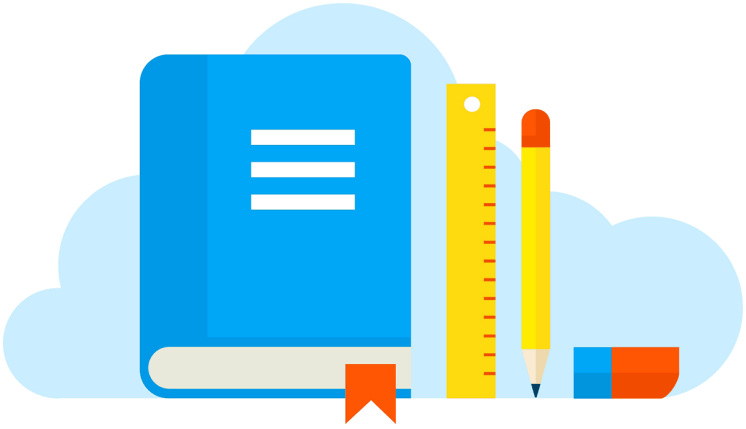


Coding Interview

Java (Part – 1)

1. Rotate Array in Java
2. Evaluate Reverse Polish Notation
3. Isomorphic Strings
4. Word Ladder
5. Word Ladder II
6. Median of Two Sorted Arrays
7. Kth Largest Element in an Array
8. Wildcard Matching
9. Regular Expression Matching in Java
10. Merge Intervals
11. Insert Interval
12. Two Sum
13. Two Sum II Input array is sorted
14. Two Sum III Data structure design
15. 3Sum
16. 4Sum
17. 3Sum Closest
18. String to Integer (atoi)

1 Rotate Array in Java

You may have been using Java for a while. Do you think a simple Java array question can be a challenge? Let's use the following problem to test.

Problem: Rotate an array of n elements to the right by k steps. For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4]. How many different ways do you know to solve this problem?

1.1 Solution 1 - Intermediate Array

In a straightforward way, we can create a new array and then copy elements to the new array. Then change the original array by using System.arraycopy().

```
public void rotate(int[] nums, int k) {
   if(k > nums.length)
      k=k%nums.length;

   int[] result = new int[nums.length];

   for(int i=0; i < k; i++){
      result[i] = nums[nums.length-k+i];
   }

   int j=0;
   for(int i=k; i<nums.length; i++){
      result[i] = nums[j];
      j++;
   }

   System.arraycopy( result, 0, nums, 0, nums.length );
}
```

Space is O(n) and time is O(n). You can check out the difference between System.arraycopy() and Arrays.copyOf().

1.2 Solution 2 - Bubble Rotate

Can we do this in O(1) space?

This solution is like a bubble sort.

```
public static void rotate(int[] arr, int order) {
  if (arr == null || order < 0) {
```

```
      throw new IllegalArgumentException("Illegal argument!");
    }

    for (int i = 0; i < order; i++) {
      for (int j = arr.length - 1; j > 0; j--) {
        int temp = arr[j];
        arr[j] = arr[j - 1];
        arr[j - 1] = temp;
      }
    }
}
```

However, the time is O(n*k).
Here is an example (length=7, order=3):

```
i=0
0 1 2 3 4 5 6
0 1 2 3 4 6 5
...
6 0 1 2 3 4 5
i=1
6 0 1 2 3 5 4
...
5 6 0 1 2 3 4
i=2
5 6 0 1 2 4 3
...
4 5 6 0 1 2 3
```

1.3 Solution 3 - Reversal

Can we do this in O(1) space and in O(n) time? The following solution does.
Assuming we are given 1,2,3,4,5,6 and order 2. The basic idea is:

```
1. Divide the array two parts: 1,2,3,4 and 5, 6
2. Reverse first part: 4,3,2,1,5,6
3. Reverse second part: 4,3,2,1,6,5
4. Reverse the whole array: 5,6,1,2,3,4
```

```
public static void rotate(int[] arr, int order) {
  if (arr == null || arr.length==0 || order < 0) {
    throw new IllegalArgumentException("Illegal argument!");
  }

  if(order > arr.length){
    order = order %arr.length;
  }
```

```
  //length of first part
  int a = arr.length - order;

  reverse(arr, 0, a-1);
  reverse(arr, a, arr.length-1);
  reverse(arr, 0, arr.length-1);

}

public static void reverse(int[] arr, int left, int right){
  if(arr == null || arr.length == 1)
    return;

  while(left < right){
    int temp = arr[left];
    arr[left] = arr[right];
    arr[right] = temp;
    left++;
    right--;
  }
}
```

