



Algorithm

1. Preface
2. Part I - Basics
3. Basics Data Structure
 - i. Linked List
 - ii. Binary Tree
 - iii. Binary Search Tree
 - iv. Huffman Compression
 - v. Priority Queue
4. Basics Sorting
 - i. Bubble Sort
 - ii. Selection Sort
 - iii. Insertion Sort
 - iv. Merge Sort
 - v. Quick Sort
 - vi. Heap Sort
 - vii. Bucket Sort
 - viii. Counting Sort
 - ix. Radix Sort
5. Basics Misc
 - i. Bit Manipulation
 - ii. Knapsack
6. Part II - Coding
7. String
 - i. strStr
 - ii. Two Strings Are Anagrams
 - iii. Compare Strings
 - iv. Anagrams
 - v. Longest Common Substring
 - vi. Rotate String
 - vii. Reverse Words in a String
 - viii. Valid Palindrome
 - ix. Longest Palindromic Substring
 - x. Space Replacement
 - xi. Wildcard Matching
 - xii. Length of Last Word
 - xiii. Count and Say
8. Integer Array
 - i. Remove Element
 - ii. Zero Sum Subarray
 - iii. Subarray Sum K
 - iv. Subarray Sum Closest
 - v. Recover Rotated Sorted Array

- vi. [Product of Array Exclude Itself](#)
 - vii. [Partition Array](#)
 - viii. [First Missing Positive](#)
 - ix. [2 Sum](#)
 - x. [3 Sum](#)
 - xi. [3 Sum Closest](#)
 - xii. [Remove Duplicates from Sorted Array](#)
 - xiii. [Remove Duplicates from Sorted Array II](#)
 - xiv. [Merge Sorted Array](#)
 - xv. [Merge Sorted Array II](#)
 - xvi. [Median](#)
 - xvii. [Partition Array by Odd and Even](#)
 - xviii. [Kth Largest Element](#)
9. [Binary Search](#)
- i. [Binary Search](#)
 - ii. [Search Insert Position](#)
 - iii. [Search for a Range](#)
 - iv. [First Bad Version](#)
 - v. [Search a 2D Matrix](#)
 - vi. [Find Peak Element](#)
 - vii. [Search in Rotated Sorted Array](#)
 - viii. [Find Minimum in Rotated Sorted Array](#)
 - ix. [Search a 2D Matrix II](#)
 - x. [Median of two Sorted Arrays](#)
 - xi. [Sqrt x](#)
 - xii. [Wood Cut](#)
10. [Math and Bit Manipulation](#)
- i. [Single Number](#)
 - ii. [Single Number II](#)
 - iii. [Single Number III](#)
 - iv. [O1 Check Power of 2](#)
 - v. [Convert Integer A to Integer B](#)
 - vi. [Factorial Trailing Zeroes](#)
 - vii. [Unique Binary Search Trees](#)
 - viii. [Update Bits](#)
 - ix. [Fast Power](#)
 - x. [Count 1 in Binary](#)
 - xi. [Fibonacci](#)
 - xii. [A plus B Problem](#)
 - xiii. [Print Numbers by Recursion](#)
 - xiv. [Majority Number](#)
 - xv. [Majority Number II](#)
 - xvi. [Majority Number III](#)
 - xvii. [Digit Counts](#)
 - xviii. [Ugly Number](#)
11. [Linked List](#)

- i. Remove Duplicates from Sorted List
- ii. Remove Duplicates from Sorted List II
- iii. Remove Duplicates from Unsorted List
- iv. Partition List
- v. Two Lists Sum
- vi. Two Lists Sum Advanced
- vii. Remove Nth Node From End of List
- viii. Linked List Cycle
- ix. Linked List Cycle II
- x. Reverse Linked List
- xi. Reverse Linked List II
- xii. Merge Two Sorted Lists
- xiii. Merge k Sorted Lists
- xiv. Reorder List
- xv. Copy List with Random Pointer
- xvi. Sort List
- xvii. Insertion Sort List
- xviii. Check if a singly linked list is palindrome
- xix. Delete Node in the Middle of Singly Linked List

12. Binary Tree

- i. Binary Tree Preorder Traversal
- ii. Binary Tree Inorder Traversal
- iii. Binary Tree Postorder Traversal
- iv. Binary Tree Level Order Traversal
- v. Binary Tree Level Order Traversal II
- vi. Maximum Depth of Binary Tree
- vii. Balanced Binary Tree
- viii. Binary Tree Maximum Path Sum
- ix. Lowest Common Ancestor
- x. Invert Binary Tree
- xi. Diameter of a Binary Tree
- xii. Construct Binary Tree from Preorder and Inorder Traversal
- xiii. Construct Binary Tree from Inorder and Postorder Traversal
- xiv. Subtree
- xv. Binary Tree Zigzag Level Order Traversal
- xvi. Binary Tree Serialization

13. Binary Search Tree

- i. Insert Node in a Binary Search Tree
- ii. Validate Binary Search Tree
- iii. Search Range in Binary Search Tree
- iv. Convert Sorted Array to Binary Search Tree
- v. Convert Sorted List to Binary Search Tree
- vi. Binary Search Tree Iterator

14. Exhaustive Search

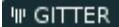
- i. Subsets
- ii. Unique Subsets

- iii. [Permutations](#)
- iv. [Unique Permutations](#)
- v. [Next Permutation](#)
- vi. [Previous Permuuation](#)
- vii. [Unique Binary Search Trees II](#)
- viii. [Permutation Index](#)
- ix. [Permutation Index II](#)
- x. [Permutation Sequence](#)
- xi. [Palindrome Partitioning](#)
- xii. [Combinations](#)
- xiii. [Combination Sum](#)
- xiv. [Combination Sum II](#)
- 15. [Dynamic Programming](#)
 - i. [Triangle](#)
 - ii. [Backpack](#)
 - iii. [Minimum Path Sum](#)
 - iv. [Unique Paths](#)
 - v. [Unique Paths II](#)
 - vi. [Climbing Stairs](#)
 - vii. [Jump Game](#)
 - viii. [Word Break](#)
 - ix. [Longest Increasing Subsequence](#)
 - x. [Palindrome Partitioning II](#)
 - xi. [Longest Common Subsequence](#)
 - xii. [Edit Distance](#)
 - xiii. [Jump Game II](#)
 - xiv. [Best Time to Buy and Sell Stock](#)
 - xv. [Best Time to Buy and Sell Stock II](#)
 - xvi. [Best Time to Buy and Sell Stock III](#)
 - xvii. [Best Time to Buy and Sell Stock IV](#)
 - xviii. [Distinct Subsequences](#)
 - xix. [Interleaving String](#)
 - xx. [Maximum Subarray](#)
 - xxi. [Maximum Subarray II](#)
- 16. [Graph](#)
 - i. [Topological Sorting](#)
- 17. [Data Structure](#)
 - i. [Implement Queue by Two Stacks](#)
 - ii. [Min Stack](#)
 - iii. [Sliding Window Maximum](#)
 - iv. [Longest Words](#)
- 18. [Problem Misc](#)
 - i. [Nuts and Bolts Problem](#)
 - ii. [Heapify](#)
 - iii. [String to Integer](#)
- 19. [Appendix I Interview and Resume](#)

- i. [Interview](#)
- ii. [Resume](#)

20.

/leetcode/lintcode

 GITTER [JOIN CHAT →](#) build passing

-
1. Part I//
 2. Part II OJ <https://leetcode.com/> <http://www.lintcode.com/>.
 3. Part III

issue :)

<https://github.com/billryan/algorithm-exercise> [Gitbook](#) [HTML](#) [GitHub](#) [star](#) [RSS](#)

/~

- (Gitbook) <http://algorithm.yuanbin.me>
- : GitHub travis-ci
 1. EPUB. [Gitbook](#) - iPhone/iPad/MAC
 2. PDF. [Gitbook](#), - , - - Gitbook
 3. MOBI. [Gitbook](#) - Kindle . Kindle Kindle ...
- Google :
- Swiftype :

- 4.0 <http://creativecommons.org/licenses/by-sa/4.0/>

/ .

-
- - (Google)
 1. ()

- 2. ()
- 3. ()
- 4. (if)

•

- 1.
- 2.
- 3.

OJ

- [LeetCode Online Judge](#) - OJ discuss lintcode OJ
- [LintCode | Coding interview questions online training system](#) - leetcodeOJ source
- CC150 locale OJ leetcode
- [leetcode/lintcode/ | billryan](#) - CS
- [LeetCode - GitBook](#) -
- [FreeTymeKiyan/LeetCode-Sol-Res](#) - Clean, Understandable Solutions and Resources on LeetCode Online Judge Algorithms Problems.
- [soulmachine/leetcode](#) - C++Java
- [Woodstock Blog](#) - IT
- [ITint5 | IT](#) -
- [Acm,ACM](#) -
- [-IT,,IT](#) - IT :)

-
- [|](#) -
 - [- julyedu.com](#) - july
 - [|](#) -
 - [VisuAlgo](#) - visualising data structures and algorithms through animation -
 - [Data Structure Visualization](#) - //
 - -
 - [julycoding/The-Art-Of-Programming-By-July](#) -
 - 5
 - [——](#) - CSDN
 - [- Lucida](#) - Google
-

- [Algorithm Design \(\)](#)
- [The Algorithm Design Manual](#), slides - Skiena's Audio Lectures
- [The Algorithm Design Manual \(\)](#)
- [*Introduction to Algorithm* TAOCP](#)
- [*Cracking The Coding Interview*. CTCI\(CC150\)](#) Google, Microsoft, LinkedIn HROO
Design, Database, System Design, Brain Teaser
- [*OfferCoding Interviews*. \(Harry He\)Amazon50show](#)
- -- 150

Part I - Basics

Reference

- VisuAlgo - visualising data structures and algorithms through animation - [/visualgo](#)
- Data Structure Visualization - [/ds-visualizations](#)

Data Structure -

Linked List -

O(1) O(n)

O(1)O(1)

```
1 -> 2 -> 3 -> null    3 -> 2 -> 1 -> null
```

- curt.next curt null
- null

```
public ListNode reverse(ListNode head) {  
    ListNode prev = null;  
    while (head != null) {  
        ListNode next = head.next;  
        head.next = prev;  
        prev = head;  
        head = next;  
    }  
    return prev;  
}
```

```
prev -> next = prev -> next -> next  Dummy Node
```

-
- curt.next curt null
 - null

Dummy Node

Dummy node “”””

Dummy node Dummy node head head dummy -> head
dummy node head Remove Duplicates From Sorted List || current = current.next
head dummy node head

head dummy node dummy.next

21

- *fast *slow *fast *slow 2 *fast slow
- *fast *slow *fast *slow 2 *fast = NUL
- *slow

Binary Tree -

$$\text{# of nodes } k = 2^i - 1 \quad n_0, 2^{i-1}, n_2, \quad n_0 = n_2 + 1$$

$$k, \quad 2^k - 1 \quad k \quad n \quad k \quad 1 \quad n$$

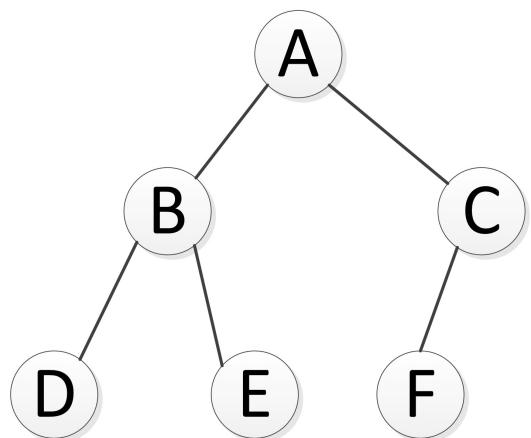
“”

0

- 1. (pre-order)
 2. (in-order)
 3. (post-order)
-

A

||||



pre-order: A BDE CF
in-order: DBE A FC
post-order: DEB FC A
level-order: A BC DEF

LeetCode

```

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
  
```

pre-order traverse

——Divide & Conquer)

“”

-

- 1.
- 2.
- 3.
- 4.

-

1. Divide
2. Conquer
3. Combine

-

- 1.
- 2.
3. Strassen
- 4.
- 5.
- 6.
- 7.
- 8.

Binary Search Tree -

(BST)

0

-
-
-
-

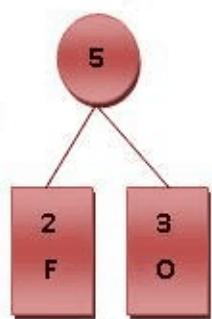
Huffman Compression -

78

Huffman

F, O, R, G, E, T

Symbol	F	O	R	G	E	T
Frequence	2	3	4	4	5	7
Code	000	001	100	101	01	10

[Huffman](#) | - CoolShell.cn

-

Priority Queue -

Reference

- [-](#)

Basics Sorting -

1sO10000000100,000,000

1. -0

2. -

◦

◦

◦

0

- Comparison Sorting

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Shell Sort
5. Merge Sort
6. Quicksort
7. Heapsort

- Bucket Sort

- Counting Sort

- Radix Sort

•

•

Reference

- + - SegmentFault
- [Sorting algorithm - Wikipedia, the free encyclopedia](#) -
- [Jark's Blog](#) - Python
- [Startup_](#) -

Bubble Sort -

6 5 3 1 8 7 2 4

Implementation

Python

```
#!/usr/bin/env python

def bubbleSort(alist):
    for i in xrange(len(alist)):
        print(alist)
        for j in xrange(1, len(alist) - i):
            if alist[j - 1] > alist[j]:
                alist[j - 1], alist[j] = alist[j], alist[j - 1]

    return alist

unsorted_list = [6, 5, 3, 1, 8, 7, 2, 4]
print(bubbleSort(unsorted_list))
```

Java

```
public class Sort {
    public static void main(String[] args) {
        int unsortedArray[] = new int[]{6, 5, 3, 1, 8, 7, 2, 4};
        bubbleSort(unsortedArray);
        System.out.println("After sort: ");
        for (int item : unsortedArray) {
            System.out.print(item + " ");
        }
    }

    public static void bubbleSort(int[] array) {
        int len = array.length;
        for (int i = 0; i < len; i++) {
            for (int item : array) {
```

```
        System.out.print(item + " ");
    }
    System.out.println();
    for (int j = 1; j < len - i; j++) {
        if (array[j - 1] > array[j]) {
            int temp = array[j - 1];
            array[j - 1] = array[j];
            array[j] = temp;
        }
    }
}
```

$O(n^2)$, temp $O(1)$.

Reference

- -

Selection Sort -

1.

2.

- $=(N-1)+(N-2)+(N-3)+\dots+2+1 \sim N^2/2$
- $=N$
-
-

[File:Selection-Sort-Animation.gif](#) - IB Computer Science

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Implementation

Python

```
#!/usr/bin/env python

def selectionSort(alist):
    for i in xrange(len(alist)):
        print(alist)
        min_index = i
        for j in xrange(i + 1, len(alist)):
            if alist[j] < alist[min_index]:
```

```

        min_index = j
    alist[min_index], alist[i] = alist[i], alist[min_index]
return alist

unsorted_list = [8, 5, 2, 6, 9, 3, 1, 4, 0, 7]
print(selectionSort(unsorted_list))

```

Java

```

public class Sort {
    public static void main(String[] args) {
        int unsortedArray[] = new int[]{8, 5, 2, 6, 9, 3, 1, 4, 0, 7};
        selectionSort(unsortedArray);
        System.out.println("After sort: ");
        for (int item : unsortedArray) {
            System.out.print(item + " ");
        }
    }

    public static void selectionSort(int[] array) {
        int len = array.length;
        for (int i = 0; i < len; i++) {
            for (int item : array) {
                System.out.print(item + " ");
            }
            System.out.println();
            int min_index = i;
            for (int j = i + 1; j < len; j++) {
                if (array[j] < array[min_index]) {
                    min_index = j;
                }
            }
            int temp = array[min_index];
            array[min_index] = array[i];
            array[i] = temp;
        }
    }
}

```

Reference

- -
- [The Selection Sort — Problem Solving with Algorithms and Data Structures](#)

Insertion Sort -

()in-place($O(1)$)

- 1.
- 2.
- 3.
4. 3
- 5.
6. 2~5

-
- $>= <=$
- 0
- $1N-1i(a[i])$
- $\sim N^2/2 \sim N^2/2N-10$
- $\sim N^2/4 \sim N^2/4$

6 5 3 1 8 7 2 4

Implementation

Python

```
#!/usr/bin/env python

def insertionSort(alist):
    for i, item_i in enumerate(alist):
        print alist
        index = i
```

```

        while index > 0 and alist[index - 1] > item_i:
            alist[index] = alist[index - 1]
            index -= 1

    alist[index] = item_i

    return alist

unsorted_list = [6, 5, 3, 1, 8, 7, 2, 4]
print(insertionSort(unsorted_list))

```

Java

```

public class Sort {
    public static void main(String[] args) {
        int unsortedArray[] = new int[]{6, 5, 3, 1, 8, 7, 2, 4};
        insertionSort(unsortedArray);
        System.out.println("After sort: ");
        for (int item : unsortedArray) {
            System.out.print(item + " ");
        }
    }

    public static void insertionSort(int[] array) {
        int len = array.length;
        for (int i = 0; i < len; i++) {
            int index = i, array_i = array[i];
            while (index > 0 && array[index - 1] > array_i) {
                array[index] = array[index - 1];
                index -= 1;
            }
            array[index] = array_i;

            /* print sort process */
            for (int item : array) {
                System.out.print(item + " ");
            }
            System.out.println();
        }
    }
}

```

(C++)

```

template<typename T>
void insertion_sort(T arr[], int len) {
    int i, j;
    T temp;
    for (int i = 1; i < len; i++) {
        temp = arr[i];
        for (int j = i - 1; j >= 0 && arr[j] > temp; j--) {
            a[j + 1] = a[j];
        }
        arr[j + 1] = temp;
    }
}

```

}

hh

(C++):

```
template<typename T>
void shell_sort(T arr[], int len) {
    int gap, i, j;
    T temp;
    for (gap = len >> 1; gap > 0; gap >>= 1)
        for (i = gap; i < len; i++) {
            temp = arr[i];
            for (j = i - gap; j >= 0 && arr[j] > temp; j -= gap)
                arr[j + gap] = arr[j];
            arr[j + gap] = temp;
        }
}
```

Reference

- -
- -
- [The Insertion Sort — Problem Solving with Algorithms and Data Structures](#)

Merge Sort -

6 5 3 1 8 7 2 4

Python

```
#!/usr/bin/env python

class Sort:
    def mergeSort(self, alist):
        if len(alist) <= 1:
            return alist

        mid = len(alist) / 2
        left = self.mergeSort(alist[:mid])
        print("left = " + str(left))
        right = self.mergeSort(alist[mid:])
        print("right = " + str(right))
        return self.mergeSortedArray(left, right)

    #@param A and B: sorted integer array A and B.
    #@return: A new sorted integer array
    def mergeSortedArray(self, A, B):
        sortedArray = []
        l = 0
        r = 0
        while l < len(A) and r < len(B):
            if A[l] < B[r]:
                sortedArray.append(A[l])
                l += 1
            else:
                sortedArray.append(B[r])
                r += 1
        sortedArray += A[l:]
        sortedArray += B[r:]

        return sortedArray

unsortedArray = [6, 5, 3, 1, 8, 7, 2, 4]
merge_sort = Sort()
```

```
print(merge_sort.mergeSort(unsortedArray))
```

