         if(root == null){
14             return 0;
15         }
16         int leftHeight = findHeight(root.left);
17         int rightHeight = findHeight(root.right);
18         return Math.max(findHeight(root.left),findHeight(root.right))+1;
19     }
20 }
21
22
23
```

```
1  /**
2   * public class TreeNode {
3   *     public int key;
4   *     public TreeNode left;
5   *     public TreeNode right;
6   *     public TreeNode(int key) {
7   *         this.key = key;
8   *     }
9   * }
10 */
11 public class Solution {
12     public int countNodes(TreeNode root) {
13         if (root == null) {
14             return 0;
15         }
16         int left = countNodes(root.left);
17         int right = countNodes(root.right);
18         return countNodes(root.left) + countNodes(root.right) + 1;
19     }
20 }
21
```

```
1  /**
2  * public class TreeNode {
3  *     public int key;
4  *     public TreeNode left;
5  *     public TreeNode right;
6  *     public TreeNode(int key) {
7  *         this.key = key;
8  *     }
9  * }
10 */
11 public class Solution {
12     public List<Integer> inOrder(TreeNode root) {
13         List<Integer> res = new ArrayList<>();
14         helper(root,res);
15         return res;
16     }
17     private void helper(TreeNode root ,List<Integer> res){
18         if (root == null){
19             return;
20         }
21         helper(root.left,res);
22         res.add(root.key);
23         helper(root.right,res);
24     }
25 }
26
27
```

```
1  /**
2   * public class TreeNode {
3   *     public int key;
4   *     public TreeNode left;
5   *     public TreeNode right;
6   *     public TreeNode(int key) {
7   *         this.key = key;
8   *     }
9   * }
10 */
11 public class Solution {
12     public List<Integer> preOrder(TreeNode root) {
13         List<Integer> res = new ArrayList<>();
14         helper(root,res);
15         return res;
16     }
17     private void helper(TreeNode root, List<Integer> res){
18         if (root == null ){
19             return;
20         }
21         res.add(root.key);
22         helper(root.left,res);
23         helper(root.right,res);
24     }
25 }
26
```



```
1  /**
2   * public class TreeNode {
3   *     public int key;
4   *     public TreeNode left;
5   *     public TreeNode right;
6   *     public TreeNode(int key) {
7   *         this.key = key;
8   *     }
9   * }
10 */
11 public class Solution {
12     public List<Integer> postOrder(TreeNode root) {
13         List<Integer> res = new ArrayList<>();
14         helper(root, res);
15         return res;
16     }
17     private void helper(TreeNode root, List<Integer> res) {
18         if (root == null) {
19             return;
20         }
21         helper(root.left, res);
22         helper(root.right, res);
23         res.add(root.key);
24     }
25 }
26
```