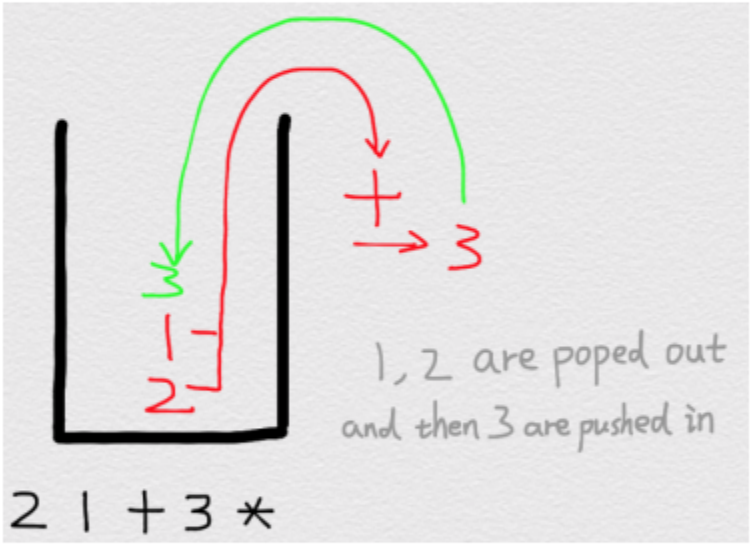
2 Evaluate Reverse Polish Notation

Evaluate the value of an arithmetic expression in Reverse Polish Notation. Valid operators are +, -, *, /. Each operand may be an integer or another expression. For example:

```
["2", "1", "+", "3", "*"] -> ((2 + 1) * 3) -> 9
  ["4", "13", "5", "/", "+"] -> (4 + (13 / 5)) -> 6
```

2.1 Naive Approach

This problem can be solved by using a stack. We can loop through each element in the given array. When it is a number, push it to the stack. When it is an operator, pop two numbers from the stack, do the calculation, and push back the result.



The following is the code. However, this code contains compilation errors in leetcode. Why?

```
public class Test {

  public static void main(String[] args) throws IOException {
    String[] tokens = new String[] { "2", "1", "+", "3", "*" };
    System.out.println(evalRPN(tokens));
  }
```

```
  public static int evalRPN(String[] tokens) {
    int returnValue = 0;
    String operators = "+-*/";

    Stack<String> stack = new Stack<String>();

    for (String t : tokens) {
      if (!operators.contains(t)) { //push to stack if it is a number
        stack.push(t);
      } else {//pop numbers from stack if it is an operator
        int a = Integer.valueOf(stack.pop());
        int b = Integer.valueOf(stack.pop());
        switch (t) {
        case "+":
          stack.push(String.valueOf(a + b));
          break;
        case "-":
          stack.push(String.valueOf(b - a));
          break;
        case "*":
          stack.push(String.valueOf(a * b));
          break;
        case "/":
          stack.push(String.valueOf(b / a));
          break;
        }
      }
    }

    returnValue = Integer.valueOf(stack.pop());

    return returnValue;
  }
}
```

The problem is that switch string statement is only available from JDK 1.7. Leetcode apparently use a JDK version below 1.7.

2.2 Accepted Solution

If you want to use switch statement, you can convert the above by using the following code which use the index of a string "+-*/".

```
public class Solution {
   public int evalRPN(String[] tokens) {

      int returnValue = 0;
```

```
        String operators = "+-*/";

        Stack<String> stack = new Stack<String>();

        for(String t : tokens){
           if(!operators.contains(t)){
              stack.push(t);
           }else{
              int a = Integer.valueOf(stack.pop());
              int b = Integer.valueOf(stack.pop());
              int index = operators.indexOf(t);
              switch(index){
                 case 0:
                    stack.push(String.valueOf(a+b));
                    break;
                 case 1:
                    stack.push(String.valueOf(b-a));
                    break;
                 case 2:
                    stack.push(String.valueOf(a*b));
                    break;
                 case 3:
                    stack.push(String.valueOf(b/a));
                    break;
              }
           }
        }

        returnValue = Integer.valueOf(stack.pop());

        return returnValue;

     }
}
```

3 Isomorphic Strings

Given two strings s and t, determine if they are isomorphic. Two strings are isomorphic if the characters in s can be replaced to get t.

For example,"egg" and "add" are isomorphic, "foo" and "bar" are not.

3.1 Analysis

We need to define a method which accepts a map & a value, and returns the value's key in the map.