Java

```
public class MergeSort {
    public static void main(String[] args) {
        int unsortedArray[] = new int[]{6, 5, 3, 1, 8, 7, 2, 4};
        mergeSort(unsortedArray);
        System.out.println("After sort: ");
        for (int item : unsortedArray) {
            System.out.print(item + " ");
        }
    }

    private static void merge(int[] array, int low, int mid, int high) {
        int[] helper = new int[array.length];
        // copy array to helper
        for (int k = low; k <= high; k++) {
            helper[k] = array[k];
        }
        // merge array[low...mid] and array[mid + 1...high]
        int i = low, j = mid + 1;
        for (int k = low; k <= high; k++) {
            // k means current location
            if (i > mid) {
                // no item in left part
                array[k] = helper[j];
                j++;
            } else if (j > high) {
                // no item in right part
                array[k] = helper[i];
                i++;
            } else if (helper[i] > helper[j]) {
                // get smaller item in the right side
                array[k] = helper[j];
                j++;
            } else {
                // get smaller item in the left side
                array[k] = helper[i];
                i++;
            }
        }
    }

    public static void sort(int[] array, int low, int high) {
        if (high <= low) return;
        int mid = low + (high - low) / 2;
        sort(array, low, mid);
        sort(array, mid + 1, high);
        merge(array, low, mid, high);
        for (int item : array) {
            System.out.print(item + " ");
        }
    }
}
```

```
    System.out.println();
}

public static void mergeSort(int[] array) {
    sort(array, 0, array.length - 1);
}
}
```

$O(N \log N)$, $O(N)$

Reference

- [Mergesort](#) - Robert Sedgewick

Quick Sort -

1. ——
2. ——
3. ——

out-in-place -

Python list comprehension

```
#!/usr/bin/env python

def qsort1(alist):
    print(alist)
    if len(alist) <= 1:
        return alist
    else:
        pivot = alist[0]
        return qsort1([x for x in alist[1:] if x < pivot]) + \
            [pivot] + \
            qsort1([x for x in alist[1:] if x >= pivot])

unsortedArray = [6, 5, 3, 1, 8, 7, 2, 4]
print(qsort1(unsortedArray))
```

```
[6, 5, 3, 1, 8, 7, 2, 4]
[5, 3, 1, 2, 4]
[3, 1, 2, 4]
[1, 2]
[]
[2]
[4]
[]
[8, 7]
[7]
[]
[1, 2, 3, 4, 5, 6, 7, 8]
```

+ list comprehension

$$\log n, \quad n, \quad O(n)$$

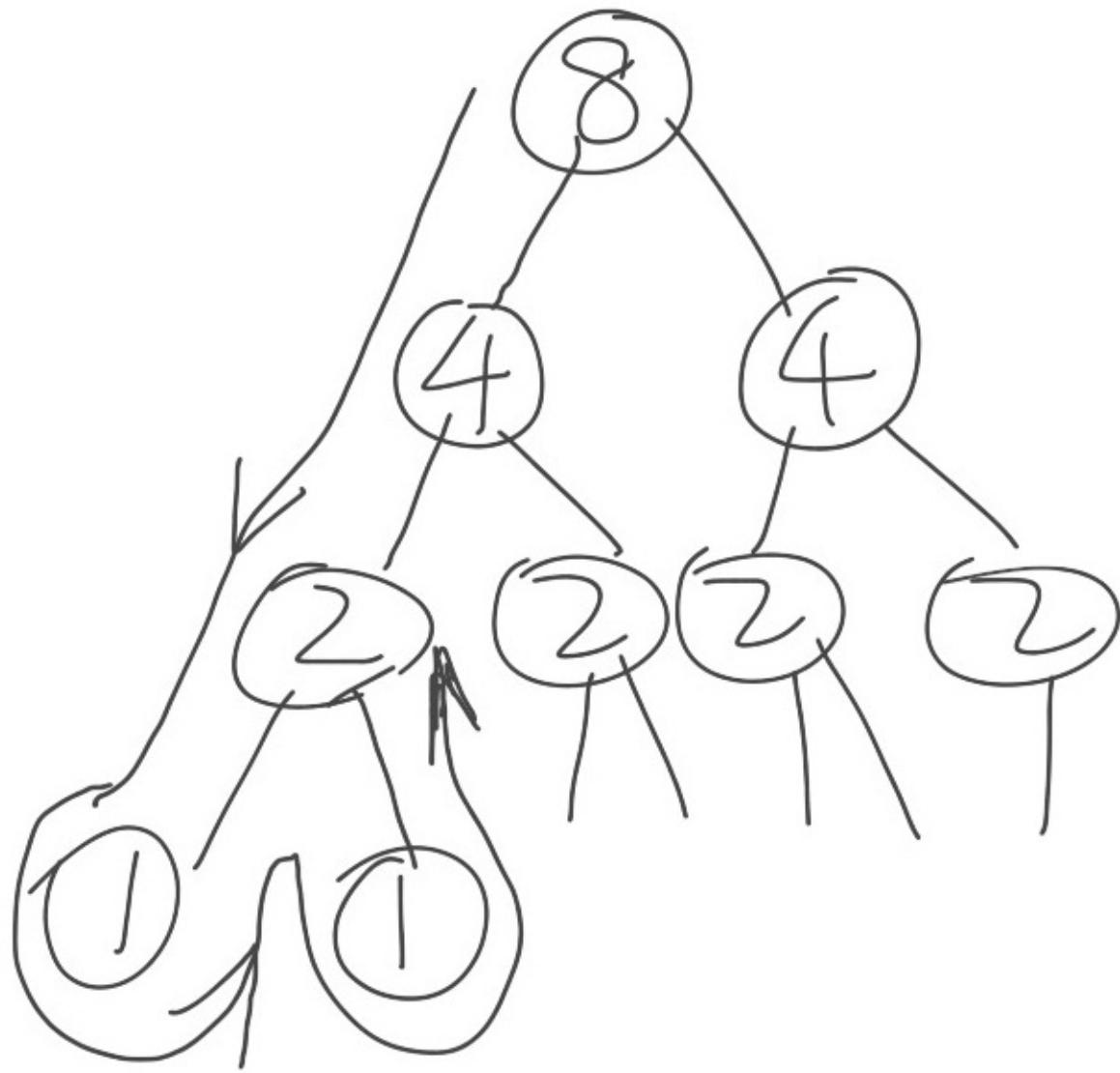
— out-in-place

$$\sum_{i=0}^n \frac{n}{2^i} = 2n$$

Python 84

out-in-place i $i - 1$

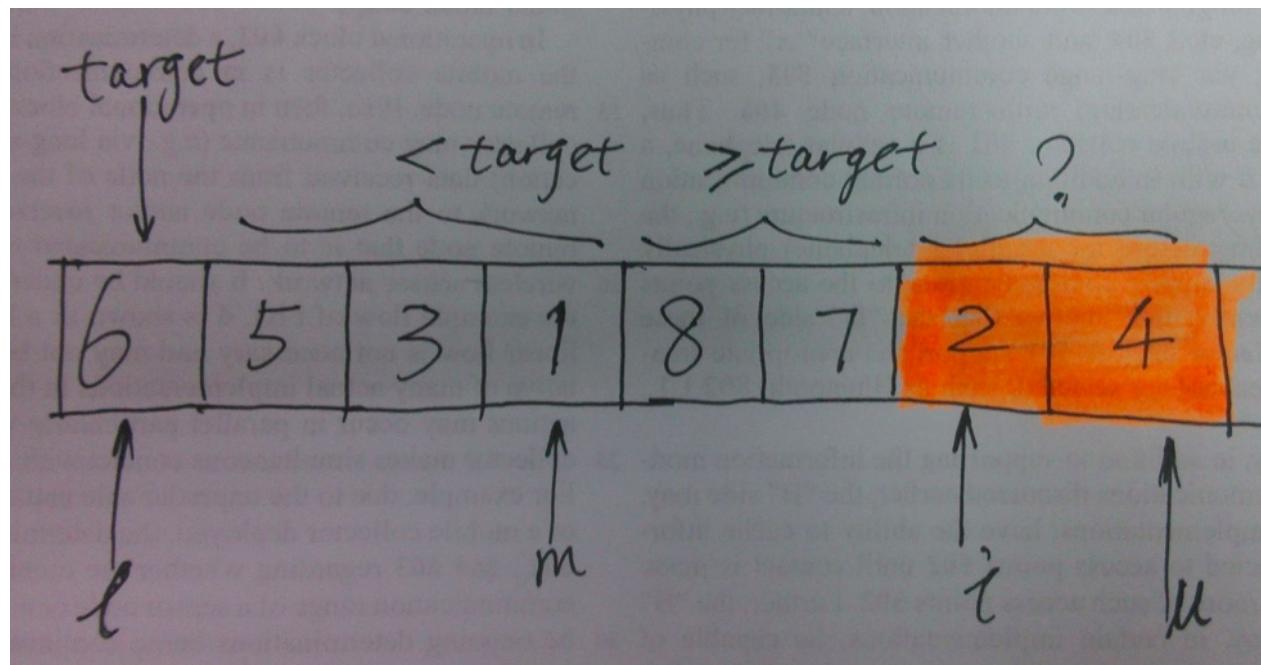
$$\sum_{i=0}^n (n - i + 1) = O(n^2)$$



in-place -

one index for partition

in-place	[6, 5, 3, 1, 8, 7, 2, 4]	l	u (lower bound)(upper bound)	m	i
partition		t , target.			



i	$x[i]$	$x[i] \geq t, i$	$x[i] < t, \quad x[i]$	<code>swap(x[++m], x[i]).</code>	$i == u$
					$l \geq u, \text{ Python}$

Python

```
#!/usr/bin/env python

def qsort2(alist, l, u):
    print(alist)
    if l >= u:
        return

    m = l
    for i in xrange(l + 1, u + 1):
        if alist[i] < alist[l]:
            m += 1
            alist[m], alist[i] = alist[i], alist[m]
    # swap between m and l after partition, important!
    alist[m], alist[l] = alist[l], alist[m]
    qsort2(alist, l, m - 1)
    qsort2(alist, m + 1, u)

unsortedArray = [6, 5, 3, 1, 8, 7, 2, 4]
print(qsort2(unsortedArray, 0, len(unsortedArray) - 1))
```

Java

```
public class Sort {
    public static void main(String[] args) {
        int unsortedArray[] = new int[]{6, 5, 3, 1, 8, 7, 2, 4};
        quickSort(unsortedArray);
        System.out.println("After sort: ");
```

```

        for (int item : unsortedArray) {
            System.out.print(item + " ");
        }
    }

    public static void quickSort1(int[] array, int l, int u) {
        for (int item : array) {
            System.out.print(item + " ");
        }
        System.out.println();

        if (l >= u) return;
        int m = l;
        for (int i = l + 1; i <= u; i++) {
            if (array[i] < array[l]) {
                m += 1;
                int temp = array[m];
                array[m] = array[i];
                array[i] = temp;
            }
        }
        // swap between array[m] and array[l]
        // put pivot in the mid
        int temp = array[m];
        array[m] = array[l];
        array[l] = temp;

        quickSort1(array, l, m - 1);
        quickSort1(array, m + 1, u);
    }

    public static void quickSort(int[] array) {
        quickSort1(array, 0, array.length - 1);
    }
}

```

partition i m $\text{alist}[i]$ $\text{alist}[l]$ $<$ \leq ! $<$ \leq $=$

```

[6, 5, 3, 1, 8, 7, 2, 4]
[4, 5, 3, 1, 2, 6, 8, 7]
[2, 3, 1, 4, 5, 6, 8, 7]
[1, 2, 3, 4, 5, 6, 8, 7]
[1, 2, 3, 4, 5, 6, 8, 7]
[1, 2, 3, 4, 5, 6, 8, 7]
[1, 2, 3, 4, 5, 6, 8, 7]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]

```

Two-way partitioning

partition $O(n^2)$. —— partition.

partition

6 5 3 1 8 7 2 4

1. 3
2. lo 6 , hi 4 , lo 3 , hi 3 lo hi —— 6 2
3. lo hi 2 lo 5 , hi 1 .
4. lo hi 3

1. $i \quad j$

2.

3. partition

- $i,$
- $j,$

4.

partition $\log n$ partition $n. \quad n \log n$ [programming_pearls](#)

Python

```
#!/usr/bin/env python

def qsort3(alist, l, u):
    print(alist)
    if l >= u:
        return

    t = alist[l]
    i = l + 1
    j = u
    while True:
        while i <= u and alist[i] < t:
            i += 1
        while alist[j] > t:
            j -= 1
        if i > j:
            break
        # swap after make sure i > j
        alist[i], alist[j] = alist[j], alist[i]
    # do not forget swap l and j
    alist[l], alist[j] = alist[j], alist[l]
```

```

qsort3(alist, l, j - 1)
qsort3(alist, j + 1, u)

unsortedArray = [6, 5, 3, 1, 8, 7, 2, 4]
print(qsort3(unsortedArray, 0, len(unsortedArray) - 1))

```

Java

```

public class Sort {
    public static void main(String[] args) {
        int unsortedArray[] = new int[]{6, 5, 3, 1, 8, 7, 2, 4};
        quickSort(unsortedArray);
        System.out.println("After sort: ");
        for (int item : unsortedArray) {
            System.out.print(item + " ");
        }
    }

    public static void quickSort2(int[] array, int l, int u) {
        for (int item : array) {
            System.out.print(item + " ");
        }
        System.out.println();

        if (l >= u) return;
        int pivot = array[l];
        int left = l + 1;
        int right = u;
        while (left <= right) {
            while (left <= right && array[left] < pivot) {
                left++;
            }
            while (array[right] > pivot) {
                right--;
            }
            if (left > right) break;
            // swap array[left] with array[right] while left <= right
            int temp = array[left];
            array[left] = array[right];
            array[right] = temp;
        }
        /* swap the smaller with pivot */
        int temp = array[right];
        array[right] = array[l];
        array[l] = temp;

        quickSort2(array, l, right - 1);
        quickSort2(array, right + 1, u);
    }

    public static void quickSort(int[] array) {
        quickSort2(array, 0, array.length - 1);
    }
}

```

```
[6, 5, 3, 1, 8, 7, 2, 4]
[2, 5, 3, 1, 4, 6, 7, 8]
[1, 2, 3, 5, 4, 6, 7, 8]
[1, 2, 3, 5, 4, 6, 7, 8]
[1, 2, 3, 5, 4, 6, 7, 8]
[1, 2, 3, 5, 4, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
```

3

- 1.
- 2.

Robert Sedgewick [Quicksort](#)

Reference

- -
- [Quicksort | Robert Sedgewick](#)
- Programming Pearls Column 11 Sorting -
 - [Quicksort Analysis](#)
 -  programming_pearls. Programming Pearls() 11 $n \log 2n$ ↵

Heap Sort -

0

0,

1.

2. — $2N \log N$

3. 0

- o `i (2*i+1)`
 - o `i (2*i+2)`
 - o `i floor((i-1)/2)`
-

1. Max_Heapify

2. Build_Max_Heap

3. HeapSort

123

6 5 3 1 8 7 2 4

C++

()TopM

C++

```

#include <iostream>
#include <vector>

using namespace std;

class HeapSort {
    // get the parent node index
    int parent(int i) {
        return (i - 1) / 2;
    }

    // get the left child node index
    int left(int i) {
        return 2 * i + 1;
    }

    // get the right child node index
    int right(int i) {
        return 2 * i + 2;
    }

    // build max heap
    void build_max_heapify(vector<int> &nums, int heap_size) {
        for (int i = heap_size / 2; i >= 0; --i) {
            max_heapify(nums, i, heap_size);
        }
        print_heap(nums, heap_size);
    }

    // build min heap
    void build_min_heapify(vector<int> &nums, int heap_size) {
        for (int i = heap_size / 2; i >= 0; --i) {
            min_heapify(nums, i, heap_size);
        }
        print_heap(nums, heap_size);
    }

    // adjust the heap to max-heap
    void max_heapify(vector<int> &nums, int k, int len) {
        // int len = nums.size();
        while (k < len) {
            int max_index = k;
            // left leaf node search
            int l = left(k);
            if (l < len && nums[l] > nums[max_index]) {
                max_index = l;
            }
            // right leaf node search
            int r = right(k);
            if (r < len && nums[r] > nums[max_index]) {
                max_index = r;
            }
            // node after k are max-heap already
            if (k == max_index) {
                break;
            }
            // keep the root node the largest
            int temp = nums[k];

```

```

        nums[k] = nums[max_index];
        nums[max_index] = temp;
        // adjust not only just current index
        k = max_index;
    }
}

// adjust the heap to min-heap
void min_heapify(vector<int> &nums, int k, int len) {
    // int len = nums.size();
    while (k < len) {
        int min_index = k;
        // left leaf node search
        int l = left(k);
        if (l < len && nums[l] < nums[min_index]) {
            min_index = l;
        }
        // right leaf node search
        int r = right(k);
        if (r < len && nums[r] < nums[min_index]) {
            min_index = r;
        }
        // node after k are min-heap already
        if (k == min_index) {
            break;
        }
        // keep the root node the largest
        int temp = nums[k];
        nums[k] = nums[min_index];
        nums[min_index] = temp;
        // adjust not only just current index
        k = min_index;
    }
}

public:
    // heap sort
    void heap_sort(vector<int> &nums) {
        int len = nums.size();
        // init heap structure
        build_max_heapify(nums, len);
        // heap sort
        for (int i = len - 1; i >= 0; --i) {
            // put the largest number int the last
            int temp = nums[0];
            nums[0] = nums[i];
            nums[i] = temp;
            // reconstruct heap
            build_max_heapify(nums, i);
        }
        print_heap(nums, len);
    }

    // print heap between [0, heap_size - 1]
    void print_heap(vector<int> &nums, int heap_size) {
        for (int i = 0; i < heap_size; ++i) {
            cout << nums[i] << ", ";
        }
        cout << endl;
    }
}

```

```

    }
};

int main(int argc, char *argv[])
{
    int A[] = {19, 1, 10, 14, 16, 4, 7, 9, 3, 2, 8, 5, 11};
    vector<int> nums;
    for (int i = 0; i < sizeof(A) / sizeof(A[0]); ++i) {
        nums.push_back(A[i]);
    }

    HeapSort sort;
    sort.print_heap(nums, nums.size());
    sort.heap_sort(nums);

    return 0;
}

```

$O(2N)$, $O(2N \log N)$. $O(N \log N)$.

(8)...

Reference

- -
- Priority Queues - Robert Sedgewick
- | Jark's Blog -
- Algorithm - Robert Sedgewick
- O(n)
- 9 9.7 - -
- 9 9.7 - -

Bucket Sort

- 1.
2. Divide -
- 3.
4. Conquer -

Reference

- [Bucket Sort Visualization](#) -
- -

Counting Sort

1. ——
2. —— iC_i
3. —— C
4. —— $iC(i)C(i)1$

Reference

- - -
- Counting Sort Visualization - ~

Radix Sort

Basics Miscellaneous

Bit Manipulation

nn

XOR -

01

```

x ^ 0 = x
x ^ 1s = ~x // 1s = ~0
x ^ (~x) = 1s
x ^ x = 0 // interesting and important!
a ^ b = c => a ^ c = b, b ^ c = a // swap
a ^ b ^ c = a ^ (b ^ c) = (a ^ b) ^ c // associative

```

/2 0b0010 * 0b0110 0b0110 << 2 .

1. x n - x & (~0 << n)
2. x n (01) - x & (1 << n)
3. x n - (x >> n) & 1
4. n 1 - x | (1 << n)
5. n 0 - x & (~(1 << n))
6. x n () - x & ((1 << n) - 1)
7. n 0() - x & (~((1 << (n + 1)) - 1))
8. n v ; v 110 - mask = ~(1 << n); x = (x & mask) | (v << i)

(Bitmap)

flag array

int

1/32.

