**LeetCode Summary**

|  |  |
| --- | --- |
| The difference between Comparable<T> and Comparator<T>  If you want to sort a primitive type array list,  Then use Arrays.sort();  String sort(String s) {  char[] content = s.toArrayList();  java.util.Arrays.sort(content);  return new String(content);  //how to use a character list form a new string  } | 1. If you use Comparable<T>    Use Collections.sort(Students), it will sort based on the rules defined in the CompareTo methods  2. If you use Comparator<T> to define a new way to sort ArrayList, sorting not based on the way defined in the CompareTo methods.    Use Collections.sort(Students, new StudentCompanyComparator ())  Or Use Collections.sort(Students, new StudentCompanyComparator().reversed())  Compare Integer, use Integer.compare();  Compare Character, use Character.compare();  Compare Double, use Double.compare();  Compare Short, use Short.compare();  Compare Long, use Long.compare();  Transfer a string to a char list,  String s = “Java Developer Rocks!”;  char[] charList = s.toCharArray(); |
| Given an array of integers, return indices of the two numbers such that they add up to a specific target.  You may assume that each input would have exactly one solution, and you may not use the same element twice. | class Solution {  public int[] twoSum(int[] nums, int target) {  for (int i = 0; i < nums.length; i++) {  for (int j = i + 1; j < nums.length; j++) {  if ( (nums[i] + nums[j]) == target) {  return new int[] {i, j};  }  }  }  throw new IllegalArgumentException("No two sum solution");  }  }  If you start loop at position i, then j = i + 1 means loop starting at the following position;  Create new int[] array; new int[] {5, 6};  Throw new IllegalArgumentException(“No two sum solution”) |

|  |  |
| --- | --- |
| <https://leetcode.com/problems/missing-number/description/>  Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.  For example, Given nums = [0, 1, 3] return 2. | class Solution {  public int missingNumber(int[] nums) {  Arrays.sort(nums);    for (int i = 0; i < (nums.length); i++) {  if ( (nums[i] - i) != 0 ) {  return i;  }  }  return (nums.length);  }  }  Sort Array use method Arrays.sort(); |

|  |  |
| --- | --- |
| <https://leetcode.com/problems/find-peak-element/description/>  A peak element is an element that is greater than its neighbors.  Given an input array where num[i] ≠ num[i+1], find a peak element and return its index.  The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.  You may imagine that num[-1] = num[n] = -∞.  For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2. | public class Solution{  public int findPeakElement(int[] nums) {  for (int i = 0; i < nums.length - 1; i++) {  if (nums[i] > nums[i + 1])  return i;  }  return nums.length - 1;  }  }  If you want to loop a list and compare the neighboring elements, then use  (int i = 0; i < nums.length – 1; i++) {  if (nums[i] > nums[i + 1]) {  return i;  }  return nums.length - 1;  }  Pay attention, because each time point to nums[i + 1], i < ( nums.length – 1 ) to avoid index out of bound. |

|  |  |
| --- | --- |
| <https://leetcode.com/problems/best-time-to-buy-and-sell-stock/description/>  Say you have an array for which the ith element is the price of a given stock on day i.  If you were only permitted to complete at most one transaction (ie, buy one and sell one share of the stock), design an algorithm to find the maximum profit. | public class Solution {  public int maxProfit(int prices[]) {  int maxprofit = 0;  for (int i = 0; i < prices.length - 1; i++) {  for (int j = i + 1; j < prices.length; j++) {  int profit = prices[j] - prices[i];  if (profit > maxprofit)  maxprofit = profit;  }  }  return maxprofit;  }  } |

|  |  |
| --- | --- |
| <https://leetcode.com/problems/palindrome-number/description/>  Determine whether an integer is a palindrome. Do this without extra space. | class Solution {  public boolean isPalindrome(int x) {  String s = String.valueOf(x);  search:  for (int i = 0; i < s.length(); i++) {  for (int j = s.length() - 1 - i; j > 0; j--) {  if (s.charAt(i) == s.charAt(j)) {  continue search;  } else {  return false;  }  }  }  return true;  }  }  Use s.length – 1 to point the last element of a string;  Use recursive method to check isPalindrome  isPalindrome(String s) {  if (s.length() <= 1) return true;  return s[0] == s.length() - 1 && isPalindrome(s.subString[1, s.length() - 1];  } |
| How to check is a character is between ‘A’ and ‘Z’, ‘a’ and ‘z’ | String s = CertainString;  char[] charList = s.toArrayList();  for (int i = 0; i < s.length(); i++) {  if ( ( s.charAt(i) < 'Z' and s.charAt() >'A' ) || ( s.charAt(i) < 'z' and s.charAt() > 'a') )  {  return true;  }  } |

|  |  |
| --- | --- |
| ASCII and Unicode | In ASCII, upper case letters come before lower-case letters.  In total, there are 128 characters defined in the ASCII encoding, the way to represent human-readable text using only 1s and 0s.  In total, there are 256 characters defined in the extended ASCII extended encoding.  Unicode is a superset of ASCII, and the numbers 0-128 have the same meaning in ASCII as they have in Unicode. Because Unicode characters don’t generally fi into one 8-bit byte, there are numerous ways of storing Unicode characters in Byte sequences, such as UTF-32 and UTF-8.  01001000 01100101 01101100 01101100 01101111 01010111 01101111 01110010 01101100 01100100  “Hello World” |

|  |  |
| --- | --- |
| Implement an algorithm to determine if a string has all unique characters. | Assuming 128 ASCII string    boolean isUnique(String str) {  if (str.length() > 128) return false;    boolean[] char\_set = new boolean[128];  for (int i = 0; i < str.length(); i++) {  int val = str.charAt(i);  if (char\_set[val]) {  return false;  }  char\_set[val] = true;  }  return true;  } |

|  |  |
| --- | --- |
| Check if the two strings have identical character counts | See character,  int[] letters = new int[128];  there are 128 characters in ACSII. |