3.2 Java Solution

```
public boolean isIsomorphic(String s, String t) {
   if(s==null || t==null)
      return false;

   if(s.length() != t.length())
      return false;

   if(s.length()==0 && t.length()==0)
      return true;

   HashMap<Character, Character> map = new HashMap<Character,Character>();
   for(int i=0; i<s.length(); i++){
      char c1 = s.charAt(i);
      char c2 = t.charAt(i);

      Character c = getKey(map, c2);
      if(c != null && c!=c1){
         return false;
      }else if(map.containsKey(c1)){
         if(c2 != map.get(c1))
            return false;
      }else{
         map.put(c1,c2);
      }
   }

   return true;
}
```

```
// a method for getting key of a target value
public Character getKey(HashMap<Character,Character> map, Character target){
   for (Map.Entry<Character,Character> entry : map.entrySet()) {
      if (entry.getValue().equals(target)) {
         return entry.getKey();
      }
   }

   return null;
}
```

4 Word Ladder

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that only one letter can be changed at a time and each intermediate word must exist in the dictionary. For example, given:

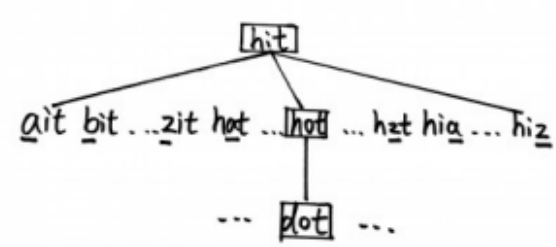
```
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
```

One shortest transformation is "hit" ->"hot" ->"dot" ->"dog" ->"cog", the program should return its length 5.

4.1 Analysis

UPDATED on 06/07/2015.

So we quickly realize that this is a search problem, and breath-first search guarantees the optimal solution.



4.2 Java Solution

```
class WordNode{
   String word;
   int numSteps;

   public WordNode(String word, int numSteps){
      this.word = word;
      this.numSteps = numSteps;
   }
}
```

```
public class Solution {
   public int ladderLength(String beginWord, String endWord, Set<String>
       wordDict) {
      LinkedList<WordNode> queue = new LinkedList<WordNode>();
      queue.add(new WordNode(beginWord, 1));

      wordDict.add(endWord);

      while(!queue.isEmpty()){
         WordNode top = queue.remove();
         String word = top.word;

         if(word.equals(endWord)){
            return top.numSteps;
         }

         char[] arr = word.toCharArray();
         for(int i=0; i<arr.length; i++){
            for(char c='a'; c<='z'; c++){
               char temp = arr[i];
               if(arr[i]!=c){
                  arr[i]=c;
               }

               String newWord = new String(arr);
               if(wordDict.contains(newWord)){
                  queue.add(new WordNode(newWord, top.numSteps+1));
                  wordDict.remove(newWord);
               }

               arr[i]=temp;
            }
         }
      }

      return 0;
   }
}
```

5 Word Ladder II

Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that: 1) Only one letter can be changed at a time, 2) Each intermediate word must exist in the dictionary.

For example, given: start = "hit", end = "cog", and dict = ["hot","dot","dog","lot","log"], return:

```
[
   ["hit","hot","dot","dog","cog"],
   ["hit","hot","lot","log","cog"]
 ]
```

5.1 Analysis

This is an extension of Word Ladder.

The idea is the same. To track the actual ladder, we need to add a pointer that points to the previous node in the WordNode class.

In addition, the used word can not directly removed from the dictionary. The used word is only removed when steps change.

5.2 Java Solution

```
class WordNode{
   String word;
   int numSteps;
   WordNode pre;

   public WordNode(String word, int numSteps, WordNode pre){
      this.word = word;
      this.numSteps = numSteps;
      this.pre = pre;
   }
}

public class Solution {
   public List<List<String>> findLadders(String start, String end,
       Set<String> dict) {
      List<List<String>> result = new ArrayList<List<String>>();
```

```
LinkedList<WordNode> queue = new LinkedList<WordNode>();
queue.add(new WordNode(start, 1, null));

dict.add(end);

int minStep = 0;

HashSet<String> visited = new HashSet<String>();
HashSet<String> unvisited = new HashSet<String>();
unvisited.addAll(dict);

int preNumSteps = 0;

while(!queue.isEmpty()){
   WordNode top = queue.remove();
   String word = top.word;
   int currNumSteps = top.numSteps;

   if(word.equals(end)){
      if(minStep == 0){
         minStep = top.numSteps;
      }

      if(top.numSteps == minStep && minStep !=0){
         //nothing
         ArrayList<String> t = new ArrayList<String>();
         t.add(top.word);
         while(top.pre !=null){
            t.add(0, top.pre.word);
            top = top.pre;
         }
         result.add(t);
         continue;
      }

   }

   if(preNumSteps < currNumSteps){
      unvisited.removeAll(visited);
   }

   preNumSteps = currNumSteps;

   char[] arr = word.toCharArray();
   for(int i=0; i<arr.length; i++){
      for(char c='a'; c<='z'; c++){
         char temp = arr[i];
         if(arr[i]!=c){
            arr[i]=c;
         }
```

```
                String newWord = new String(arr);
                if(unvisited.contains(newWord)){
                   queue.add(new WordNode(newWord, top.numSteps+1, top));
                   visited.add(newWord);
                }

                arr[i]=temp;
             }
          }


       }

       return result;
    }
}
```