100setbittestbit

```

#define N 1000000 // 1 million
#define WORD_LENGTH sizeof(int) * 8 //sizeof8int

//bitsii0~1000000
void setbit(unsigned int* bits, unsigned int i){
    bits[i / WORD_LENGTH] |= 1<<(i % WORD_LENGTH);
}

int testbit(unsigned int* bits, unsigned int i){

```

```
    return bits[i/WORD_LENGTH] & (1<<(i % WORD_LENGTH));
}

unsigned int bits[N/WORD_LENGTH + 1];
```

Reference

- 1 » NoAlgO
- 2 » NoAlgO
- | Matrix67: The Aha Moments
- cc150 chapter 8.5 and chapter 9.5
- 2
- Elementary Algorithms Larry LIU Xinyu

Knapsack -

Item(jewellery)	Weight	Value
1	6	23
2	3	13
3	4	11

17 kg...

$$W \text{ kg}, \quad n \quad \omega_1, \dots, \omega_n, \quad v_1, \dots, v_n.$$

Knapsack with repetition -

Knapsack with repetition, Unbounded knapsack problem().

$$\omega \leq W, (\quad 1, 2, \dots, j, \text{ for } j \leq n). \quad K(\omega) = \max_{\omega_i \in \Omega} K(\omega - \omega_i) + v_i$$

$$K(\omega) = \max_{\omega_i \in \Omega} K(\omega - \omega_i) + v_i \quad K(\omega) = F(\omega - \omega_i) + v_i \quad (\omega_i \in \Omega)$$

$$\begin{aligned} & F(\omega - \omega_i) \quad i \quad \omega_i < \omega(\dots), \quad \Omega \quad K(\omega) \quad \omega_i \quad K(\omega) \quad F(\omega - \omega_i), \\ & \dots \quad i \quad v_i \quad K(\omega) \quad F(\omega - \omega_i) \quad \omega - \omega_i \\ & F(\omega - \omega_i) < K(\omega - \omega_i), \quad K(\omega) = F(\omega - \omega_i) + v_i < K(\omega - \omega_i) + v_i = K'(\omega) \quad K(\omega) \\ & \omega \quad F(\omega - \omega_i) = K(\omega - \omega_i). \quad K(\omega) = K(\omega - \omega_i) + v_i \\ & \quad i \quad v_i \quad \omega \quad i, \quad \omega_i. \quad \omega \quad \omega \quad \omega_i \\ & K(\omega) \quad K(\omega), \quad K(\omega) \geq K(\omega - \omega_j) + v_j \quad (\omega_j \notin \Omega) \\ & K(\omega) = \max_{i: \omega_i \leq \omega} \{K(\omega - \omega_i) + v_i\} \end{aligned}$$

... ...

Knapsack without repetition - 01

$$\begin{aligned} & K(\omega) \quad K(i, \omega) \quad i \quad \omega \\ & K(i-1, \omega) \quad K(i, \omega) \quad i \quad i-1 \quad \omega \quad K(i-1, \omega) \quad i \quad i-1 \\ & \omega - \omega_i \quad K(i-1, \omega - \omega_i) + v_i. \\ & K(i, \omega) = \max\{K(i-1, \omega), K(i-1, \omega - \omega_i) + v_i\} \end{aligned}$$

Reference

- Chapter 6.4 Knapsack *Algorithm* - S. Dasgupta
- 0019——0-1 - liufeng_king
- § → 2.0 alpha1

Part II - Coding

leetcode

String -

|

strStr

Source

- leetcode: [Implement strStr\(\) | LeetCode OJ](#)
- lintcode: [lintcode - \(13\) strstr](#)

strstr (a.k.a find sub string), is a useful function in string operation.
Your task is to implement this function.

For a given source string and a target string,
you should output the "first" index(from 0) of target string in source string.

If target is not exist in source, just return -1.

Example

If source="source" and target="target", return -1.

If source="abcdabcde" and target="bcd", return 1.

Challenge

O(n) time.

Clarification

Do I need to implement KMP Algorithm in an interview?

- Not necessary. When this problem occurs in an interview,
the interviewer just want to test your basic implementation ability.

forKMP

Java

```
/**
 * http://www.jiuzhang.com//solutions/implement-strstr
 */
class Solution {
    /**
     * Returns a index to the first occurrence of target in source,
     * or -1 if target is not part of source.
     * @param source string to be scanned.
     * @param target string containing the sequence of characters to match.
     */
    public int strStr(String source, String target) {
        if (source == null || target == null) {
            return -1;
        }
    }
}
```

```

int i, j;
for (i = 0; i < source.length() - target.length() + 1; i++) {
    for (j = 0; j < target.length(); j++) {
        if (source.charAt(i + j) != target.charAt(j)) {
            break;
        } //if
    } //for j
    if (j == target.length()) {
        return i;
    }
} //for i

// did not find the target
return -1;
}
}

```

1. source target
2. i i < source.length() source.charAt(i + j)
3. 1 == 2 s1``s2 target``source 3if {return -1;} 4Java C++
5 int i, j;
4. for i , j

Another Similar Question

```

/**
 * http://www.jiuzhang.com//solutions/implement-strstr
 */
public class Solution {
    public String strStr(String haystack, String needle) {
        if(haystack == null || needle == null) {
            return null;
        }
        int i, j;
        for(i = 0; i < haystack.length() - needle.length() + 1; i++) {
            for(j = 0; j < needle.length(); j++) {
                if(haystack.charAt(i + j) != needle.charAt(j)) {
                    break;
                }
            }
            if(j == needle.length()) {
                return haystack.substring(i);
            }
        }
        return null;
    }
}

```

Two Strings Are Anagrams

Source

- CC150: [\(158\) Two Strings Are Anagrams](#)

Write a method `anagram(s,t)` to decide if two strings are anagrams or not.

Example

Given `s="abcd"`, `t="dcab"`, return true.

Challenge

$O(n)$ time, $O(1)$ extra space

1 - hashmap

`false`.

C++

```
class Solution {
public:
    /**
     * @param s: The first string
     * @param t: The second string
     * @return true or false
     */
    bool anagram(string s, string t) {
        if (s.empty() || t.empty()) {
            return false;
        }
        if (s.size() != t.size()) {
            return false;
        }

        int letterCount[256] = {0};

        for (int i = 0; i != s.size(); ++i) {
            ++letterCount[s[i]];
            --letterCount[t[i]];
        }
        for (int i = 0; i != t.size(); ++i) {
            if (letterCount[t[i]] != 0) {
                return false;
            }
        }

        return true;
    }
};
```

```

1. ()  

2. 256  

3. s t      letterCount 0    false .  
  

()      for      t.size() > 256 256    t.size(), i t[i] .

```

$O(2n)$, $O(256)$.

2 -

1 hashmap hashmap

C++

```

class Solution {
public:
    /**
     * @param s: The first string
     * @param b: The second string
     * @return true or false
     */
    bool anagram(string s, string t) {
        if (s.empty() || t.empty()) {
            return false;
        }
        if (s.size() != t.size()) {
            return false;
        }

        sort(s.begin(), s.end());
        sort(t.begin(), t.end());

        if (s == t) {
            return true;
        } else {
            return false;
        }
    }
};

```

s t

C++ STL sort $O(n)$ $O(n^2)$ $s == t$ $O(n)$.

Reference

- CC150 Chapter 9.1 p109

Compare Strings

Source

- lintcode: [\(55\) Compare Strings](#)

Compare two strings A and B, determine whether A contains all of the characters in B.

The characters in string A and B are all Upper Case letters.

Example

For A = "ABCD", B = "ABC", return true.

For A = "ABCD" B = "AABC", return false.

[Two Strings Are Anagrams | Data Structure and Algorithm](#) BAB="AABC" A

A="ABCD" A false.

strstr AB A B

C++

```
class Solution {
public:
    /**
     * @param A: A string includes Upper Case letters
     * @param B: A string includes Upper Case letter
     * @return: if string A contains all of the characters in B return true
     *          else return false
     */
    bool compareStrings(string A, string B) {
        if (A.size() < B.size()) {
            return false;
        }

        const int AlphabetNum = 26;
        int letterCount[AlphabetNum] = {0};
        for (int i = 0; i != A.size(); ++i) {
            ++letterCount[A[i] - 'A'];
        }
        for (int i = 0; i != B.size(); ++i) {
            --letterCount[B[i] - 'A'];
            if (letterCount[B[i] - 'A'] < 0) {
                return false;
            }
        }
    }

    return true;
}
```

```
    }  
};
```

1. B A `false`,

2.

A B $O(2n)$, $O(26)$.

Anagrams

Source

- leetcode: [Anagrams | LeetCode OJ](#)
- lintcode: [\(171\) Anagrams](#)

Given an array of strings, return all groups of strings that are anagrams.

Example

Given ["lint", "intl", "inlt", "code"], return ["lint", "inlt", "intl"].

Given ["ab", "ba", "cd", "dc", "e"], return ["ab", "ba", "cd", "dc"].

Note

All inputs will be in lower-case

1 - for (TLE)

Two Strings Are Anagrams for $O(n)$

C++

```
class Solution {
public:
    /**
     * @param strs: A list of strings
     * @return: A list of strings
     */
    vector<string> anagrams(vector<string> &strs) {
        if (strs.size() < 2) {
            return strs;
        }

        vector<string> result;
        vector<bool> visited(strs.size(), false);
        for (int s1 = 0; s1 != strs.size(); ++s1) {
            bool has_anagrams = false;
            for (int s2 = s1 + 1; s2 < strs.size(); ++s2) {
                if ((!visited[s2]) && isAnagrams(strs[s1], strs[s2])) {
                    result.push_back(strs[s2]);
                    visited[s2] = true;
                    has_anagrams = true;
                }
            }
            if ((!visited[s1]) && has_anagrams) result.push_back(strs[s1]);
        }

        return result;
    }
}
```

```

private:
    bool isAnagrams(string &s, string &t) {
        if (s.size() != t.size()) {
            return false;
        }

        const int AlphabetNum = 26;
        int letterCount[AlphabetNum] = {0};
        for (int i = 0; i != s.size(); ++i) {
            ++letterCount[s[i] - 'a'];
            --letterCount[t[i] - 'a'];
        }
        for (int i = 0; i != t.size(); ++i) {
            if (letterCount[t[i] - 'a'] < 0) {
                return false;
            }
        }
        return true;
    }
};

```

1. strs 1
2. strs
- 3.
4. isAnagrams

isAnagrams $O(2L)$, L for $\frac{1}{2}O(n^2)$, n $O(n^2L)$. 26 int $O(1)$.

2 - + hashmap

[Two Strings Are Anagrams](#) hashmap hashmap key value vector

leetcode signature anagrams

C++ - lintcode

```

class Solution {
public:
    /**
     * @param strs: A list of strings
     * @return: A list of strings
     */
    vector<string> anagrams(vector<string> &strs) {
        unordered_map<string, int> hash;

```

```

        for (int i = 0; i < strs.size(); i++) {
            string str = strs[i];
            sort(str.begin(), str.end());
            ++hash[str];
        }

        vector<string> result;
        for (int i = 0; i < strs.size(); i++) {
            string str = strs[i];
            sort(str.begin(), str.end());
            if (hash[str] > 1) {
                result.push_back(strs[i]);
            }
        }

        return result;
    }
};

```

Java - leetcode

```

public class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {
        List<List<String>> result = new ArrayList<List<String>>();
        if (strs == null) return result;

        // one key to multiple value multiMap
        Map<String, ArrayList<String>> multiMap = new HashMap<String, ArrayList<String>>();
        for (String str : strs) {
            char[] strChar = str.toCharArray();
            Arrays.sort(strChar);
            String strSorted = String.valueOf(strChar);
            if (multiMap.containsKey(strSorted)) {
                ArrayList<String> aList = multiMap.get(strSorted);
                aList.add(str);
                multiMap.put(strSorted, aList);
            } else {
                ArrayList<String> aList = new ArrayList<String>();
                aList.add(str);
                multiMap.put(strSorted, aList);
            }
        }

        // add List group to result
        Set<String> keySet = multiMap.keySet();
        for (String key : keySet) {
            ArrayList<String> aList = multiMap.get(key);
            Collections.sort(aList);
            result.add(aList);
        }

        return result;
    }
}

```

key value hashmap, `unordered_map` C++ 11 `unordered_map`, `map` `map-unordered_map`

1

leetcode signature `HashMap` `ArrayList` `value`. `Java` `String` `char[]`,
`String`.

$O(n)$, $O(L \log L)$. $O(nL \log L)$. $O(K)$, K

Reference

- `unordered_map`. [unordered_map - C++ Reference](#) ↵
- `map-unordered_map`. [c++ - Choosing between std::map and std::unordered_map - Stack Overflow](#) ↵
- [Anagrams](#) |

Longest Common Substring

Source

- lintcode: [\(79\) Longest Common Substring](#)

Given two strings, find the longest common substring.
Return the length of it.

Example

Given A="ABCD", B="CBCE", return 2.

Note

The characters in substring should occur continuously in original string.
This is different with subsequence.

C++

```
class Solution {
public:
    /**
     * @param A, B: Two string.
     * @return: the length of the longest common substring.
     */
    int longestCommonSubstring(string &A, string &B) {
        if (A.empty() || B.empty()) {
            return 0;
        }

        int lcs = 0, lcs_temp = 0;
        for (int i = 0; i < A.size(); ++i) {
            for (int j = 0; j < B.size(); ++j) {
                lcs_temp = 0;
                while ((i + lcs_temp < A.size()) &&\n                    (j + lcs_temp < B.size()) &&\n                    (A[i + lcs_temp] == B[j + lcs_temp]))
                {
                    ++lcs_temp;
                }

                // update lcs
                if (lcs_temp > lcs) {
                    lcs = lcs_temp;
                }
            }
        }
    }
}
```

```
        return lcs;
    }
};
```

1. 0.
2. i j
3. lcs_temp lcs lcs .
4. lcs .

```
while ++i ++j
```

for $O(mn \cdot lcs)$.

Reference

- [Longest Common Substring |](#)

Rotate String

Source

- lintcode: (8) Rotate String

```
Given a string and an offset, rotate string by offset. (rotate from left to right)
```

Example

Given "abcdefg"

```
for offset=0, return "abcdefg"  
for offset=1, return "gabcdef"  
for offset=2, return "fgabcde"  
for offset=3, return "efgabcd"  
...
```

offset

Python

```
class Solution:  
    """  
    param A: A string  
    param offset: Rotate string with offset.  
    return: Rotated string.  
    """  
    def rotateString(self, A, offset):  
        if A is None or len(A) == 0:  
            return A  
  
        offset %= len(A)  
        before = A[:len(A) - offset]  
        after = A[len(A) - offset:]  
        # [::-1] means reverse in Python  
        A = before[::-1] + after[::-1]  
        A = A[::-1]  
  
        return A
```

C++

```

class Solution {
public:
    /**
     * param A: A string
     * param offset: Rotate string with offset.
     * return: Rotated string.
     */
    string rotateString(string A, int offset) {
        if (A.empty() || A.size() == 0) {
            return A;
        }

        int len = A.size();
        offset %= len;
        reverse(A, 0, len - offset - 1);
        reverse(A, len - offset, len - 1);
        reverse(A, 0, len - 1);
        return A;
    }

private:
    void reverse(string &str, int start, int end) {
        while (start < end) {
            char temp = str[start];
            str[start] = str[end];
            str[end] = temp;
            start++;
            end--;
        }
    }
};

```

Java

```

public class Solution {
    /*
     * param A: A string
     * param offset: Rotate string with offset.
     * return: Rotated string.
     */
    public char[] rotateString(char[] A, int offset) {
        if (A == null || A.length == 0) {
            return A;
        }

        int len = A.length;
        offset %= len;
        reverse(A, 0, len - offset - 1);
        reverse(A, len - offset, len - 1);
        reverse(A, 0, len - 1);

        return A;
    }

    private void reverse(char[] str, int start, int end) {
        while (start < end) {
            char temp = str[start];

```

```
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
}
};
```

1. A0
2. offset A len
- 3.

Python slice Pythonic!

$O(n)$, $O(1)$. 3 $O(n)$, $O(1)$.

Reference

- [Reverse a string in Python - Stack Overflow](#)

Reverse Words in a String

Source

- lintcode: [\(53\) Reverse Words in a String](#)

Given an input string, reverse the string word by word.

For example,
Given s = "the sky is blue",
return "blue is sky the".

Example
Clarification

- What constitutes a word?

A sequence of non-space characters constitutes a word.

- Could the input string contain leading or trailing spaces?

Yes. However, your reversed string should not contain leading or trailing spaces.

- How about multiple spaces between two words?

Reduce them to a single space in the reversed string.

1.

2.

3.

0——appendappend

C++

```
class Solution {
public:
    /**
     * @param s : A string
     * @return : A string
     */
    string reverseWords(string s) {
        if (s.empty()) {
            return s;
        }

        string s_ret, s_temp;
        string::size_type ix = s.size();
        while (ix != 0) {
            s_temp.clear();
```

```

        while (!isspace(s[--ix])) {
            s_temp.push_back(s[ix]);
            if (ix == 0) {
                break;
            }
        }
        if (!s_temp.empty()) {
            if (!s_ret.empty()) {
                s_ret.push_back(' ');
            }
            std::reverse(s_temp.begin(), s_temp.end());
            s_ret.append(s_temp);
        }
    }

    return s_ret;
}
};

```

1. s
2. ix = s.size() ix = s.size() - 1 ix == 0
3. s_ret, s_temp O(n) s_temp append s_ret
4. s_ret

O(1)

- 1.
- 2.
- 3.
- 4.

Valid Palindrome

- tags: [palindrome]

Source

- leetcode: [Valid Palindrome | LeetCode OJ](#)
- lintcode: [\(415\) Valid Palindrome](#)

Given a string, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Example

"A man, a plan, a canal: Panama" is a palindrome.

"race a car" is not a palindrome.

Note

Have you consider that the string might be empty?
 This is a good question to ask during an interview.
 For the purpose of this problem,
 we define empty string as valid palindrome.

Challenge

$O(n)$ time without extra memory.

[Check if a singly linked list is palindrome.](#)

Python

```
class Solution:
    # @param {string} s A string
    # @return {boolean} Whether the string is a valid palindrome
    def isPalindrome(self, s):
        if not s:
            return True

        l, r = 0, len(s) - 1

        while l < r:
            # find left alphanumeric character
            if not s[l].isalnum():
                l += 1
                continue
            # find right alphanumeric character
            if not s[r].isalnum():
                r -= 1
                continue
            # case insensitive compare
```

```

    if s[l].lower() == s[r].lower():
        l += 1
        r -= 1
    else:
        return False
    #
return True

```

C++

```

class Solution {
public:
    /**
     * @param s A string
     * @return Whether the string is a valid palindrome
     */
    bool isPalindrome(string& s) {
        if (s.empty()) return true;

        int l = 0, r = s.size() - 1;
        while (l < r) {
            // find left alphanumeric character
            if (!isalnum(s[l])) {
                ++l;
                continue;
            }
            // find right alphanumeric character
            if (!isalnum(s[r])) {
                --r;
                continue;
            }
            // case insensitive compare
            if (tolower(s[l]) == tolower(s[r])) {
                ++l;
                --r;
            } else {
                return false;
            }
        }

        return true;
    }
};

```

Java

```

public class Solution {
    /**
     * @param s A string
     * @return Whether the string is a valid palindrome
     */
    public boolean isPalindrome(String s) {
        if (s == null || s.isEmpty()) return true;

```

```

int l = 0, r = s.length() - 1;
while (l < r) {
    // find left alphanumeric character
    if (!Character.isLetterOrDigit(s.charAt(l))) {
        l++;
        continue;
    }
    // find right alphanumeric character
    if (!Character.isLetterOrDigit(s.charAt(r))) {
        r--;
        continue;
    }
    // case insensitive compare
    if (Character.toLowerCase(s.charAt(l)) == Character.toLowerCase(s.charAt(r)))
        l++;
    r--;
} else {
    return false;
}
}

return true;
}
}

```

1. ()
- 2.

API

$O(n)$, $O(1)$.

Longest Palindromic Substring

- tags: [palindrome]

Source

- leetcode: [Longest Palindromic Substring | LeetCode OJ](#)
- lintcode: [\(200\) Longest Palindromic Substring](#)

Given a string S, find the longest palindromic substring in S.
 You may assume that the maximum length of S is 1000,
 and there exists one unique longest palindromic substring.

Example

Given the string = "abcdzdcab", return "cdzdc".

Challenge

$O(n^2)$ time is acceptable. Can you do it in $O(n)$ time.