6 Median of Two Sorted Arrays

There are two sorted arrays A and B of size m and n respectively. Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

6.1 Java Solution 1

If we see log(n), we should think about using binary something.

This problem can be converted to the problem of finding kth element, k is (A's length + B' Length)/2.

If any of the two arrays is empty, then the kth element is the non-empty array's kth element. If k == 0, the kth element is the first element of A or B.

For normal cases(all other cases), we need to move the pointer at the pace of half of an array length to get log(n) time.

```
public static double findMedianSortedArrays(int A[], int B[]) {
  int m = A.length;
  int n = B.length;

  if ((m + n) % 2 != 0) // odd
    return (double) findKth(A, B, (m + n) / 2, 0, m - 1, 0, n - 1);
  else { // even
    return (findKth(A, B, (m + n) / 2, 0, m - 1, 0, n - 1)
      + findKth(A, B, (m + n) / 2 - 1, 0, m - 1, 0, n - 1)) * 0.5;
  }
}

public static int findKth(int A[], int B[], int k,
  int aStart, int aEnd, int bStart, int bEnd) {

  int aLen = aEnd - aStart + 1;
  int bLen = bEnd - bStart + 1;

  // Handle special cases
  if (aLen == 0)
    return B[bStart + k];
  if (bLen == 0)
    return A[aStart + k];
  if (k == 0)
    return A[aStart] < B[bStart] ? A[aStart] : B[bStart];

  int aMid = aLen * k / (aLen + bLen); // a's middle count
  int bMid = k - aMid - 1; // b's middle count
```

```
  // make aMid and bMid to be array index
  aMid = aMid + aStart;
  bMid = bMid + bStart;

  if (A[aMid] > B[bMid]) {
    k = k - (bMid - bStart + 1);
    aEnd = aMid;
    bStart = bMid + 1;
  } else {
    k = k - (aMid - aStart + 1);
    bEnd = bMid;
    aStart = aMid + 1;
  }

  return findKth(A, B, k, aStart, aEnd, bStart, bEnd);
}
```

6.2 Java Solution 2

Solution 1 is a general solution to find the kth element. We can also come up with a simpler solution which only finds the median of two sorted arrays for this particular problem. Thanks to Gunner86. The description of the algorithm is awesome!

```
1) Calculate the medians m1 and m2 of the input arrays ar1[] and ar2[]
    respectively.
2) If m1 and m2 both are equal then we are done, and return m1 (or m2)
3) If m1 is greater than m2, then median is present in one of the below two
    subarrays.
a) From first element of ar1 to m1 (ar1[0...|_n/2_|])
b) From m2 to last element of ar2 (ar2[|_n/2_|...n-1])
4) If m2 is greater than m1, then median is present in one of the below two
    subarrays.
a) From m1 to last element of ar1 (ar1[|_n/2_|...n-1])
b) From first element of ar2 to m2 (ar2[0...|_n/2_|])
5) Repeat the above process until size of both the subarrays becomes 2.
6) If size of the two arrays is 2 then use below formula to get the median.
Median = (max(ar1[0], ar2[0]) + min(ar1[1], ar2[1]))/2
```

7 Kth Largest Element in an Array

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

For example, given [3,2,1,5,6,4] and k = 2, return 5.

Note: You may assume k is always valid, $1 \leq k \leq$ array's length.