1 -

Python

```
class Solution:
    # @param {string} s input string
    # @return {string} the longest palindromic substring
    def longestPalindrome(self, s):
        if not s:
            return ""

        n = len(s)
        longest, left, right = 0, 0, 0
        for i in xrange(0, n):
            for j in xrange(i + 1, n + 1):
                substr = s[i:j]
                if self.isPalindrome(substr) and len(substr) > longest:
                    longest = len(substr)
                    left, right = i, j
        # construct longest substr
        result = s[left:right]
        return result

    def isPalindrome(self, s):
        if not s:
            return False
        # reverse compare
        return s == s[::-1]
```

C++

```

class Solution {
public:
    /**
     * @param s input string
     * @return the longest palindromic substring
     */
    string longestPalindrome(string& s) {
        string result;
        if (s.empty()) return s;

        int n = s.size();
        int longest = 0, left = 0, right = 0;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j <= n; ++j) {
                string substr = s.substr(i, j - i);
                if (isPalindrome(substr) && substr.size() > longest) {
                    longest = j - i;
                    left = i;
                    right = j;
                }
            }
        }

        result = s.substr(left, right - left);
        return result;
    }

private:
    bool isPalindrome(string &s) {
        int n = s.size();
        for (int i = 0; i < n; ++i) {
            if (s[i] != s[n - i - 1]) return false;
        }
        return true;
    }
};

```

Java

```

public class Solution {
    /**
     * @param s input string
     * @return the longest palindromic substring
     */
    public String longestPalindrome(String s) {
        String result = new String();
        if (s == null || s.isEmpty()) return result;

        int n = s.length();
        int longest = 0, left = 0, right = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j <= n; j++) {
                String substr = s.substring(i, j);

```

```

        if (isPalindrome(substr) && substr.length() > longest) {
            longest = substr.length();
            left = i;
            right = j;
        }
    }

    result = s.substring(left, right);
    return result;
}

private boolean isPalindrome(String s) {
    if (s == null || s.isEmpty()) return false;

    int n = s.length();
    for (int i = 0; i < n; i++) {
        if (s.charAt(i) != s.charAt(n - i - 1)) return false;
    }

    return true;
}
}

```

`left` , `right`

$$O(C_n^2) = O(n^2), \quad O(n), \quad O(n^3). \quad \text{TLE. } \text{substr} \quad O(n).$$

Reference

- [Longest Palindromic Substring Part I | LeetCode](#)
- [Longest Palindromic Substring Part II | LeetCode](#)

Space Replacement

Source

- lintcode: [\(212\) Space Replacement](#)

Write a method to replace all spaces in a string with %20.
The string is given in a characters array, you can assume it has enough space for replacement and you are given the true length of the string.

Example

Given "Mr John Smith", length = 13.

The string after replacement should be "Mr%20John%20Smith".

Note

If you are using Java or Python please use characters array instead of string.

Challenge

Do it in-place.

%20 %20 ——

Java

```
public class Solution {
    /**
     * @param string: An array of Char
     * @param length: The true length of the string
     * @return: The true length of new string
     */
    public int replaceBlank(char[] string, int length) {
        if (string == null) return 0;

        int space = 0;
        for (char c : string) {
            if (c == ' ') space++;
        }

        int r = length + 2 * space - 1;
        for (int i = length - 1; i >= 0; i--) {
            if (string[i] != ' ') {
                string[r] = string[i];
                r--;
            } else {
                string[r--] = '0';
            }
        }
    }
}
```

```
        string[r--] = '2';
        string[r--] = '%';
    }

    return length + 2 * space;
}
}
```

$O(n)$, $r = O(1)$.

Wildcard Matching

Source

- leetcode: [Wildcard Matching | LeetCode OJ](#)
- lintcode: [\(192\) Wildcard Matching](#)

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

'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).
The matching should cover the entire input string (not partial).

Example

```
isMatch("aa", "a") → false
isMatch("aa", "aa") → true
isMatch("aaa", "aa") → false
isMatch("aa", "*") → true
isMatch("aa", "a*") → true
isMatch("ab", "?*") → true
isMatch("aab", "c*a*b") → false
```

1 - DFS

'	?	*	'()	s , p	p ? '	*	', *	0, 1, 2...	*	s	p *	s
p	*		s	p	*							

Java

```
public class Solution {
    /**
     * @param s: A string
     * @param p: A string includes "?" and "*"
     * @return: A boolean
     */
    public boolean isMatch(String s, String p) {
        if (s == null || p == null) return false;
        if (s.length() == 0 || p.length() == 0) return false;

        return helper(s, 0, p, 0);
    }

    private boolean helper(String s, int si, String p, int pj) {
        // index out of range check
        if (si == s.length() || pj == p.length()) {
            if (si == s.length() && pj == p.length()) {
                return true;
            } else {
                return false;
            }
        }

        if (s.charAt(si) == p.charAt(pj) || p.charAt(pj) == '?') {
            return helper(s, si + 1, p, pj + 1);
        }

        if (p.charAt(pj) == '*') {
            if (helper(s, si, p, pj + 1)) return true;
            if (helper(s, si + 1, p, pj)) return true;
            return false;
        }

        return false;
    }
}
```

```

        }
    }

    if (p.charAt(pj) == '*') {
        // remove continuous *
        while (p.charAt(pj) == '*') {
            pj++;
            // index out of range check
            if (pj == p.length()) return true;
        }

        // compare remaining part of p after * with s
        while (si < s.length() && !helper(s, si, p, pj)) {
            si++;
        }
        // substring of p equals to s
        return si != s.length();
    } else if (s.charAt(si) == p.charAt(pj) || p.charAt(pj) == '?') {
        return helper(s, si + 1, p, pj + 1);
    } else {
        return false;
    }
}
}

```

*

$$O(n!) \times O(m!), \quad O(1)0$$

Reference

- Soulmachine leetcode
-

Length of Last Word

Source

- leetcode: Length of Last Word | LeetCode OJ
- lintcode: (422) Length of Last Word

Given a string s consists of upper/lower-case alphabets and empty space characters ' ', return the length of last word in the string.

If the last word does not exist, return 0.

Have you met this question in a real interview? Yes

Example

Given s = "Hello World", return 5.

Note

A word is defined as a character sequence consists of non-space characters only.



Java

```
public class Solution {
    /**
     * @param s A string
     * @return the length of last word
     */
    public int lengthOfLastWord(String s) {
        if (s == null || s.isEmpty()) return 0;

        // trim right space
        int begin = 0, end = s.length();
        while (end > 0 && s.charAt(end - 1) == ' ') {
            end--;
        }
        // find the last space
        for (int i = 0; i < end; i++) {
            if (s.charAt(i) == ' ') {
                begin = i + 1;
            }
        }

        return end - begin;
    }
}
```

$O(n)$.

Count and Say

Source

- leetcode: [Count and Say | LeetCode OJ](#)
- lintcode: [\(420\) Count and Say](#)

The count-and-say sequence is the sequence of integers beginning as follows:

1, 11, 21, 1211, 111221, ...

1 is read off as "one 1" or 11.

11 is read off as "two 1s" or 21.

21 is read off as "one 2, then one 1" or 1211.

Given an integer n, generate the nth sequence.

Example

Given n = 5, return "111221".

Note

The sequence of integers will be represented as a string.

n ()+

Java

```
public class Solution {
    /**
     * @param n the nth
     * @return the nth sequence
     */
    public String countAndSay(int n) {
        if (n <= 0) return null;

        String s = "1";
        for (int i = 1; i < n; i++) {
            int count = 1;
            StringBuilder sb = new StringBuilder();
            int sLen = s.length();
            for (int j = 0; j < sLen; j++) {
                if (j < sLen - 1 && s.charAt(j) == s.charAt(j + 1)) {
                    count++;
                } else {
                    sb.append(count + "" + s.charAt(j));
                    // reset
                    count = 1;
                }
            }
            s = sb.toString();
        }
        return s;
    }
}
```

```
        count = 1;
    }
}
s = sb.toString();
}

return s;
}
}
```

StringBuilder s "1", for n-1.

Reference

- [\[leetcode\]Count and Say -](#)

Integer Array -

Remove Element

Source

- leetcode: Remove Element | LeetCode OJ
- lintcode: (172) Remove Element

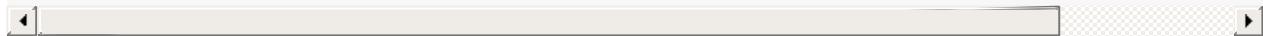
Given an array and a value, remove all occurrences of that value in place and return the length of the array.

The order of elements can be changed, and the elements after the new length don't matter.

Example

Given an array [0,4,4,0,0,2,4,4], value=4

return 4 and front four elements of the array is [0,0,0,2]



1 -

lintcode

C++

```
class Solution {
public:
    /**
     *@param A: A list of integers
     *@param elem: An integer
     *@return: The new length after remove
     */
    int removeElement(vector<int> &A, int elem) {
        for (vector<int>::iterator iter = A.begin(); iter < A.end(); ++iter) {
            if (*iter == elem) {
                iter = A.erase(iter);
                --iter;
            }
        }
        return A.size();
    }
};
```

--iter , for iter A.erase() while

vector~~erase~~ $O(n)$ $O(n^2)$ 2

2 -

C++

```
class Solution {
public:
    int removeElement(int A[], int n, int elem) {
        for (int i = 0; i < n; ++i) {
            if (A[i] == elem) {
                A[i] = A[n - 1];
                --i;
                --n;
            }
        }
        return n;
    }
};
```

A[i] == elem (n) i n 1 n

$O(n)$

Reference

- [Remove Element](#) |

Zero Sum Subarray

Source

- lintcode: [\(138\) Subarray Sum](#)
- GeeksforGeeks: [Find if there is a subarray with 0 sum - GeeksforGeeks](#)

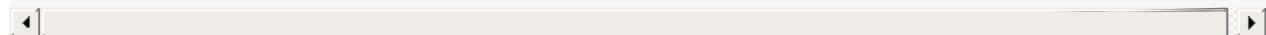
Given an integer array, find a subarray where the sum of numbers is zero.
Your code should return the index of the first number and the index of the last number.

Example

Given $[-3, 1, 2, -3, 4]$, return $[0, 2]$ or $[1, 3]$.

Note

There is at least one subarray that it's sum equals to zero.



1 - for

(00) $O(n^2)$, TLE.

2 - (TLE)

for $f(i) = \sum_0^i \text{nums}[i]$ 0 $i < 0$ $i_1 \quad i_2 \quad f(i_1) - f(i_2) = 0$,
 $f(i_1) = f(i_2)$. vector 0 i push vector vector

C++

```
class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    vector<int> subarraySum(vector<int> nums){
        vector<int> result;

        int curr_sum = 0;
        vector<int> sum_i;
        for (int i = 0; i != nums.size(); ++i) {
            curr_sum += nums[i];

            if (0 == curr_sum) {
                result.push_back(0);
                result.push_back(i);
            }
        }
        return result;
    }
}
```

```

    }

    vector<int>::iterator iter = find(sum_i.begin(), sum_i.end(), curr_sum);
    if (iter != sum_i.end()) {
        result.push_back(iter - sum_i.begin() + 1);
        result.push_back(i);
        return result;
    }

    sum_i.push_back(curr_sum);
}

return result;
};

}

```

`curr_sum i sum_i sum_i.push_back curr_sum 0 curr_sum sum_i 1` `find`
 $O(1)$.

$O(n^2), 1$

3 -

2 :)

C++

```

class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    vector<int> subarraySum(vector<int> nums){
        vector<int> result;
        // curr_sum for the first item, index for the second item
        map<int, int> hash;
        hash[0] = 0;

        int curr_sum = 0;
        for (int i = 0; i != nums.size(); ++i) {
            curr_sum += nums[i];
            if (hash.find(curr_sum) != hash.end()) {
                result.push_back(hash[curr_sum]);
                result.push_back(i);
                return result;
            } else {

```

```

        hash[curr_sum] = i + 1;
    }
}

return result;
};

curr_sum == 0 <0, 0>. hash i + 1, push_back 1.

```

C++ map C++ 11 unordered_map 1300ms 1000ms.

$O(n)$, $O(\log L)$, L unordered_map

4 -

pair sum sort_pair_second.

C++

```

class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    vector<int> subarraySum(vector<int> nums){
        vector<int> result;
        if (nums.empty()) {
            return result;
        }

        const int num_size = nums.size();
        vector<pair<int, int>> sum_index(num_size + 1);
        for (int i = 0; i != num_size; ++i) {
            sum_index[i + 1].first = sum_index[i].first + nums[i];
            sum_index[i + 1].second = i + 1;
        }

        sort(sum_index.begin(), sum_index.end());
        for (int i = 1; i < num_size + 1; ++i) {
            if (sum_index[i].first == sum_index[i - 1].first) {
                result.push_back(sum_index[i - 1].second);
                result.push_back(sum_index[i].second - 1);
                return result;
            }
        }
    }
}

```

```
        return result;
    }
};
```

dummy pair

$O(n)$, $O(n)$. $O(n \log n)$, $O(n)$. $O(n \log n)$. $O(n)$.

Stackoverflow stackoverflow

Google - Find subarray with given sum - GeeksforGeeks.

Reference

- stackoverflow. [algorithm - Zero sum SubArray - Stack Overflow](#) ↵
- sort_pair_second. [C++ - How do I sort a vector of pairs based on the second element of the pair? - Stack Overflow](#) ↵

Subarray Sum K

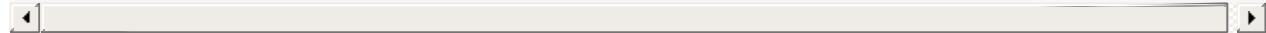
Source

- GeeksforGeeks: [Find subarray with given sum - GeeksforGeeks](#)

Given an nonnegative integer array, find a subarray where the sum of numbers is k. Your code should return the index of the first number and the index of the last number.

Example

Given [1, 4, 20, 3, 10, 5], sum k = 33, return [2, 4].



1 -

[Zero Sum Subarray | Data Structure and Algorithm](#)

$$f(i_1) - f(i_2) = 0$$

$$f(i_1) - f(i_2) = k$$

C++

```
#include <iostream>
#include <vector>
#include <map>

using namespace std;

class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    vector<int> subarraySum(vector<int> nums, int k){
        vector<int> result;
        // curr_sum for the first item, index for the second item
        // unordered_map<int, int> hash;
        map<int, int> hash;
        hash[0] = 0;

        int curr_sum = 0;
        for (int i = 0; i != nums.size(); ++i) {
            curr_sum += nums[i];
            if (hash.find(curr_sum - k) != hash.end()) {
                result.push_back(hash[curr_sum - k]);
                result.push_back(i);
                return result;
            } else {
                hash[curr_sum] = i + 1;
            }
        }
    }
}
```

```

        }
    }

    return result;
}
};

int main(int argc, char *argv[])
{
    int int_array1[] = {1, 4, 20, 3, 10, 5};
    int int_array2[] = {1, 4, 0, 0, 3, 10, 5};
    vector<int> vec_array1;
    vector<int> vec_array2;
    for (int i = 0; i != sizeof(int_array1) / sizeof(int); ++i) {
        vec_array1.push_back(int_array1[i]);
    }
    for (int i = 0; i != sizeof(int_array2) / sizeof(int); ++i) {
        vec_array2.push_back(int_array2[i]);
    }

    Solution solution;
    vector<int> result1 = solution.subarraySum(vec_array1, 33);
    vector<int> result2 = solution.subarraySum(vec_array2, 7);

    cout << "result1 = [" << result1[0] << " , " << result1[1] << " ]" << endl;
    cout << "result2 = [" << result2[0] << " , " << result2[1] << " ]" << endl;

    return 0;
}

```

Zero Sum Subarray `hash.find(curr_sum - k)`, `result.push_back(hash[curr_sum - k]);`
`result.push_back(hash[curr_sum]);`

Zero Sum Subarray | Data Structure and Algorithm

2 -

—**nonnegative integer array**, $f(i)$ $i_2 > i_1$, $f(i_2) - f(i_1) = k$,
 $f(i_2) \geq k$.

$\{1, 4, 20, 3, 10, 5\}$ 33 $f(i)$.

$f(i)$	1	5	25	28	38
i	0	1	2	3	4

33438033

C++

```

#include <iostream>
#include <vector>
#include <map>

using namespace std;

class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    vector<int> subarraySum2(vector<int> &nums, int k){
        vector<int> result;

        int left_index = 0, curr_sum = 0;
        for (int i = 0; i != nums.size(); ++i) {
            while (curr_sum > k) {
                curr_sum -= nums[left_index];
                ++left_index;
            }

            if (curr_sum == k) {
                result.push_back(left_index);
                result.push_back(i - 1);
                return result;
            }
            curr_sum += nums[i];
        }
        return result;
    }
};

int main(int argc, char *argv[])
{
    int int_array1[] = {1, 4, 20, 3, 10, 5};
    int int_array2[] = {1, 4, 0, 0, 3, 10, 5};
    vector<int> vec_array1;
    vector<int> vec_array2;
    for (int i = 0; i != sizeof(int_array1) / sizeof(int); ++i) {
        vec_array1.push_back(int_array1[i]);
    }
    for (int i = 0; i != sizeof(int_array2) / sizeof(int); ++i) {
        vec_array2.push_back(int_array2[i]);
    }

    Solution solution;
    vector<int> result1 = solution.subarraySum2(vec_array1, 33);
    vector<int> result2 = solution.subarraySum2(vec_array2, 7);

    cout << "result1 = [" << result1[0] << " , " << result1[1] << " ]" << endl;
    cout << "result2 = [" << result2[0] << " , " << result2[1] << " ]" << endl;

    return 0;
}

```

```
for , curr_sum > k while curr_sum , left_index , curr_sum bug, (@glbtchen  
bug)
```

$O(2n)$, $O(1)$.

Reference

- Find subarray with given sum - GeeksforGeeks

Subarray Sum Closest

Source

- lintcode: [\(139\) Subarray Sum Closest](#)

Given an integer array, find a subarray with sum closest to zero.
Return the indexes of the first number and last number.

Example

Given [-3, 1, 1, -3, 5], return [0, 2], [1, 3], [1, 1], [2, 2] or [0, 4]

Challenge

$O(n \log n)$ time

[Zero Sum Subarray | Data Structure and Algorithm](#) 4 -

$O(n \log n)$

- 1.
- 2.
- 3.

C++

```
class Solution {
public:
    /**
     * @param nums: A list of integers
     * @return: A list of integers includes the index of the first number
     *          and the index of the last number
     */
    vector<int> subarraySumClosest(vector<int> nums){
        vector<int> result;
        if (nums.empty()) {
            return result;
        }

        const int num_size = nums.size();
        vector<pair<int, int>> sum_index(num_size + 1);

        for (int i = 0; i < num_size; ++i) {
            sum_index[i + 1].first = sum_index[i].first + nums[i];
            sum_index[i + 1].second = i + 1;
        }

        sort(sum_index.begin(), sum_index.end());

        int min_diff = INT_MAX;
```

```

int closest_index = 1;
for (int i = 1; i < num_size + 1; ++i) {
    int sum_diff = abs(sum_index[i].first - sum_index[i - 1].first);
    if (min_diff > sum_diff) {
        min_diff = sum_diff;
        closest_index = i;
    }
}

int left_index = min(sum_index[closest_index - 1].second,
                     sum_index[closest_index].second);
int right_index = -1 + max(sum_index[closest_index - 1].second,
                           sum_index[closest_index].second);
result.push_back(left_index);
result.push_back(right_index);
return result;
}
};

dummy           sum_index num_size + 1 vector      resize      sum_index
left_index  right_index

```

1. $O(n)$, $O(n + 1)$.
2. $O(n \log n)$.
3. $O(n)$.

$O(n \log n)$, $O(n)$.

-
- algorithm - How to find the subarray that has sum closest to zero or a certain value t in $O(n \log n)$ - Stack Overflow

Recover Rotated Sorted Array

Source

- lintcode: (39) Recover Rotated Sorted Array

Given a rotated sorted array, recover it to sorted array in-place.

Example

[4, 5, 1, 2, 3] -> [1, 2, 3, 4, 5]

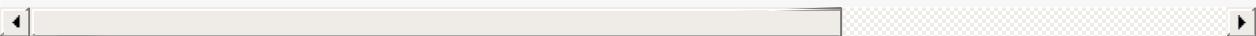
Challenge

In-place, O(1) extra space and O(n) time.

Clarification

What is rotated array:

- For example, the orginal array is [1,2,3,4], The rotated array of it can be [1,2,3,4]



[4, 5, 1, 2, 3]

1. 5 1
2. 4, 5 5, 4 1, 2, 3 3, 2, 1 [5, 4, 3, 2, 1]
3. [1, 2, 3, 4, 5]

3in-placefor

Java

```
public class Solution {
    /**
     * @param nums: The rotated sorted array
     * @return: The recovered sorted array
     */
    public void recoverRotatedSortedArray(ArrayList<Integer> nums) {
        if (nums == null || nums.size() <= 1) {
            return;
        }

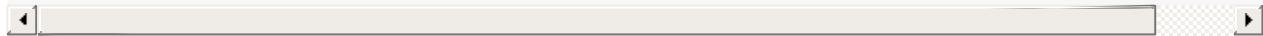
        int pos = 1;
        while (pos < nums.size()) { // find the break point
            if (nums.get(pos - 1) > nums.get(pos)) {
                break;
            }
            pos++;
        }
        myRotate(nums, 0, pos - 1);
        myRotate(nums, pos, nums.size() - 1);
        myRotate(nums, 0, nums.size() - 1);
    }
}
```

```

    }

    private void myRotate(ArrayList<Integer> nums, int left, int right) { // in-place rot
        while (left < right) {
            int temp = nums.get(left);
            nums.set(left, nums.get(right));
            nums.set(right, temp);
            left++;
            right--;
        }
    }
}

```



C++

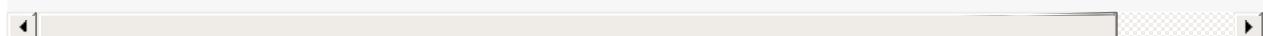
```

/*
 * forked from
 * http://www.jiuzhang.com/solutions/recover-rotated-sorted-array/
 */
class Solution {
private:
    void reverse(vector<int> &nums, vector<int>::size_type start, vector<int>::size_type
        for (vector<int>::size_type i = start, j = end; i < j; ++i, --j) {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
    }

public:
    void recoverRotatedSortedArray(vector<int> &nums) {
        for (vector<int>::size_type index = 0; index != nums.size() - 1; ++index) {
            if (nums[index] > nums[index + 1]) {
                reverse(nums, 0, index);
                reverse(nums, index + 1, nums.size() - 1);
                reverse(nums, 0, nums.size() - 1);

                return;
            }
        }
    }
};

```



`vector<int>::size_type int`

Product of Array Exclude Itself

Source

- lintcode: [\(50\) Product of Array Exclude Itself](#)
- GeeksforGeeks: [A Product Array Puzzle - GeeksforGeeks](#)

Given an integers array A.

Define $B[i] = A[0] * \dots * A[i-1] * A[i+1] * \dots * A[n-1]$, calculate B WITHOUT divide ope

Example

For A=[1, 2, 3], return [6, 3, 2].



1 -

$$result[i] = left[i] \cdot right[i], \quad left[i] = \prod_{j=0}^{i-1} A[j], right[i] = \prod_{j=i+1}^{n-1} A[j].$$

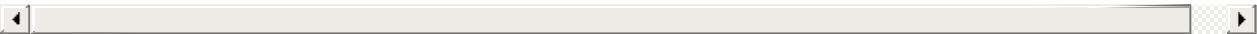
C++

```
class Solution {
public:
    /**
     * @param A: Given an integers array A
     * @return: A long long array B and B[i]= A[0] * ... * A[i-1] * A[i+1] * ... * A[n-1]
     */
    vector<long long> productExcludeItself(vector<int> &nums) {
        const int nums_size = nums.size();
        vector<long long> result(nums_size, 1);
        if (nums.empty() || nums_size == 1) {
            return result;
        }

        vector<long long> left(nums_size, 1);
        vector<long long> right(nums_size, 1);
        for (int i = 1; i != nums_size; ++i) {
            left[i] = left[i - 1] * nums[i - 1];
            right[nums_size - i - 1] = right[nums_size - i] * nums[nums_size - i];
        }
        for (int i = 0; i != nums_size; ++i) {
            result[i] = left[i] * right[i];
        }

        return result;
    }
};
```





```
for
```

$O(n)$. $O(2n)$.

2 -

1 result

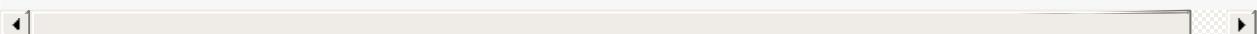
C++

```
class Solution {
public:
    /**
     * @param A: Given an integers array A
     * @return: A long long array B and B[i]= A[0] * ... * A[i-1] * A[i+1] * ... * A[n-1]
     */
    vector<long long> productExcludeItself(vector<int> &nums) {
        const int nums_size = nums.size();
        vector<long long> result(nums_size, 1);

        // solve the left part first
        for (int i = 1; i < nums_size; ++i) {
            result[i] = result[i - 1] * nums[i - 1];
        }

        // solve the right part
        long long temp = 1;
        for (int i = nums_size - 1; i >= 0; --i) {
            result[i] *= temp;
            temp *= nums[i];
        }

        return result;
    }
};
```



temp temp long long ,

$O(1)$.

Partition Array

Source

- (31) Partition Array

Given an array `nums` of integers and an int `k`, partition the array (i.e move the elements in "nums") such that:

All elements < `k` are moved to the left

All elements $\geq k$ are moved to the right

Return the partitioning index, i.e the first index `i` $nums[i] \geq k$.

Example

If `nums=[3,2,2,1]` and `k=2`, a valid answer is 1.

Note

You should do really partition in array `nums` instead of just counting the numbers of integers smaller than `k`.

If all elements in `nums` are smaller than `k`, then return `nums.length`

Challenge

Can you partition the array in-place and in $O(n)$?

1 -

`right` `k` `i` `i >= right` , `right`

C++

```
class Solution {
public:
    int partitionArray(vector<int> &nums, int k) {
        int right = 0;
        const int size = nums.size();
        for (int i = 0; i < size; ++i) {
            if (nums[i] < k && i >= right) {
                int temp = nums[i];
                nums[i] = nums[right];
                nums[right] = temp;
                ++right;
            }
        }
        return right;
    }
};
```

```
k      right    right  right k. if i >= right i > right , k bug. bug
~
```

$O(n)$,

2 -

Quick Sort $left, right \ k$ $left > right.$

C++

```
class Solution {
public:
    int partitionArray(vector<int> &nums, int k) {
        int left = 0, right = nums.size() - 1;
        while (left <= right) {
            while (left <= right && nums[left] < k) ++left;
            while (left <= right && nums[right] >= k) --right;
            if (left <= right) {
                int temp = nums[left];
                nums[left] = nums[right];
                nums[right] = temp;
                ++left;
                --right;
            }
        }
        return left;
    }
};
```

$left <= right, k$ while
 $left, right .$ $left$ $nums$ $left = 0$ $left$ $right .$

$O(n), 12 k$

Reference

- [Partition Array |](#)

First Missing Positive

Source

- leetcode: [First Missing Positive | LeetCode OJ](#)
- lintcode: [\(189\) First Missing Positive](#)

Given an unsorted integer array, find the first missing positive integer.

Example

Given [1,2,0] return 3, and [3,4,-1,1] return 2.

Challenge

Your algorithm should run in $O(n)$ time and uses constant space.

$O(n \log n)$,

$O(n)$ $O(n)$

12()

$$A[i] = x, \quad A[x-1] = x, \quad A[x-1] = A[i].$$

$$f[i] = i + 1, 1$$

C++

```
class Solution {
public:
    /**
     * @param A: a vector of integers
     * @return: an integer
     */
    int firstMissingPositive(vector<int> A) {
        const int size = A.size();

        for (int i = 0; i < size; ++i) {
            while (A[i] > 0 && A[i] <= size && \
                   (A[i] != i + 1) && (A[i] != A[A[i] - 1])) {
                int temp = A[A[i] - 1];
                A[A[i] - 1] = A[i];
                A[i] = temp;
            }
        }

        for (int i = 0; i < size; ++i) {
            if (A[i] != i + 1) {

```

```

        return i + 1;
    }

    return size + 1;
}
};

```

1. A[i] ...
2. A[i] \leq size
3. A[i] != i + 1 ,
4. A[i] != A[A[i] - 1] ,

while i

```

int temp = A[i];
A[i] = A[A[i] - 1];
A[A[i] - 1] = temp;

```

bug :(A[i]

11

while $O(n)$. $O(n)$, temp $O(1)$.

Reference

- [Find First Missing Positive | N00tc0d3r](#)
- [LeetCode: First Missing Positive - Yu's Garden -](#)
- [First Missing Positive |](#)

2 Sum

Source

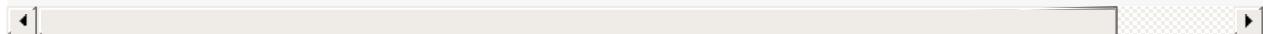
- leetcode: [Two Sum | LeetCode OJ](#)
- lintcode: [\(56\) 2 Sum](#)

Given an array of integers, find two numbers such that they add up to a specific target n .

The function `twoSum` should return indices of the two numbers such that they add up to the target, where `index1` must be less than `index2`. Please note that your returned answers (both `index1` and `index2`) are not zero-based.

You may assume that each input would have exactly one solution.

Input: `numbers={2, 7, 11, 15}`, `target=9`
 Output: `index1=1, index2=2`



1 -

target , target too naive... target $O(n^2)$, target —
 $x_i + x_j = \text{target}$, $x_i = \text{target} - x_j$, $i \quad j \dots$
 $0 \quad \quad \quad x_j, \quad \quad \quad \text{target}$

C++

```
class Solution {
public:
    /*
     * @param numbers : An array of Integer
     * @param target : target = numbers[index1] + numbers[index2]
     * @return : [index1+1, index2+1] (index1 < index2)
     */
    vector<int> twoSum(vector<int> &nums, int target) {
        vector<int> result;
        const int length = nums.size();
        if (0 == length) {
            return result;
        }

        // first value, second index
        unordered_map<int, int> hash(length);
        for (int i = 0; i != length; ++i) {
            if (hash.find(target - nums[i]) != hash.end()) {
                result.push_back(hash[target - nums[i]]);
                result.push_back(i + 1);
            }
        }
    }
}
```

```

        return result;
    } else {
        hash[nums[i]] = i + 1;
    }
}

return result;
};


```

- 1.
2. C++ 11 `unordered_map`
- 3.

$O(n)$, $O(n)$.

Python

```

class Solution:
    """
    @param numbers : An array of Integer
    @param target : target = numbers[index1] + numbers[index2]
    @return : [index1 + 1, index2 + 1] (index1 < index2)
    """
    def twoSum(self, numbers, target):
        hashdict = {}
        for i, item in enumerate(numbers):
            if (target - item) in hashdict:
                return (hashdict[target - item] + 1, i + 1)
            hashdict[item] = i

        return (-1, -1)

```

Python `dict` `enumerate` `list`, `tuple` $(-1, -1)$.

2 -

C++

```

class Solution {
public:

```

```

/*
 * @param numbers : An array of Integer
 * @param target : target = numbers[index1] + numbers[index2]
 * @return : [index1+1, index2+1] (index1 < index2)
 */
vector<int> twoSum(vector<int> &nums, int target) {
    vector<int> result;
    const int length = nums.size();
    if (0 == length) {
        return result;
    }

    // first num, second is index
    vector<pair<int, int>> num_index(length);
    // map num value and index
    for (int i = 0; i != length; ++i) {
        num_index[i].first = nums[i];
        num_index[i].second = i + 1;
    }

    sort(num_index.begin(), num_index.end());
    int start = 0, end = length - 1;
    while (start < end) {
        if (num_index[start].first + num_index[end].first > target) {
            --end;
        } else if (num_index[start].first + num_index[end].first == target) {
            int min_index = min(num_index[start].second, num_index[end].second);
            int max_index = max(num_index[start].second, num_index[end].second);
            result.push_back(min_index);
            result.push_back(max_index);
            return result;
        } else {
            ++start;
        }
    }

    return result;
}
};

```

1.

2. `length` `nums.size()`
3. `pair`
- 4.
- 5.

`pair` $O(n)$, $O(n)$, $O(n \log n)$, $O(n)$.

lintcode	$O(n \log n)$	$O(1)$,	$O(n)$	$O(1)$	$O(n)$	$O(n)$
----------	---------------	----------	--------	--------	--------	--------

3 Sum

Source

- leetcode: [3Sum | LeetCode OJ](#)
- lintcode: [\(57\) 3 Sum](#)

Given an array S of n integers, are there elements a, b, c in S such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Example

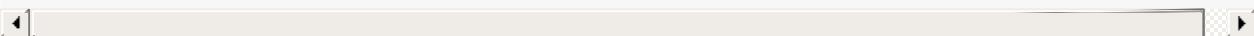
For example, given array S = {-1 0 1 2 -1 -4}, A solution set is:

```
(-1, 0, 1)
(-1, -1, 2)
```

Note

Elements in a triplet (a,b,c) must be in non-descending order. (ie, a ≤ b ≤ c)

The solution set must not contain duplicate triplets.



1 - + + 2 Sum

[2 Sum](#), [3 Sum](#) [2 Sum](#) [1 Sum](#) [3 Sum](#) [1 Sum](#) [+ 2 Sum](#) [2 Sum](#)

Python

```
class Solution:
    """
    @param numbersbers : Give an array numbersbers of n integer
    @return : Find all unique triplets in the array which gives the sum of zero.
    """
    def threeSum(self, numbers):
        triplets = []
        length = len(numbers)
        if length < 3:
            return triplets

        numbers.sort()
        for i in xrange(length):
            target = 0 - numbers[i]
            # 2 Sum
            hashmap = {}
            for j in xrange(i + 1, length):
                item_j = numbers[j]
                if (target - item_j) in hashmap:
                    triplet = [numbers[i], target - item_j, item_j]
                    if triplet not in triplets:
                        triplets.append(triplet)
```

```

        else:
            hashmap[item_j] = j

    return triplets

```

1. 3
- 2.
3. 2 Sum
- 4.

2 Sum

$O(n \log n)$, for $O(n^2)0$ $O(n)$.

leetcode 500 + ms,

C++

```

class Solution {
public:
    vector<vector<int>> threeSum(vector<int> &num)
    {
        vector<vector<int>> result;
        if (num.size() < 3) return result;

        int ans = 0;

        sort(num.begin(), num.end());

        for (int i = 0; i < num.size() - 2; ++i)
        {
            if (i > 0 && num[i] == num[i - 1])
                continue;
            int j = i + 1;
            int k = num.size() - 1;

            while (j < k)
            {
                ans = num[i] + num[j] + num[k];

                if (ans == 0)
                {
                    result.push_back({num[i], num[j], num[k]});
                    ++j;
                    while (j < num.size() && num[j] == num[j - 1])
                        ++j;
                    --k;
                    while (k >= 0 && num[k] == num[k + 1])
                        --k;
                }
            }
        }
    }
};

```

```

        }
        else if (ans > 0)
            --k;
        else
            ++j;
    }

    return result;
}
};

```

pythonhash map

```

S = {-1 0 1 2 -1 -4}

S = {-4 -1 -1 0 1 2}
      ↑   ↑       ↑
      i   j       k
      →           ←
ijkS[i]+S[j]+S[k]=ans0js[j]ks[k]
ans>0S[k]kans<0S[j]jans==0

```

in,jkn-i $O(n^2)$ 52ms

Reference

- [3Sum |](#)
- [A simply Python version based on 2sum - O\(n^2\) - Leetcode Discuss](#)

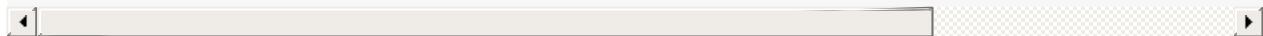
3 Sum Closest

Source

- leetcode: [3Sum Closest | LeetCode OJ](#)
- lintcode: [\(59\) 3 Sum Closest](#)

Given an array S of n integers, find three integers in S such that the sum is closest to 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).



1 - + 2 Sum + +

3 Sum 3 Sum 1 Sum + 2 Sum Closest Cloest

Python

```
class Solution:
    """
    @param numbers: Give an array numbers of n integer
    @param target : An integer
    @return : return the sum of the three integers, the sum closest target.
    """
    def threeSumClosest(self, numbers, target):
        result = 2**31 - 1
        length = len(numbers)
        if length < 3:
            return result

        numbers.sort()
        larger_count = 0
        for i, item_i in enumerate(numbers):
            start = i + 1
            end = length - 1
            # optimization 1 - filter the smallest sum greater than target
            if start < end:
                sum3_smallest = numbers[start] + numbers[start + 1] + item_i
                if sum3_smallest > target:
                    larger_count += 1
                    if larger_count > 1:
                        return result

            while (start < end):
                sum3 = numbers[start] + numbers[end] + item_i
                if abs(sum3 - target) < abs(result - target):
                    result = sum3
```

```

# optimization 2 - filter the sum3 closest to target
sum_flag = 0
if sum3 > target:
    end -= 1
    if sum_flag == -1:
        break
    sum_flag = 1
elif sum3 < target:
    start += 1
    if sum_flag == 1:
        break
    sum_flag = -1
else:
    return result

return result

```

1. leetcode sys result
- 2.
3. item_i 2 Sum Cloest2 Sum Cloest i + 1
4. —— target target
5. 2 Sum Cloest sum3 result target result
6. 2 Sum Cloest target

$O(n \log n)$, for $O(n^2)$, result target $O(1)$.

C++

```

class Solution {
public:
    int threeSumClosest(vector<int> &num, int target)
    {
        if (num.size() <= 3) return accumulate(num.begin(), num.end(), 0);
        sort (num.begin(), num.end());

        int result = 0, n = num.size(), temp;
        result = num[0] + num[1] + num[2];
        for (int i = 0; i < n - 2; ++i)
        {
            int j = i + 1, k = n - 1;
            while (j < k)
            {
                temp = num[i] + num[j] + num[k];

                if (abs(target - result) > abs(target - temp))
                    result = temp;
                if (result == target)
                    return result;
            }
        }
    }
};

```

```
        ( temp > target ) ? --k : ++j;
    }
}
return result;
};

};
```

3Sumi,j,k

3sumtarget0target

$O(n^2)$ 16ms

Reference

- [3Sum Closest |](#)

Remove Duplicates from Sorted Array

Source

- leetcode: Remove Duplicates from Sorted Array | LeetCode OJ
- lintcode: (100) Remove Duplicates from Sorted Array

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this in place with constant memory.

For example,
Given input array A = [1,1,2],

Your function should return length = 2, and A is now [1,2].

Example

000

C++

```
class Solution {
public:
    /**
     * @param A: a list of integers
     * @return : return an integer
     */
    int removeDuplicates(vector<int> &nums) {
        if (nums.size() <= 1) return nums.size();

        int len = nums.size();
        int newIndex = 0;
        for (int i = 0; i < len; ++i) {
            if (nums[i] != nums[newIndex]) {
                newIndex++;
                nums[newIndex] = nums[i];
            }
        }

        return newIndex + 1;
    }
};
```

Java

```
public class Solution {  
    /**  
     * @param A: a array of integers  
     * @return : return an integer  
     */  
    public int removeDuplicates(int[] nums) {  
        if (nums == null) return -1;  
        if (nums.length <= 1) return nums.length;  
  
        int newIndex = 0;  
        for (int i = 0; i < nums.length; i++) {  
            if (nums[i] != nums[newIndex]) {  
                newIndex++;  
                nums[newIndex] = nums[i];  
            }  
        }  
  
        return newIndex + 1;  
    }  
}
```

1

$O(n)$, $O(1)$.