7.1 Java Solution 1 - Sorting

```
public int findKthLargest(int[] nums, int k) {
   Arrays.sort(nums);
   return nums[nums.length-k];
}
```

Time is O(nlog(n))

7.2 Java Solution 2 - Quick Sort

This problem can also be solve by using the quickselect approach, which is similar to quicksort.

```
public int findKthLargest(int[] nums, int k) {
  if (k < 1 || nums == null) {
    return 0;
  }

  return getKth(nums.length - k +1, nums, 0, nums.length - 1);
}

public int getKth(int k, int[] nums, int start, int end) {

  int pivot = nums[end];

  int left = start;
  int right = end;

  while (true) {

    while (nums[left] < pivot && left < right) {
      left++;
    }
```

```
    while (nums[right] >= pivot && right > left) {
      right--;
    }

    if (left == right) {
      break;
    }

    swap(nums, left, right);
  }

  swap(nums, left, end);

  if (k == left + 1) {
    return pivot;
  } else if (k < left + 1) {
    return getKth(k, nums, start, left - 1);
  } else {
    return getKth(k, nums, left + 1, end);
  }
}

public void swap(int[] nums, int n1, int n2) {
  int tmp = nums[n1];
  nums[n1] = nums[n2];
  nums[n2] = tmp;
}
```

Average case time is O(n), worst case time is O(n^2).

7.3 Java Solution 3 - Heap

We can use a min heap to solve this problem. The heap stores the top k elements. Whenever the size is greater than k, delete the min. Time complexity is O(nlog(k)). Space complexity is O(k) for storing the top k numbers.

```
public int findKthLargest(int[] nums, int k) {
   PriorityQueue<Integer> q = new PriorityQueue<Integer>(k);
   for(int i: nums){
      q.offer(i);

      if(q.size()>k){
         q.poll();
      }
   }

   return q.peek();
}
```

8 Wildcard Matching

Implement wildcard pattern matching with support for '?' and '*'.

8.1 Java Solution

To understand this solution, you can use s="aab" and p="*ab".

```
public boolean isMatch(String s, String p) {
  int i = 0;
  int j = 0;
  int starIndex = -1;
  int iIndex = -1;

  while (i < s.length()) {
    if (j < p.length() && (p.charAt(j) == '?' || p.charAt(j) == s.charAt(i)))
        {
      ++i;
      ++j;
    } else if (j < p.length() && p.charAt(j) == '*') {
      starIndex = j;
      iIndex = i;
      j++;
    } else if (starIndex != -1) {
      j = starIndex + 1;
      i = iIndex+1;
      iIndex++;
    } else {
      return false;
    }
  }

  while (j < p.length() && p.charAt(j) == '*') {
    ++j;
  }

  return j == p.length();
}
```

9 Regular Expression Matching in Java

Implement regular expression matching with support for '.' and '*'.
'.' Matches any single character. '*' Matches zero or more of the preceding element.
The matching should cover the entire input string (not partial).
The function prototype should be: bool isMatch(const char *s, const char *p)
Some examples: isMatch("aa","a") return false isMatch("aa","aa") return true isMatch("aaa","aa") return false isMatch("aa", "a*") return true isMatch("aa", ".*") return true isMatch("ab", ".*") return true isMatch("aab", "c*a*b") return true

9.1 Analysis

First of all, this is one of the most difficulty problems. It is hard to think through all different cases. The problem should be simplified to handle 2 basic cases:

- the second char of pattern is "*"
- the second char of pattern is not "*"

For the 1st case, if the first char of pattern is not ".", the first char of pattern and string should be the same. Then continue to match the remaining part.
For the 2nd case, if the first char of pattern is "." or first char of pattern == the first i char of string, continue to match the remaining part.

9.2 Java Solution 1 (Short)

The following Java solution is accepted.

```
public class Solution {
   public boolean isMatch(String s, String p) {

      if(p.length() == 0)
         return s.length() == 0;

      //p's length 1 is special case
      if(p.length() == 1 || p.charAt(1) != '*'){
         if(s.length() < 1 || (p.charAt(0) != '.' && s.charAt(0) !=
             p.charAt(0)))
            return false;
         return isMatch(s.substring(1), p.substring(1));

      }else{
```

```
            int len = s.length();

            int i = -1;
            while(i<len && (i < 0 || p.charAt(0) == '.' || p.charAt(0) ==
                s.charAt(i))){
               if(isMatch(s.substring(i+1), p.substring(2)))
                  return true;
               i++;
            }
            return false;
         }
      }
}
```