Remove Duplicates from Sorted Array II

Source

- leetcode: Remove Duplicates from Sorted Array II | LeetCode OJ
- lintcode: (101) Remove Duplicates from Sorted Array II

Follow up for "Remove Duplicates":
What if duplicates are allowed at most twice?

For example,
Given sorted array A = [1,1,1,2,2,3],

Your function should return length = 5, and A is now [1,1,2,2,3].
Example

— occurrence (10) reset — @meishenme -1

C++

```
class Solution {
public:
    /**
     * @param A: a list of integers
     * @return : return an integer
     */
    int removeDuplicates(vector<int> &nums) {
        if (nums.size() <= 2) return nums.size();

        int len = nums.size();
        int newIndex = 1;
        for (int i = 2; i < len; ++i) {
            if (nums[i] != nums[newIndex] || nums[i] != nums[newIndex - 1]) {
                ++newIndex;
                nums[newIndex] = nums[i];
            }
        }

        return newIndex + 1;
    }
};
```

Java

```
public class Solution {
```

```
/*
 * @param A: a array of integers
 * @return : return an integer
 */
public int removeDuplicates(int[] nums) {
    if (nums == null) return -1;
    if (nums.length <= 2) return nums.length;

    int newIndex = 1;
    for (int i = 2; i < nums.length; i++) {
        if (nums[i] != nums[newIndex] || nums[i] != nums[newIndex - 1]) {
            newIndex++;
            nums[newIndex] = nums[i];
        }
    }

    return newIndex + 1;
}
```

i 2newIndex 1

$O(n)$, $O(1)$.

Merge Sorted Array

Source

- leetcode: Merge Sorted Array | LeetCode OJ
- lintcode: (6) Merge Sorted Array

Given two sorted integer arrays A and B, merge B into A as one sorted array.

Example

A = [1, 2, 3, empty, empty], B = [4, 5]

After merge, A will be filled as [1, 2, 3, 4, 5]

Note

You may assume that A has enough space (size that is greater or equal to m + n) to hold additional elements from B.

The number of elements initialized in A and B are m and n respectively.

in-place

$O(n^2)$ A A B

m == 0

n == 0 B A

A

Python

```
class Solution:
    """
    @param A: sorted integer array A which has m elements,
              but size of A is m+n
    @param B: sorted integer array B which has n elements
    @return: void
    """
    def mergeSortedArray(self, A, m, B, n):
        if B is None:
            return A

        index = m + n - 1
        while m > 0 and n > 0:
            if A[m - 1] > B[n - 1]:
                A[index] = A[m - 1]
                m -= 1
            else:
                A[index] = B[n - 1]
                n -= 1
            index -= 1

        # B has elements left
        while n > 0:
```

```

A[index] = B[n - 1]
n -= 1
index -= 1

```

C++

```

class Solution {
public:
    /**
     * @param A: sorted integer array A which has m elements,
     *           but size of A is m+n
     * @param B: sorted integer array B which has n elements
     * @return: void
     */
    void mergeSortedArray(int A[], int m, int B[], int n) {
        int index = m + n - 1;
        while (m > 0 && n > 0) {
            if (A[m - 1] > B[n - 1]) {
                A[index] = A[m - 1];
                --m;
            } else {
                A[index] = B[n - 1];
                --n;
            }
            --index;
        }

        // B has elements left
        while (n > 0) {
            A[index] = B[n - 1];
            --n;
            --index;
        }
    }
};

```

Java

```

class Solution {
    /**
     * @param A: sorted integer array A which has m elements,
     *           but size of A is m+n
     * @param B: sorted integer array B which has n elements
     * @return: void
     */
    public void mergeSortedArray(int[] A, int m, int[] B, int n) {
        if (A == null || B == null) return;

        int index = m + n - 1;
        while (m > 0 && n > 0) {
            if (A[m - 1] > B[n - 1]) {
                A[index] = A[m - 1];
                m--;
            } else {

```

```
        A[index] = B[n - 1];
        n--;
    }
    index--;
}

// B has elements left
while (n > 0) {
    A[index] = B[n - 1];
    n--;
    index--;
}
}
```

while

$O(n)$. $O(1)$.

Merge Sorted Array II

Source

- lintcode: [\(64\) Merge Sorted Array II](#)

Merge two given sorted integer array A and B into a new sorted integer array.

Example

A=[1, 2, 3, 4]

B=[2, 4, 5, 6]

return [1, 2, 2, 3, 4, 4, 5, 6]

Challenge

How can you optimize your algorithm

if one array is very large and the other is very small?

in-place,

Python

```
class Solution:
    #param A and B: sorted integer array A and B.
    #return: A new sorted integer array
    def mergeSortedArray(self, A, B):
        if A is None or len(A) == 0:
            return B
        if B is None or len(B) == 0:
            return A

        C = []
        aLen, bLen = len(A), len(B)
        i, j = 0, 0
        while i < aLen and j < bLen:
            if A[i] < B[j]:
                C.append(A[i])
                i += 1
            else:
                C.append(B[j])
                j += 1

        # A has elements left
        while i < aLen:
            C.append(A[i])
            i += 1
```

```

# B has elements left
while j < bLen:
    C.append(B[j])
    j += 1

return C

```

C++

```

class Solution {
public:
    /**
     * @param A and B: sorted integer array A and B.
     * @return: A new sorted integer array
     */
    vector<int> mergeSortedArray(vector<int> &A, vector<int> &B) {
        if (A.empty()) return B;
        if (B.empty()) return A;

        int aLen = A.size(), bLen = B.size();
        vector<int> C;
        int i = 0, j = 0;
        while (i < aLen && j < bLen) {
            if (A[i] < B[j]) {
                C.push_back(A[i]);
                ++i;
            } else {
                C.push_back(B[j]);
                ++j;
            }
        }

        // A has elements left
        while (i < aLen) {
            C.push_back(A[i]);
            ++i;
        }

        // B has elements left
        while (j < bLen) {
            C.push_back(B[j]);
            ++j;
        }

        return C;
    }
};

```

Java

```

class Solution {
    /**
     * @param A and B: sorted integer array A and B.
     * @return: A new sorted integer array
     */

```

```

/*
public ArrayList<Integer> mergeSortedArray(ArrayList<Integer> A, ArrayList<Integer> B
    if (A == null || A.isEmpty()) return B;
    if (B == null || B.isEmpty()) return A;

    ArrayList<Integer> C = new ArrayList<Integer>();
    int aLen = A.size(), bLen = B.size();
    int i = 0, j = 0;
    while (i < aLen && j < bLen) {
        if (A.get(i) < B.get(j)) {
            C.add(A.get(i));
            i++;
        } else {
            C.add(B.get(j));
            j++;
        }
    }

    // A has elements left
    while (i < aLen) {
        C.add(A.get(i));
        i++;
    }

    // B has elements left
    while (j < bLen) {
        C.add(B.get(j));
        j++;
    }

    return C;
}
}

```

A, B $O(n)$, $O(1)$.

Challenge

Merge

Median

Source

- lintcode: [\(80\) Median](#)

Given a unsorted array with integers, find the median of it.

A median is the middle number of the array after it is sorted.

If there are even numbers in the array, return the $N/2$ -th number after sorted.

Example

Given [4, 5, 1, 2, 3], return 3

Given [7, 9, 4, 5], return 5

Challenge

$O(n)$ time.

$O(n \log n)$,

[Quick Sort, \(\)\(\)](#)

Python

```
class Solution:
    """
    @param nums: A list of integers.
    @return: An integer denotes the middle number of the array.
    """
    def median(self, nums):
        if not nums:
            return -1
        return self.helper(nums, 0, len(nums) - 1, (1 + len(nums)) / 2)

    def helper(self, nums, l, u, size):
        if l >= u:
            return nums[u]

        m = l
        for i in xrange(l + 1, u + 1):
            if nums[i] < nums[l]:
                m += 1
                nums[m], nums[i] = nums[i], nums[m]

        # swap between m and l after partition, important!
        nums[m], nums[l] = nums[l], nums[m]
```

```

    if m - l + 1 == size:
        return nums[m]
    elif m - l + 1 > size:
        return self.helper(nums, l, m - 1, size)
    else:
        return self.helper(nums, m + 1, u, size - (m - l + 1))
}

```

C++

```

class Solution {
public:
    /**
     * @param nums: A list of integers.
     * @return: An integer denotes the middle number of the array.
     */
    int median(vector<int> &nums) {
        if (nums.empty()) return 0;

        int len = nums.size();
        return helper(nums, 0, len - 1, (len + 1) / 2);
    }

private:
    int helper(vector<int> &nums, int l, int u, int size) {
        // if (l >= u) return nums[u];

        int m = l; // index m to track pivot
        for (int i = l + 1; i <= u; ++i) {
            if (nums[i] < nums[l]) {
                ++m;
                int temp = nums[i];
                nums[i] = nums[m];
                nums[m] = temp;
            }
        }

        // swap with the pivot
        int temp = nums[m];
        nums[m] = nums[l];
        nums[l] = temp;

        if (m - l + 1 == size) {
            return nums[m];
        } else if (m - l + 1 > size) {
            return helper(nums, l, m - 1, size);
        } else {
            return helper(nums, m + 1, u, size - (m - l + 1));
        }
    }
};

```

Java

```

public class Solution {
    /**
     * @param nums: A list of integers.
     * @return: An integer denotes the middle number of the array.
     */
    public int median(int[] nums) {
        if (nums == null) return -1;

        return helper(nums, 0, nums.length - 1, (nums.length + 1) / 2);
    }

    // l: lower, u: upper, m: median
    private int helper(int[] nums, int l, int u, int size) {
        if (l >= u) return nums[u];

        int m = l;
        for (int i = l + 1; i <= u; i++) {
            if (nums[i] < nums[l]) {
                m++;
                int temp = nums[m];
                nums[m] = nums[i];
                nums[i] = temp;
            }
        }
        // swap between array[m] and array[l]
        // put pivot in the mid
        int temp = nums[m];
        nums[m] = nums[l];
        nums[l] = temp;

        if (m - l + 1 == size) {
            return nums[m];
        } else if (m - l + 1 > size) {
            return helper(nums, l, m - 1, size);
        } else {
            return helper(nums, m + 1, u, size - (m - l + 1));
        }
    }
}

```

00

size (len(nums) + 1) / 2 , m - 1 + 1 == size , l ()()	(len(nums) + 1) / 2 m -
1 + 1 > size , l size ; m - 1 + 1 < size , size size - (m - 1 + 1) ,	
size m - 1 + 1 .	

$$m \quad O(n(1 + 1/2 + 1/4 + \dots)) = O(2n), \quad O(1),$$

Partition Array by Odd and Even

Source

- lintcode: [\(373\) Partition Array by Odd and Even](#)
- Segregate Even and Odd numbers - GeeksforGeeks

Partition an integers array into odd number first and even number second.

Example

Given [1, 2, 3, 4], return [1, 3, 2, 4]

Challenge

Do it in-place.

Java

```
public class Solution {
    /**
     * @param nums: an array of integers
     * @return: nothing
     */
    public void partitionArray(int[] nums) {
        if (nums == null) return;

        int left = 0, right = nums.length - 1;
        while (left < right) {
            // odd number
            while (left < right && nums[left] % 2 != 0) {
                left++;
            }
            // even number
            while (left < right && nums[right] % 2 == 0) {
                right--;
            }
            // swap
            if (left < right) {
                int temp = nums[left];
                nums[left] = nums[right];
                nums[right] = temp;
            }
        }
    }
}
```

`left < right .`

$O(n)$, $O(1)$.

Kth Largest Element

Source

- leetcode: Kth Largest Element in an Array | LeetCode OJ
- lintcode: (5) Kth Largest Element

Find K-th largest element in an array.

Example

In array [9,3,2,4,8], the 3rd largest element is 4.

In array [1,2,3,4,5], the 1st largest element is 5,
2nd largest element is 4, 3rd largest element is 3 and etc.

Note

You can swap elements in the array

Challenge

$O(n)$ time, $O(1)$ extra memory.

K $O(n)$, K Quick Sort K

Java

```
class Solution {
    //param k : description of k
    //param numbers : array of numbers
    //return: description of return
    public int kthLargestElement(int k, ArrayList<Integer> numbers) {
        if (numbers == null || numbers.isEmpty()) return -1;

        int result = qSort(numbers, 0, numbers.size() - 1, k);
        return result;
    }

    private int qSort(ArrayList<Integer> nums, int l, int u, int k) {
        // l should not greater than u
        if (l >= u) return nums.get(u);

        // index m of nums
        int m = l;
        for (int i = l + 1; i <= u; i++) {
            if (nums.get(i) > nums.get(l)) {
                m++;
                Collections.swap(nums, m, i);
            }
        }
    }
}
```

```

Collections.swap(nums, m, l);

if (m + 1 == k) {
    return nums.get(m);
} else if (m + 1 > k) {
    return qSort(nums, l, m - 1, k);
} else {
    return qSort(nums, m + 1, u, k);
}
}
}

```

$(l > u), \quad m + 1 == k . K \text{ for } \quad \text{nums.get}(i) > \text{nums.get}(l)$

$n + n - 1 + \dots + 1 = O(n^2), \quad n + n/2 + n/4 + \dots + 1 = O(2n) = O(n). \quad O(n).$
 $O(1).$

Search -

-
- 1find the first/last position of...2O(lgn)
-

Binary Search -

Source

- lintcode: [lintcode - \(14\) Binary Search](#)

Binary search is a famous question in algorithm.

For a given sorted array (ascending order) and a target number, find the first index of t

If the target number does not exist in the array, return -1.

Example

If the array is [1, 2, 3, 3, 4, 5, 10], for given target 3, return 2.

Challenge

If the count of numbers is bigger than MAXINT, can your code work properly?



Java

```
/**
 * fork
 * http://www.jiuzhang.com//solutions/binary-search/
 */
class Solution {
    /**
     * @param nums: The integer array.
     * @param target: Target to find.
     * @return: The first position of target. Position starts from 0.
     */
    public int binarySearch(int[] nums, int target) {
        if (nums == null || nums.length == 0) {
            return -1;
        }

        int start = 0;
        int end = nums.length - 1;
        int mid;
        while (start + 1 < end) {
            mid = start + (end - start) / 2; // avoid overflow when (end + start)
            if (target < nums[mid]) {
                end = mid;
            } else if (target > nums[mid]) {
                start = mid;
            } else {

```

```
        end = mid;
    }
}

if (nums[start] == target) {
    return start;
}
if (nums[end] == target) {
    return end;
}

return -1;
}
```

1. 0
2. start, end, mid mid
- 3.
4. while start + 1 < end start <= end start == end
5. targetstartend—— first position or last position
6. end = mid
7. while start + 1 < end mid +1 -1

Search Insert Position

Source

- lintcode: [\(60\) Search Insert Position](#)

Given a sorted array and a target value, return the index if the target is found. If not,

You may assume no duplicates in the array.

Example

```
[1,3,5,6], 5 → 2
[1,3,5,6], 2 → 1
[1,3,5,6], 7 → 4
[1,3,5,6], 0 → 0
```



find the first/last position of...

Java

```
public class Solution {
    /**
     * param A : an integer sorted array
     * param target : an integer to be inserted
     * return : an integer
     */
    public int searchInsert(int[] A, int target) {
        if (A == null) {
            return -1;
        }
        if (A.length == 0) {
            return 0;
        }

        int start = 0, end = A.length - 1;
        int mid;

        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (A[mid] == target) {
                return mid; // no duplicates, if not `end = target;`
            } else if (A[mid] < target) {
                start = mid;
            } else {
                end = mid;
            }
        }
    }
}
```

```
    if (A[start] >= target) {
        return start;
    } else if (A[end] >= target) {
        return end; // in most cases
    } else {
        return end + 1; // A[end] < target;
    }
}
```

, [1,3,5,6], 7 → 4else return end + 1;

Search for a Range

Source

- lintcode: [\(61\) Search for a Range](#)

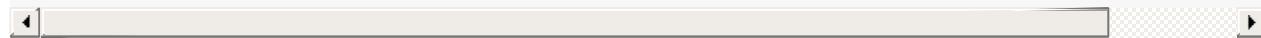
Given a sorted array of integers, find the starting and ending position of a given target

Your algorithm's runtime complexity must be in the order of $O(\log n)$.

If the target is not found in the array, return [-1, -1].

Example

Given [5, 7, 7, 8, 8, 10] and target value 8,
return [3, 4].



Search for a range first & last position

```
(target == nums[mid] end = mid start =
mid
```

Java

```
/*
 * fork
 * http://www.jiuzhang.com/solutions/search-for-a-range/
 */
public class Solution {
    /**
     *@param A : an integer sorted array
     *@param target : an integer to be inserted
     *return : a list of length 2, [index1, index2]
     */
    public ArrayList<Integer> searchRange(ArrayList<Integer> A, int target) {
        ArrayList<Integer> result = new ArrayList<Integer>();
        int start, end, mid;
        result.add(-1);
        result.add(-1);

        if (A == null || A.size() == 0) {
            return result;
        }

        // search for left bound
        start = 0;
        end = A.size() - 1;
        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (A.get(mid) == target) {
```

```

        end = mid; // set end = mid to find the minimum mid
    } else if (A.get(mid) > target) {
        end = mid;
    } else {
        start = mid;
    }
}
if (A.get(start) == target) {
    result.set(0, start);
} else if (A.get(end) == target) {
    result.set(0, end);
} else {
    return result;
}

// search for right bound
start = 0;
end = A.size() - 1;
while (start + 1 < end) {
    mid = start + (end - start) / 2;
    if (A.get(mid) == target) {
        start = mid; // set start = mid to find the maximum mid
    } else if (A.get(mid) > target) {
        end = mid;
    } else {
        start = mid;
    }
}
if (A.get(end) == target) {
    result.set(1, end);
} else if (A.get(start) == target) {
    result.set(1, start);
} else {
    return result;
}

return result;
// write your code here
}
}

```

1. 0
2. start, end, mid
- 3.
4. while start + 1 < end start <= end start == end
5. A.get(start) == target A.get(end) == target target start end start start.
6. A.get(end) == target A.get(start) == target
7. A.get(mid) == target first position end = mid last position start = mid
8. start, end

First Bad Version

Source

- lintcode: [\(74\) First Bad Version](#)

The code base version is an integer and start from 1 to n. One day, someone commit a bad You can determine whether a version is bad by the following interface:

Java:

```
public VersionControl {
    boolean isBadVersion(int version);
}
```

C++:

```
class VersionControl {
public:
    bool isBadVersion(int version);
};
```

Python:

```
class VersionControl:
    def isBadVersion(version)
```

Find the first bad version.

Note

You should call isBadVersion as few as possible.

Please read the annotation in code area to get the correct way to call isBadVersion in di

Example

Given n=5

Call isBadVersion(3), get false

Call isBadVersion(5), get true

Call isBadVersion(4), get true

return 4 is the first bad version

Challenge

Do not call isBadVersion exceed O(logn) times.



Search for a Range

Java

```
/**
 * public class VersionControl {
 *     public static boolean isBadVersion(int k);
 * }
```

```

/*
 * you can use VersionControl.isBadVersion(k) to judge whether
 * the kth code version is bad or not.
 */
class Solution {
    /**
     * @param n: An integers.
     * @return: An integer which is the first bad version.
     */
    public int findFirstBadVersion(int n) {
        // write your code here
        if (n == 0) {
            return -1;
        }

        int start = 1, end = n, mid;
        while (start + 1 < end) {
            mid = start + (end - start)/2;
            if (VersionControl.isBadVersion(mid) == false) {
                start = mid;
            } else {
                end = mid;
            }
        }

        if (VersionControl.isBadVersion(start) == true) {
            return start;
        } else if (VersionControl.isBadVersion(end) == true) {
            return end;
        } else {
            return -1; // not found
        }
    }
}

```

C++

```

/**
 * class VersionControl {
 *     public:
 *         static bool isBadVersion(int k);
 * }
 * you can use VersionControl::isBadVersion(k) to judge whether
 * the kth code version is bad or not.
 */
class Solution {
public:
    /**
     * @param n: An integers.
     * @return: An integer which is the first bad version.
     */
    int findFirstBadVersion(int n) {
        if (n < 1) {
            return -1;
        }

        int start = 1;
        int end = n;

```

```

int mid;
while (start + 1 < end) {
    mid = start + (end - start) / 2;
    if (VersionControl::isBadVersion(mid)) {
        end = mid;
    } else {
        start = mid;
    }
}

if (VersionControl::isBadVersion(start)) {
    return start;
} else if (VersionControl::isBadVersion(end)) {
    return end;
}

return -1; // find no bad version
}
};


```

Search for a Rangeendgood version

Python

```

#class VersionControl:
#    @classmethod
#    def isBadVersion(cls, id)
#        # Run unit tests to check whether verison `id` is a bad version
#        # return true if unit tests passed else false.
# You can use VersionControl.isBadVersion(10) to check whether version 10 is a
# bad version.
class Solution:
    """
    @param n: An integers.
    @return: An integer which is the first bad version.
    """
    def findFirstBadVersion(self, n):
        if n < 1:
            return -1

        start, end = 1, n
        while start + 1 < end:
            mid = start + (end - start) / 2
            if VersionControl.isBadVersion(mid):
                end = mid
            else:
                start = mid

        if VersionControl.isBadVersion(start):
            return start
        elif VersionControl.isBadVersion(end):
            return end

        return -1

```


Search a 2D Matrix

Source

- lintcode: [\(28\) Search a 2D Matrix](#)

Write an efficient algorithm that searches for a value in an $m \times n$ matrix.

This matrix has the following properties:

- * Integers in each row are sorted from left to right.
- * The first integer of each row is greater than the last integer of the previous row.

Example

Consider the following matrix:

[

```
[1, 3, 5, 7],  
[10, 11, 16, 20],  
[23, 30, 34, 50]
```

]

Given target = 3, return true.

Challenge

$O(\log(n) + \log(m))$ time



- V.S.

$$\Theta(O(\log(mn))) = O(\log(m) + \log(n))$$

Java

```
/* *  
*  
*/
```

```

* http://www.jiuzhang.com/solutions/search-a-2d-matrix
*/
// Binary Search Once
public class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        if (matrix == null || matrix.length == 0) {
            return false;
        }
        if (matrix[0] == null || matrix[0].length == 0) {
            return false;
        }

        int row = matrix.length, column = matrix[0].length;
        int start = 0, end = row * column - 1;
        int mid, number;

        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            number = matrix[mid / column][mid % column];
            if (number == target) {
                return true;
            } else if (number < target) {
                start = mid;
            } else {
                end = mid;
            }
        }

        if (matrix[start / column][start % column] == target) {
            return true;
        } else if (matrix[end / column][end % column] == target) {
            return true;
        }

        return false;
    }
}

```

1. matrixmatrix[0]
2. targetstartend

A...

Find Peak Element

Source

- leetcode: [Find Peak Element | LeetCode OJ](#)
- lintcode: [\(75\) Find Peak Element](#)

There is an integer array which has the following features:

- * The numbers in adjacent positions are different.
- * $A[0] < A[1] \&& A[A.length - 2] > A[A.length - 1]$.

We define a position P is a peek if $A[P] > A[P-1] \&& A[P] > A[P+1]$.

Find a peak element in this array. Return the index of the peak.

Note

The array may contains multiple peaks, find any of them.

Example

[1, 2, 1, 3, 4, 5, 7, 6]

return index 1 (which is number 2) or 6 (which is number 7)

Challenge

Time complexity $O(\log N)$

1 - lintcode

```
A[mid] > A[mid - 1] && A[mid] < A[mid + 1] peakA[mid] A[mid - 1] > A[mid] A[mid]
peakpeakA[mid] A[0] > A[1] > ... > A[mid - 1] > A[mid] A[
A[mid] A[mid]peak
```

first/last peak

Python

```
class Solution:
    #@param A: An integers list.
    #@return: return any of peek positions.
    def findPeak(self, A):
        if not A:
            return -1

        l, r = 0, len(A) - 1
        while l + 1 < r:
            mid = l + (r - 1) / 2
            if A[mid] < A[mid - 1]:
                r = mid
            else:
                l = mid
```

```

        elif A[mid] < A[mid + 1]:
            l = mid
        else:
            return mid
    mid = l if A[l] > A[r] else r
    return mid

```

C++

```

class Solution {
public:
    /**
     * @param A: An integers array.
     * @return: return any of peek positions.
     */
    int findPeak(vector<int> A) {
        if (A.size() == 0) return -1;

        int l = 0, r = A.size() - 1;
        while (l + 1 < r) {
            int mid = l + (r - l) / 2;
            if (A[mid] < A[mid - 1]) {
                r = mid;
            } else if (A[mid] < A[mid + 1]) {
                l = mid;
            } else {
                return mid;
            }
        }

        int mid = A[l] > A[r] ? l : r;
        return mid;
    }
};

```

Java

```

class Solution {
    /**
     * @param A: An integers array.
     * @return: return any of peek positions.
     */
    public int findPeak(int[] A) {
        if (A == null || A.length == 0) return -1;

        int l = 0, r = A.length - 1;
        while (l + 1 < r) {
            int mid = l + (r - l) / 2;
            if (A[mid] < A[mid - 1]) {
                r = mid;
            } else if (A[mid] < A[mid + 1]) {
                l = mid;
            } else {
                return mid;
            }
        }

        int mid = A[l] > A[r] ? l : r;
        return mid;
    }
};

```

2 - leetcode

leetcode lintcode

A peak element is an element that is greater than its neighbors.

Given an input array where $\text{num}[i] \neq \text{num}[i+1]$, 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 $\text{num}[-1] = \text{num}[n] = -\infty$.

For example, in array [1, 2, 3, 1], 3 is a peak element and your function should return the index number 2.

[click to show spoilers.](#)

Note:

Your solution should be in logarithmic complexity.

leetcode lintcode leetcode case $\text{num}[-1] = \text{num}[n] = -\infty$, lintcode

`num[-1] < num[0] && num[n-1] > num[n]` , —

Java

```
public class Solution {
    public int findPeakElement(int[] nums) {
        if (nums == null || nums.length == 0) return 0;

        int l = 0, r = nums.length - 1;
        while (l + 1 < r) {
            int mid = l + (r - l) / 2;
            if (nums[mid] < nums[mid - 1]) {
                // 1 peak at least in the left side
                r = mid;
            } else if (nums[mid] < nums[mid + 1]) {
                // 1 peak at least in the right side
                l = mid;
            } else {
                return mid;
            }
        }
    }
}
```

```

        mid = nums[l] > nums[r] ? l : r;
        return mid;
    }
}

```

1 < r 2.

$O(\log n)$.

Java - compact implementation [leetcode_discussion](#)

```

public class Solution {
    public int findPeakElement(int[] nums) {
        if (nums == null || nums.length == 0) {
            return -1;
        }

        int start = 0, end = nums.length - 1, mid = end / 2;
        while (start < end) {
            if (nums[mid] < nums[mid + 1]) {
                // 1 peak at least in the right side
                start = mid + 1;
            } else {
                // 1 peak at least in the left side
                end = mid;
            }
            mid = start + (end - start) / 2;
        }

        return start;
    }
}

```

C++ Java @xuewei4d

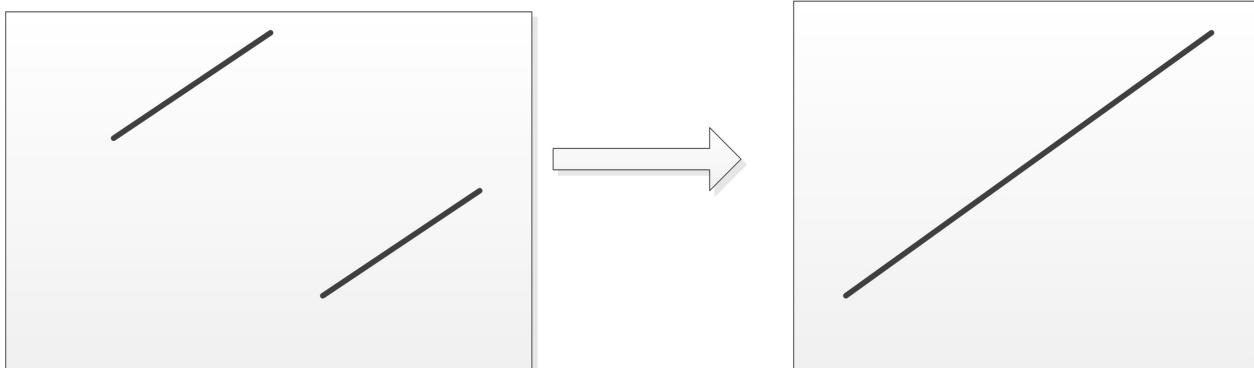
leetcode lintcode leetcode

findPeakElement lintcode findPeak

Reference

- leetcode_discussion. [Java - Binary-Search Solution - Leetcode Discuss](#) ↵

Search in Rotated Sorted Array



Source

- lintcode: [\(62\) Search in Rotated Sorted Array](#)

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Example

For `[4, 5, 1, 2, 3]` and target=1, return 2

For `[4, 5, 1, 2, 3]` and target=0, return -1



C++

```
/*
 * fork
 * http://www.jiuzhang.com/solutions/search-in-rotated-sorted-array/
 */
class Solution {
    /**
     * param A : an integer ratated sorted array
     * param target : an integer to be searched
     * return : an integer
     */
public:
```

```

int search(vector<int> &A, int target) {
    if (A.empty()) {
        return -1;
    }

    vector<int>::size_type start = 0;
    vector<int>::size_type end = A.size() - 1;
    vector<int>::size_type mid;

    while (start + 1 < end) {
        mid = start + (end - start) / 2;
        if (target == A[mid]) {
            return mid;
        }
        if (A[start] < A[mid]) {
            // situation 1, numbers between start and mid are sorted
            if (A[start] <= target && target < A[mid]) {
                end = mid;
            } else {
                start = mid;
            }
        } else {
            // situation 2, numbers between mid and end are sorted
            if (A[mid] < target && target <= A[end]) {
                start = mid;
            } else {
                end = mid;
            }
        }
    }

    if (A[start] == target) {
        return start;
    }
    if (A[end] == target) {
        return end;
    }
    return -1;
}
};

```

1. target == A[mid]
2. A[mid] > A[start] A[start] < A[mid] startmid
3. A[start]~A[mid] A[start]~A[mid] A[start] <= target <= A[mid]
4. A[mid]~A[end]
5. midwhileA[start]A[end]
6. -1.

Java

```

public class Solution {
    /**

```

```

/*@param A : an integer rotated sorted array
*@param target : an integer to be searched
*return : an integer
*/
public int search(int[] A, int target) {
    // write your code here
    if (A == null || A.length == 0) {
        return -1;
    }

    int start = 0, end = A.length - 1, mid = 0;
    while (start + 1 < end) {
        mid = start + (end - start)/2;
        if (A[mid] == target) {
            return mid;
        }
        if (A[start] < A[mid]) {//part 1
            if (A[start] <= target && target <= A[mid]) {
                end = mid;
            } else {
                start = mid;
            }
        } else { //part 2
            if (A[mid] <= target && target <= A[end]) {
                start = mid;
            } else {
                end = mid;
            }
        }
    }
    // end while

    if (A[start] == target) {
        return start;
    } else if (A[end] == target) {
        return end;
    } else {
        return -1; // not found
    }
}
}

```

Search in Rotated Sorted Array II

Source

- lintcode: [\(63\) II](#)

""

```
[3,4,4,5,7,0,1,2]target=4 true
```

A[start] < A[mid] A[start] == A[mid] start/endO(n)

C++

```
class Solution {
    /**
     * param A : an integer ratated sorted array and duplicates are allowed
     * param target : an integer to be search
     * return : a boolean
     */
public:
    bool search(vector<int> &A, int target) {
        if (A.empty()) {
            return false;
        }

        vector<int>::size_type start = 0;
        vector<int>::size_type end = A.size() - 1;
        vector<int>::size_type mid;

        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (target == A[mid]) {
                return true;
            }
            if (A[start] < A[mid]) {
                // situation 1, numbers between start and mid are sorted
                if (A[start] <= target && target < A[mid]) {
                    end = mid;
                } else {
                    start = mid;
                }
            } else if (A[start] > A[mid]) {
                // situation 2, numbers between mid and end are sorted
                if (A[mid] < target && target <= A[end]) {
                    start = mid;
                } else {
                    end = mid;
                }
            } else {
                // increment start
                ++start;
            }
        }

        if (A[start] == target || A[end] == target) {
            return true;
        }
    }
}
```

```
        return false;
    }
};
```

```
A[start] == A[mid] start
```

Find Minimum in Rotated Sorted Array

Source

- lintcode: [\(159\) Find Minimum in Rotated Sorted Array](#)

Suppose a sorted array is rotated at some pivot unknown to you beforehand.

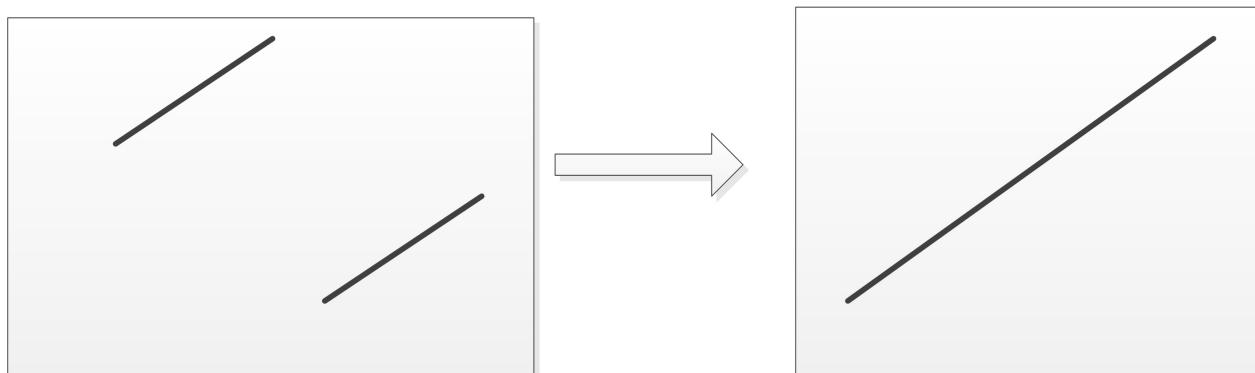
(i.e., `0 1 2 4 5 6 7` might become `4 5 6 7 0 1 2`).

Find the minimum element.

You may assume no duplicate exists in the array.

Example

Given `[4,5,6,7,0,1,2]` return `0`



target

C++

```
class Solution {
public:
    /**
     * @param num: a rotated sorted array
     * @return: the minimum number in the array
     */
    int findMin(vector<int> &num) {
        if (num.empty()) {
            return -1;
        }

        vector<int>::size_type start = 0;
        vector<int>::size_type end = num.size() - 1;
```

```

vector<int>::size_type mid;
while (start + 1 < end) {
    mid = start + (end - start) / 2;
    if (num[mid] < num[end]) {
        end = mid;
    } else {
        start = mid;
    }
}

if (num[start] < num[end]) {
    return num[start];
} else {
    return num[end];
}
}
};


```

`num[end]` `num[mid] == num[end]`

Find Minimum in Rotated Sorted Array II

Source

- lintcode: [\(160\) Find Minimum in Rotated Sorted Array II](#)

`num[mid] == num[end]` `start``end``start``end`

C++

```

class Solution {
public:
    /**
     * @param num: a rotated sorted array
     * @return: the minimum number in the array
     */
    int findMin(vector<int> &num) {
        if (num.empty()) {
            return -1;
        }

        vector<int>::size_type start = 0;
        vector<int>::size_type end = num.size() - 1;
        vector<int>::size_type mid;
        while (start + 1 < end) {
            mid = start + (end - start) / 2;
            if (num[mid] > num[end]) {

```

```
        start = mid;
    } else if (num[mid] < num[end]) {
        end = mid;
    } else {
        --end;
    }
}

if (num[start] < num[end]) {
    return num[start];
} else {
    return num[end];
}
}
};

};
```

Search a 2D Matrix II

Source

- lintcode: [\(38\) Search a 2D Matrix II](#)

Write an efficient algorithm that searches for a value in an $m \times n$ matrix, return the occurrence count.

This matrix has the following properties:

- * Integers in each row are sorted from left to right.
- * Integers in each column are sorted from up to bottom.
- * No duplicate integers in each row or column.

Example

Consider the following matrix:

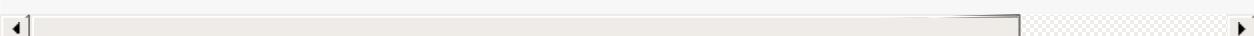
```
[  
    [1, 3, 5, 7],  
    [2, 4, 7, 8],  
    [3, 5, 9, 10]
```

]

Given target = 3, return 2.

Challenge

$O(m+n)$ time and $O(1)$ extra space



- 1. —— $O(m+n)$ time and $O(1)$ extra space
- 2. target
- 3.

C++

```
class Solution {  
public:  
    /**  
     * @param matrix: A list of lists of integers  
     * @return: An integer  
     */
```

```

* @param target: An integer you want to search in matrix
* @return: An integer indicate the total occurrence of target in the given matrix
*/
int searchMatrix(vector<vector<int>> &matrix, int target) {
    if (matrix.empty() || matrix[0].empty()) {
        return 0;
    }

    const int ROW = matrix.size();
    const int COL = matrix[0].size();

    int row = 0, col = COL - 1;
    int occur = 0;
    while (row < ROW && col >= 0) {
        if (target == matrix[row][col]) {
            ++occur;
            --col;
        } else if (target < matrix[row][col]){
            --col;
        } else {
            ++row;
        }
    }

    return occur;
}
};

```

Java

```

public class Solution {
    /**
     * @param matrix: A list of lists of integers
     * @param target: A number you want to search in the matrix
     * @return: An integer indicate the occurrence of target in the given matrix
     */
    public int searchMatrix(int[][] matrix, int target) {
        int occurrence = 0;

        if (matrix == null || matrix.length == 0) {
            return occurrence;
        }
        if (matrix[0] == null || matrix[0].length == 0) {
            return occurrence;
        }

        int row = matrix.length - 1;
        int column = matrix[0].length - 1;
        int index_row = 0, index_column = column;
        int number;

        if (target < matrix[0][0] || target > matrix[row][column]) {
            return occurrence;
        }

        while (index_row < row + 1 && index_column + 1 > 0) {
            number = matrix[index_row][index_column];

```

```
    if (target == number) {
        occurence++;
        index_column--;
    } else if (target < number) {
        index_column--;
    } else if (target > number) {
        index_row++;
    }
}

return occurence;
}
}
```

1. matrixmatrix[0]

2.