9.3 Java Solution 2 (More Readable)

```
public boolean isMatch(String s, String p) {
  // base case
  if (p.length() == 0) {
    return s.length() == 0;
  }

  // special case
  if (p.length() == 1) {

    // if the length of s is 0, return false
    if (s.length() < 1) {
      return false;
    }

    //if the first does not match, return false
    else if ((p.charAt(0) != s.charAt(0)) && (p.charAt(0) != '.')) {
      return false;
    }

    // otherwise, compare the rest of the string of s and p.
    else {
      return isMatch(s.substring(1), p.substring(1));
    }
  }

  // case 1: when the second char of p is not '*'
  if (p.charAt(1) != '*') {
    if (s.length() < 1) {
      return false;
    }
    if ((p.charAt(0) != s.charAt(0)) && (p.charAt(0) != '.')) {
```

```
        return false;
      } else {
        return isMatch(s.substring(1), p.substring(1));
      }
    }

    // case 2: when the second char of p is '*', complex case.
    else {
      //case 2.1: a char & '*' can stand for 0 element
      if (isMatch(s, p.substring(2))) {
        return true;
      }

      //case 2.2: a char & '*' can stand for 1 or more preceding element,
      //so try every sub string
      int i = 0;
      while (i<s.length() && (s.charAt(i)==p.charAt(0) || p.charAt(0)=='.')){
        if (isMatch(s.substring(i + 1), p.substring(2))) {
          return true;
        }
        i++;
      }
      return false;
    }
}
```

10 Merge Intervals

Problem:

```
Given a collection of intervals, merge all overlapping intervals.

For example,
Given [1,3],[2,6],[8,10],[15,18],
return [1,6],[8,10],[15,18].
```

10.1 Thoughts of This Problem

The key to solve this problem is defining a Comparator first to sort the arraylist of Intevals. And then merge some intervals.

The take-away message from this problem is utilizing the advantage of sorted list/array.

10.2 Java Solution

```
class Interval {
  int start;
  int end;

  Interval() {
    start = 0;
    end = 0;
  }

  Interval(int s, int e) {
    start = s;
    end = e;
  }
}

public class Solution {
  public ArrayList<Interval> merge(ArrayList<Interval> intervals) {

    if (intervals == null || intervals.size() <= 1)
      return intervals;

    // sort intervals by using self-defined Comparator
```

```
    Collections.sort(intervals, new IntervalComparator());

    ArrayList<Interval> result = new ArrayList<Interval>();

    Interval prev = intervals.get(0);
    for (int i = 1; i < intervals.size(); i++) {
      Interval curr = intervals.get(i);

      if (prev.end >= curr.start) {
        // merged case
        Interval merged = new Interval(prev.start, Math.max(prev.end,
            curr.end));
        prev = merged;
      } else {
        result.add(prev);
        prev = curr;
      }
    }

    result.add(prev);

    return result;
  }
}

class IntervalComparator implements Comparator<Interval> {
  public int compare(Interval i1, Interval i2) {
    return i1.start - i2.start;
  }
}
```

11 Insert Interval

Problem:

Given a set of non-overlapping & sorted intervals, insert a new interval into the intervals (merge if necessary).

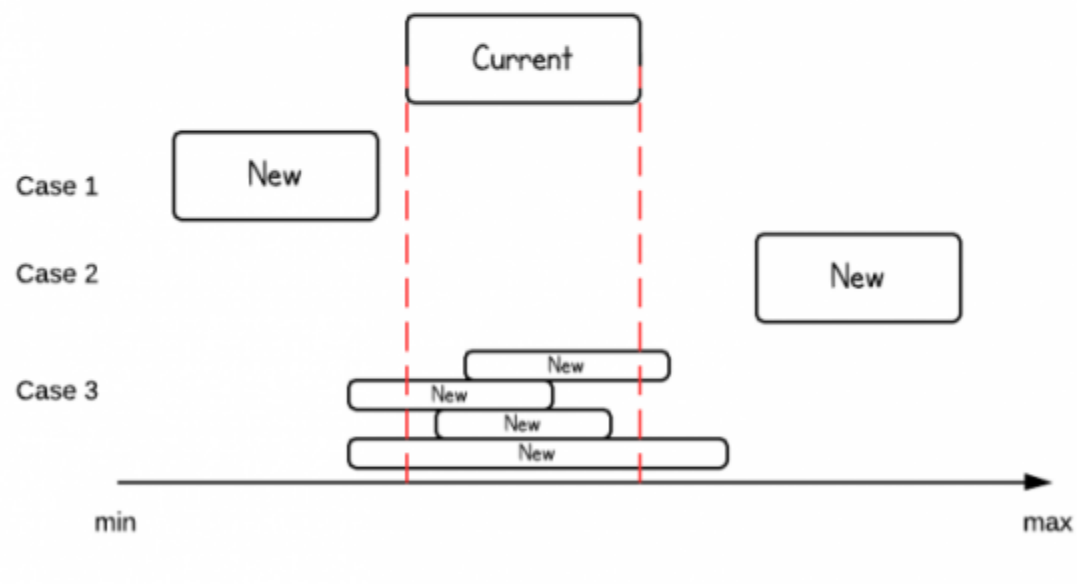
```
Example 1:
Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].

Example 2:
Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] in as
    [1,2],[3,10],[12,16].

This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10].
```

11.1 Thoughts of This Problem

Quickly summarize 3 cases. Whenever there is intersection, created a new interval.



11.2 Java Solution

```java
/**
 * Definition for an interval.
 * public class Interval {
 *    int start;
 *    int end;
 *    Interval() { start = 0; end = 0; }
 *    Interval(int s, int e) { start = s; end = e; }
 * }
 */
public class Solution {
   public ArrayList<Interval> insert(ArrayList<Interval> intervals, Interval
       newInterval) {

      ArrayList<Interval> result = new ArrayList<Interval>();

      for(Interval interval: intervals){
         if(interval.end < newInterval.start){
            result.add(interval);
         }else if(interval.start > newInterval.end){
            result.add(newInterval);
            newInterval = interval;
         }else if(interval.end >= newInterval.start || interval.start <=
             newInterval.end){
            newInterval = new Interval(Math.min(interval.start,
                newInterval.start), Math.max(newInterval.end, interval.end));
         }
      }

      result.add(newInterval);

      return result;
   }
}
```

14 Two Sum III Data structure design

Design and implement a TwoSum class. It should support the following operations: add and find.

add - Add the number to an internal data structure. find - Find if there exists any pair of numbers which sum is equal to the value.

For example,

```
add(1);
add(3);
add(5);
find(4) -> true
find(7) -> false
```

14.1 Java Solution

Since the desired class need add and get operations, HashMap is a good option for this purpose.

```
public class TwoSum {
  private HashMap<Integer, Integer> elements = new HashMap<Integer,
      Integer>();

  public void add(int number) {
    if (elements.containsKey(number)) {
      elements.put(number, elements.get(number) + 1);
    } else {
      elements.put(number, 1);
    }
  }

  public boolean find(int value) {
    for (Integer i : elements.keySet()) {
      int target = value - i;
      if (elements.containsKey(target)) {
        if (i == target && elements.get(target) < 2) {
          continue;
        }
        return true;
      }
    }
    return false;



  }
}
```

15 3Sum

Problem:

Given an array S of n integers, are there elements a, b, c in S such that a + b + c = o? Find all unique triplets in the array which gives the sum of zero.

Note: Elements in a triplet (a,b,c) must be in non-descending order. (ie, a $\leq$ b $\leq$ c) The solution set must not contain duplicate triplets.

```
For example, given array S = {-1 0 1 2 -1 -4},

   A solution set is:
   (-1, 0, 1)
   (-1, -1, 2)
```

15.1 Naive Solution

Naive solution is 3 loops, and this gives time complexity O(n3). Apparently this is not an acceptable solution, but a discussion can start from here.

```
public class Solution {
   public ArrayList<ArrayList<Integer>> threeSum(int[] num) {
      //sort array
      Arrays.sort(num);

      ArrayList<ArrayList<Integer>> result = new
          ArrayList<ArrayList<Integer>>();
      ArrayList<Integer> each = new ArrayList<Integer>();
      for(int i=0; i<num.length; i++){
         if(num[i] > 0) break;

         for(int j=i+1; j<num.length; j++){
            if(num[i] + num[j] > 0 && num[j] > 0) break;

            for(int k=j+1; k<num.length; k++){
             if(num[i] + num[j] + num[k] == 0) {

                each.add(num[i]);
                each.add(num[j]);
                each.add(num[k]);
                result.add(each);
                each.clear();
             }
```

```
            }
          }
        }

        return result;
    }
}
```

* The solution also does not handle duplicates. Therefore, it is not only time inefficient, but also incorrect.

Result:

```
Submission Result: Output Limit Exceeded
```

15.2 Better Solution

A better solution is using two pointers instead of one. This makes time complexity of O(n2).

To avoid duplicate, we can take advantage of sorted arrays, i.e., move pointers by >1 to use same element only once.

```
public ArrayList<ArrayList<Integer>> threeSum(int[] num) {
  ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();

  if (num.length < 3)
    return result;

  // sort array
  Arrays.sort(num);

  for (int i = 0; i < num.length - 2; i++) {
    // //avoid duplicate solutions
    if (i == 0 || num[i] > num[i - 1]) {

      int negate = -num[i];

      int start = i + 1;
      int end = num.length - 1;

      while (start < end) {
        //case 1
        if (num[start] + num[end] == negate) {
          ArrayList<Integer> temp = new ArrayList<Integer>();
          temp.add(num[i]);
          temp.add(num[start]);
          temp.add(num[end]);

          result.add(temp);
```

```
            start++;
            end--;
            //avoid duplicate solutions
            while (start < end && num[end] == num[end + 1])
              end--;

            while (start < end && num[start] == num[start - 1])
              start++;
          //case 2
          } else if (num[start] + num[end] < negate) {
            start++;
          //case 3
          } else {
            end--;
          }
        }

      }
    }

    return result;
}
```

16 4Sum

Given an array S of n integers, are there elements a, b, c, and d in S such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note: Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, a $\leq$ b $\leq$ c $\leq$ d) The solution set must not contain duplicate quadruplets.

```
For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

   A solution set is:
   (-1, 0, 0, 1)
   (-2, -1, 1, 2)
   (-2, 0, 0, 2)
```

16.1 Thoughts

A typical k-sum problem. Time is N to the power of (k-1).

16.2 Java Solution

```
public ArrayList<ArrayList<Integer>> fourSum(int[] num, int target) {
  Arrays.sort(num);

  HashSet<ArrayList<Integer>> hashSet = new HashSet<ArrayList<Integer>>();
  ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();

  for (int i = 0; i < num.length; i++) {
    for (int j = i + 1; j < num.length; j++) {
      int k = j + 1;
      int l = num.length - 1;

      while (k < l) {
        int sum = num[i] + num[j] + num[k] + num[l];

        if (sum > target) {
          l--;
        } else if (sum < target) {
          k++;
        } else if (sum == target) {
          ArrayList<Integer> temp = new ArrayList<Integer>();
          temp.add(num[i]);
```

```
            temp.add(num[j]);
            temp.add(num[k]);
            temp.add(num[l]);

            if (!hashSet.contains(temp)) {
              hashSet.add(temp);
              result.add(temp);
            }

            k++;
            l--;
          }
        }
      }
    }

    return result;
}
```

Here is the hashCode method of ArrayList. It makes sure that if all elements of two lists are the same, then the hash code of the two lists will be the same. Since each element in the ArrayList is Integer, same integer has same hash code.

```
int hashCode = 1;
Iterator<E> i = list.iterator();
while (i.hasNext()) {
     E obj = i.next();
     hashCode = 31*hashCode + (obj==null ? 0 : obj.hashCode());
}
```

17 3Sum Closest

Given an array S of n integers, find three integers in S such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

```
For example, given array S = {-1 2 1 -4}, and target = 1.
The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

17.1 Analysis

This problem is similar to 2 Sum. This kind of problem can be solved by using a similar approach, i.e., two pointers from both left and right.

17.2 Java Solution

```
public int threeSumClosest(int[] nums, int target) {
   int min = Integer.MAX_VALUE;
  int result = 0;

  Arrays.sort(nums);

  for (int i = 0; i < nums.length; i++) {
    int j = i + 1;
    int k = nums.length - 1;
    while (j < k) {
      int sum = nums[i] + nums[j] + nums[k];
      int diff = Math.abs(sum - target);

      if(diff == 0) return sum;

      if (diff < min) {
        min = diff;
        result = sum;
      }
      if (sum <= target) {
        j++;
      } else {
        k--;
      }
    }


  }

  return result;
}
```

Time Complexity is O(n2).

18 String to Integer (atoi)

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

18.1 Analysis

The following cases should be considered for this problem:

```
1. null or empty string
2. white spaces
3. +/- sign
4. calculate real value
5. handle min & max
```

18.2 Java Solution

```
public int atoi(String str) {
  if (str == null || str.length() < 1)
    return 0;

  // trim white spaces
  str = str.trim();

  char flag = '+';

  // check negative or positive
  int i = 0;
  if (str.charAt(0) == '-') {
    flag = '-';
    i++;
  } else if (str.charAt(0) == '+') {
    i++;
  }
  // use double to store result
  double result = 0;

  // calculate value
  while (str.length() > i && str.charAt(i) >= '0' && str.charAt(i) <= '9') {
    result = result * 10 + (str.charAt(i) - '0');
```

```
      i++;
    }

    if (flag == '-')
      result = -result;

    // handle max and min
    if (result > Integer.MAX_VALUE)
      return Integer.MAX_VALUE;

    if (result < Integer.MIN_VALUE)
      return Integer.MIN_VALUE;

    return (int) result;
  }
```

Curated By: Sumit Soni | LinkedIn

Credits: Program Creek