3. target

Reference

[Searching a 2D Sorted Matrix Part II | LeetCode](#)

Median of two Sorted Arrays

Source

- lintcode: [\(65\) Median of two Sorted Arrays](#)

There are two sorted arrays A and B of size m and n respectively. Find the median of the

Example

For A = [1,2,3,4,5,6] B = [2,3,4,5], the median is 3.5

For A = [1,2,3] B = [4,5], the median is 3

Challenge

Time Complexity O(logn)



"Median"? - O(log)

$k \rightarrow k = (A.length + B.length) / 2$

$O(n)kA[k/2 - 1]B[k/2 - 1]k$

1. $A[k/2 - 1] \leq B[k/2 - 1] \Rightarrow ABkA[0] \sim A[k/2 - 1]$
2. $k/2 - 1AB[0] \sim B[k/2 - 1]$

C++

```
class Solution {
public:
    /**
     * @param A: An integer array.
     * @param B: An integer array.
     * @return: a double whose format is *.5 or *.0
     */
    double findMedianSortedArrays(vector<int> A, vector<int> B) {
        if (A.empty() && B.empty()) {
            return 0;
        }

        vector<int> NonEmpty;
        if (A.empty()) {
            NonEmpty = B;
        }
        if (B.empty()) {
            NonEmpty = A;
        }

        int m = NonEmpty.size();
        int l = 0;
        int r = m - 1;
        int mid = (l + r) / 2;
        double result = NonEmpty[mid];
        if (m % 2 == 1) {
            return result;
        } else {
            l = mid;
            r = mid + 1;
            mid = (l + r) / 2;
            result = (NonEmpty[l] + NonEmpty[r]) / 2.0;
        }
        return result;
    }
}
```

```

if (!NonEmpty.empty()) {
    vector<int>::size_type len_vec = NonEmpty.size();
    return len_vec % 2 == 0 ?
        (NonEmpty[len_vec / 2 - 1] + NonEmpty[len_vec / 2]) / 2.0 :
        NonEmpty[len_vec / 2];
}

vector<int>::size_type len = A.size() + B.size();
if (len % 2 == 0) {
    return ((findKth(A, 0, B, 0, len / 2) + findKth(A, 0, B, 0, len / 2 + 1)) / 2
} else {
    return findKth(A, 0, B, 0, len / 2 + 1);
}
// write your code here
}

private:
    int findKth(vector<int> &A, vector<int>::size_type A_start, vector<int> &B, vector<int>::size_type B_start, int k) {
        if (A_start > A.size() - 1) {
            // all of the element of A are smaller than the kTh number
            return B[B_start + k - 1];
        }
        if (B_start > B.size() - 1) {
            // all of the element of B are smaller than the kTh number
            return A[A_start + k - 1];
        }

        if (k == 1) {
            return A[A_start] < B[B_start] ? A[A_start] : B[B_start];
        }

        int A_key = A_start + k / 2 - 1 < A.size() ?
            A[A_start + k / 2 - 1] : INT_MAX;
        int B_key = B_start + k / 2 - 1 < B.size() ?
            B[B_start + k / 2 - 1] : INT_MAX;

        if (A_key > B_key) {
            return findKth(A, A_start, B, B_start + k / 2, k - k / 2);
        } else {
            return findKth(A, A_start + k / 2, B, B_start, k - k / 2);
        }
    }
};

```

k

1. A_start > A.size() - 1 ABB B_start kB...
2. k1
3. A A_start k / 2 A_start + k / 2 - 1 < A.size() int
4. A_key > B_key findKth

findMedianSortedArrays

1. A, BA/B
2. A+B len / 2 len / 2 + 1len / 2 + 1

Java

```

class Solution {
    /**
     * @param A: An integer array.
     * @param B: An integer array.
     * @return: a double whose format is *.5 or *.0
     */
    public double findMedianSortedArrays(int[] A, int[] B) {
        // write your code here
        if (A.length == 0 && B.length == 0) {
            return 0;
        }
        int len = A.length + B.length;
        if (len % 2 == 0) {
            return (findKth(A, 0, B, 0, len/2) + findKth(A, 0, B, 0, len/2+1)) / 2.0;
        } else {
            return findKth(A, 0, B, 0, len/2 + 1);
        }
    }

    //find kth number of two sorted array
    public static int findKth(int[] A, int A_start, int[] B, int B_start, int k) {
        if (A_start >= A.length) {
            return B[B_start + k - 1];
        }
        if (B_start >= B.length) {
            return A[A_start + k - 1];
        }
        if (k == 1) {
            return Math.min(A[A_start], B[B_start]);
        }

        int A_key = (A_start + k/2 - 1 < A.length) // if one array is too short
            ? A[A_start + k/2 - 1] : Integer.MAX_VALUE; // trick
        int B_key = (B_start + k/2 - 1 < B.length) // if one array is too short
            ? B[B_start + k/2 - 1] : Integer.MAX_VALUE; // trick

        if (A_key < B_key) {
            return findKth(A, A_start + k/2, B, B_start, k - k/2);
        } else {
            return findKth(A, A_start, B, B_start + k/2, k - k/2);
        }
    }
}

```

- 1.
2. []A, []B, k
3. []A, []B k/2Integer.MAX_VALUE

reference

- | Median of Two Sorted Arrays
- LeetCode: Median of Two Sorted Arrays - Yu's Garden -

Sqrt x

Source

- leetcode: [Sqrt\(x\) | LeetCode OJ](#)
- lintcode: [\(141\) Sqrt\(x\)](#)

-

$$x, \quad k, \quad 1 \leq k \leq x, \quad k$$

Python

```
class Solution:
    # @param {integer} x
    # @return {integer}
    def mySqrt(self, x):
        if x < 0:
            return -1
        elif x == 0:
            return 0

        start, end = 1, x
        while start + 1 < end:
            mid = start + (end - start) / 2
            if mid**2 == x:
                return mid
            elif mid**2 > x:
                end = mid
            else:
                start = mid

        return start
```

1. 0

2. `start < end , 1`

3. `start .`

<code>start, end, mid?</code>	<code>while start + 1 < end</code>	<code>start end</code>	<code>end == 1 end == 2</code>	
1	<code>start end x k,</code>	$start \leq k \leq end$	$mid^2 == x$	<code>st</code>
<code>start</code>				

$$O(\log n), \text{ start , end , mid} \quad O(1).$$

`sqrt()` --

Wood Cut

Source

- lintcode: [\(183\) Wood Cut](#)

Given n pieces of wood with length $L[i]$ (integer array).
Cut them into small pieces to guarantee you could have equal or more than k pieces with t .
What is the longest length you can get from the n pieces of wood?
Given L & k , return the maximum length of the small pieces.

Example

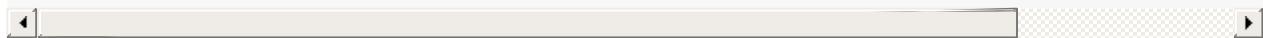
For $L=[232, 124, 456]$, $k=7$, return 114.

Note

You couldn't cut wood into float length.

Challenge

$O(n \log Len)$, where Len is the longest length of the wood.



Challenge n $L[i]$, n k k n $k=7$ L 7114,

$232/114 = 2$, $124/114 = 1$, $456/114 = 4$, $2 + 1 + 4 = 7$.

$$2, 1, 4 \quad \sum_{i=1}^n \frac{L[i]}{l} \geq k$$

$$l \quad 1 \leq l \leq \max(L[i]). 1$$

$$L[i]/l \quad L[i]$$

$$\text{Sqrt } x \quad l \quad [1, \max(L[i])]$$

Python

```
class Solution:
    """
    @param L: Given n pieces of wood with length L[i]
    @param k: An integer
    return: The maximum length of the small pieces.
    """
    def woodCut(self, L, k):
        if sum(L) < k:
            return 0

        max_len = max(L)
        start, end = 1, max_len
```

```

while start + 1 < end:
    mid = start + (end - start) / 2
    pieces_sum = sum([len_i / mid for len_i in L])
    if pieces_sum < k:
        end = mid
    else:
        start = mid

# corner case
if end == 2 and sum([len_i / 2 for len_i in L]) >= k:
    return 2

return start

```

1. L, k
 2. $start, end,$
 3. list comprehension $\sum_{i=1}^n \frac{L[i]}{l}$.
 4. $pieces_sum, k, mid, mid, mid, end.$
 5. $pieces_sum == k, mid, pieces_sum == k, mid, pieces_sum < k, start = end$
 6. $end == 2, start$
- 6 $sum(L) \geq k, end, end == 1 \text{ or } end == 2, end == 2$

$O(n), O(\log \max(L)). O(n \log \max(L)). O(1).$

Reference

- [Wood Cut](#) |

Bit Manipulation

//

Reference

- | [Matrix67: The Aha Moments](#)
- *cc150* chapter 8.5 and chapter 9.5

Single Number

Source

- lintcode: [\(82\) Single Number](#)

Given $2^n + 1$ numbers, every numbers occurs twice except one, find it.

Example

Given [1,2,2,1,3,4,3], return 4

Challenge

One-pass, constant extra space

$2^n + 1$

$x \wedge x = 0 \quad x \wedge 0 = x$

C++

```
class Solution {
public:
    /**
     * @param A: Array of integers.
     * @return: The single number.
     */
    int singleNumber(vector<int> &A) {
        if (A.empty()) {
            return -1;
        }
        int result = 0;

        for (vector<int>::iterator iter = A.begin(); iter != A.end(); ++iter) {
            result = result ^ *iter;
        }

        return result;
    }
};
```

1. (OJvector0-1)

2. $result \ 0 \quad x \wedge 0 = x$

Single Number II

Source

- lintcode: [\(83\) Single Number II](#)

Given $3n + 1$ numbers, every numbers occurs triple times except one, find it.

Example

Given [1,1,2,3,3,3,2,2,4,1] return 4

Challenge

One-pass, constant extra space

-

Single Number $3n + 1$ 0333

```
0011
0011
0011
-----
0033
```

33 int $O((3n + 1) \cdot \text{sizeof}(int) \cdot 8)$

C++ bit by bit

```
class Solution {
public:
    /**
     * @param A : An integer array
     * @return : An integer
     */
    int singleNumberII(vector<int> &A) {
        if (A.empty()) {
            return 0;
        }

        int result = 0, bit_i_sum = 0;

        for (int i = 0; i != 8 * sizeof(int); ++i) {
            bit_i_sum = 0;
            for (int j = 0; j != A.size(); ++j) {
                // get the  $i^{th}$  bit of A
                bit_i_sum += ((A[j] >> i) & 1);
            }
            // set the  $i^{th}$  bit of result
            result |= (bit_i_sum << i);
        }
        return result;
    }
}
```

```

        result |= ((bit_i_sum % 3) << i);
    }

    return result;
}
};

```

- 1.
2. result int 1 i $8 \cdot \text{sizeof}(int) - 1$, =>
3. i 3 result i result 0

Reference

[Single Number II - Leetcode Discuss](#)

Given an array of integers, every element appears k times except for one. Find that singl

@ranmocy

We need a array $x[i]$ with size k for saving the bits appears i times. For every input number a , generate the new counter by $x[j] = (x[j-1] \& a) | (x[j] \& \sim a)$. Except $x[0] = (x[k] \& a) | (x[0] \& \sim a)$.

In the equation, the first part indicates the carries from previous one. The second part indicates the bits not carried to next one.

Then the algorithms run in $O(kn)$ and the extra space $O(k)$.

Java

```

public class Solution {
    public int singleNumber(int[] A, int k, int l) {
        if (A == null) return 0;
        int t;
        int[] x = new int[k];
        x[0] = ~0;
        for (int i = 0; i < A.length; i++) {
            t = x[k-1];
            for (int j = k-1; j > 0; j--) {
                x[j] = (x[j-1] & A[i]) | (x[j] & ~A[i]);
            }
            x[0] = (t & A[i]) | (x[0] & ~A[i]);
        }
        return x[l];
    }
}

```


Single Number III

Source

- lintcode: [\(84\) Single Number III](#)

Given $2^*n + 2$ numbers, every numbers occurs twice except two, find them.

Example

Given [1,2,2,3,4,4,5,3] return 1 and 5

Challenge

$O(n)$ time, $O(1)$ extra space.

Single Number follow up, $x_1, x_2 . \quad x_1 \wedge x_2 , \quad x_1 \quad x_2 , \quad x_1 \wedge x_2 \wedge x_1 = x_2$

$x_2 , \quad x_{-1} . \quad x_1 \wedge x_2 \dots$

$x_1 \wedge x_2 \quad x_1 \wedge x_2 \quad x_1 \wedge x_2 \ 0 \quad x_1 \wedge x_2 \ 0 \quad x_1 \wedge x_2 \ 011 \quad x_1 \wedge x_2 \quad x_1 \wedge x_2 \ 011$

Java

```
public class Solution {
    /**
     * @param A : An integer array
     * @return : Two integers
     */
    public List<Integer> singleNumberIII(int[] A) {
        ArrayList<Integer> nums = new ArrayList<Integer>();
        if (A == null || A.length == 0) return nums;

        int x1xorx2 = 0;
        for (int i : A) {
            x1xorx2 ^= i;
        }

        // get the last 1 bit of x1xorx2, e.g. 1010 ==> 0010
        int last1Bit = x1xorx2 - (x1xorx2 & (x1xorx2 - 1));
        int single1 = 0, single2 = 0;
        for (int i : A) {
            if ((last1Bit & i) == 0) {
                single1 ^= i;
            } else {
                single2 ^= i;
            }
        }
    }
}
```

```
        nums.add(single1);
        nums.add(single2);
        return nums;
    }
}
```

1 `x1xorx2 - (x1xorx2 & (x1xorx2 - 1))`, Bit Manipulation last1Bit 01.

$O(n)$, $O(1)$.

Reference

- Single Number III Java/C++/Python

O(1) Check Power of 2

Source

- lintcode: [\(142\) O\(1\) Check Power of 2](#)

```
Using O(1) time to check whether an integer n is a power of 2.
```

Example

For n=4, return true;

For n=5, return false;

Challenge

O(1) time

:(

21 $O(1)$ 210

$x - 1$ $x = 4$

```
0100 ==> 4
0011 ==> 3
```

02

Python

```
class Solution:
    """
    @param n: An integer
    @return: True or False
    """
    def checkPowerOf2(self, n):
        if n < 1:
            return False
        else:
            return (n & (n - 1)) == 0
```

C++

```
class Solution {
public:
    /*
```

```

 * @param n: An integer
 * @return: True or false
 */
bool checkPowerOf2(int n) {
    if (1 > n) {
        return false;
    } else {
        return 0 == (n & (n - 1));
    }
};


```

Java

```

class Solution {
    /*
     * @param n: An integer
     * @return: True or false
     */
    public boolean checkPowerOf2(int n) {
        if (n < 1) {
            return false;
        } else {
            return (n & (n - 1)) == 0;
        }
    }
};


```

00 $(n \& (n - 1))$

$O(1)$.

Convert Integer A to Integer B

Source

- CC150, lintcode: [\(181\) Convert Integer A to Integer B](#)

Determine the number of bits required to convert integer A to integer B

Example

Given n = 31, m = 14, return 2

$(31)_{10} = (11111)_2$

$(14)_{10} = (01110)_2$

01110

O1 Check Power of 2

$x \& (x - 1)$

21 —— 22

$-2(-2) \& (-2 - 1)(8)$

```

11111110 <==> -2      -2 <==> 11111110
+
11111111 <==> -1      -3 <==> 11111101
=
11111101           11111100

```

— $x \& (x - 1)$ 10

C/C++ Java Python Orz... Python 0

Python

```

class Solution:
    """
    @param a, b: Two integer
    return: An integer
    """
    def bitSwapRequired(self, a, b):
        count = 0
        a_xor_b = a ^ b
        neg_flag = False
        if a_xor_b < 0:
            a_xor_b = abs(a_xor_b) - 1
            neg_flag = True
        while a_xor_b > 0:
            count += 1
            a_xor_b = a_xor_b & (a_xor_b - 1)

```

```

    a_xor_b &= (a_xor_b - 1)

    # bit_wise = 32
    if neg_flag:
        count = 32 - count
    return count

```

C++

```

class Solution {
public:
    /**
     *@param a, b: Two integer
     *return: An integer
     */
    int bitSwapRequired(int a, int b) {
        int count = 0;
        int a_xor_b = a ^ b;
        while (a_xor_b != 0) {
            ++count;
            a_xor_b &= (a_xor_b - 1);
        }

        return count;
    }
};

```

Java

```

class Solution {
    /**
     *@param a, b: Two integer
     *return: An integer
     */
    public static int bitSwapRequired(int a, int b) {
        int count = 0;
        int a_xor_b = a ^ b;
        while (a_xor_b != 0) {
            ++count;
            a_xor_b &= (a_xor_b - 1);
        }

        return count;
    }
};

```

Python int long 01

C/C++, Java `a_xor_b != 0 a_xor_b > 0 .`

```
1    0(max(ones in a ^ b)) .
```

Python

Reference

- [BitManipulation - Python Wiki](#)
- [5. Expressions — Python 2.7.10rc0 documentation](#)
- [Python - -](#)

Factorial Trailing Zeros

Source

- leetcode: Factorial Trailing Zeros | LeetCode OJ
 - lintcode: (2) Trailing Zeros

Write an algorithm which computes the number of trailing zeros in n factorial.

Example

$11! = 39916800$, so the out should be 2

Challenge

$O(\log N)$ time

1 - Iterative

10 $2 \times 5, 1 \times 10$ 510255...

wikipedia 522555100100

Python

```
class Solution:
    # @param {integer} n
    # @return {integer}
    def trailingZeroes(self, n):
        if n < 0:
            return -1

        count = 0
        while n > 0:
            n /= 5
            count += n

        return count
```

C++

```
class Solution {
public:
    int trailingZeroes(int n) {
        if (n < 0) {
            return -1;
        }
        int count = 0;
        for (; n > 0; n /= 5) {
```

```

        count += (n / 5);
    }

    return count;
}
};


```

Java

```

public class Solution {
    public int trailingZeroes(int n) {
        if (n < 0) {
            return -1;
        }

        int count = 0;
        for (; n > 0; n /= 5) {
            count += (n / 5);
        }

        return count;
    }
}


```

1. 0-1.
2. $5 \lfloor n/5 \rfloor$
3. $n \text{ } /=\text{ } 5 \quad i \text{ } *=\text{ } 5,$

lintcode leetcode OJ

$n \text{ } /=\text{ } 5 \quad \log_5 n \text{ } \text{count} \quad O(1).$

2 - Recursive

$n \leq 0.$

Python

```

class Solution:
    # @param {integer} n
    # @return {integer}
    def trailingZeroes(self, n):
        if n == 0:
            return 0
        elif n < 0:

```

```

        return -1
    else:
        return n / 5 + self.trailingZeroes(n / 5)

```

C++

```

class Solution {
public:
    int trailingZeroes(int n) {
        if (n == 0) {
            return 0;
        } else if (n < 0) {
            return -1;
        } else {
            return n / 5 + trailingZeroes(n / 5);
        }
    }
};

```

Java

```

public class Solution {
    public int trailingZeroes(int n) {
        if (n == 0) {
            return 0;
        } else if (n < 0) {
            return -1;
        } else {
            return n / 5 + trailingZeroes(n / 5);
        }
    }
}

```

-10. $\log_5 n$,

$\log_5 n$,

Reference

- wikipedia. Prime factor - Wikipedia, the free encyclopedia ↵
- Count trailing zeroes in factorial of a number - GeeksforGeeks

Unique Binary Search Trees

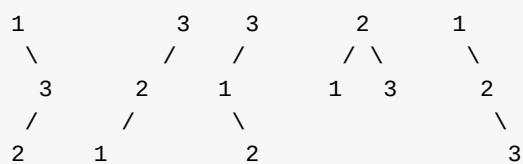
Source

- leetcode: Unique Binary Search Trees | LeetCode OJ
- lintcode: (163) Unique Binary Search Trees

Given n , how many structurally unique BSTs (binary search trees) that store values $1 \dots n$?

Example

Given $n = 3$, there are a total of 5 unique BST's.



1 -

— $i[0, i-1] [i+1, n]$ BST

$\text{count}[n], \text{count}[n] \ i \ \text{count}[1] = 1, 1 \text{count}[2] = 2, 12 \text{count}[3] 123. 1 \text{BST}$
2312133123

$i \text{ BST } \text{BST } \text{BST } i[0, i - 1] \text{ BST } \text{count}[i - 1], [i + 1, n]$

$\text{BST } 1, 2, 3] [4, 5, 6]$

BST **BST**

BST **c**

$\text{count}[i] = \sum_{j=0}^{i-1} (\text{count}[j] \cdot \text{count}[i - j - 1])$

$\text{count}[3] \text{ BST } \text{BST}$

Python

```

class Solution:
    # @param n: An integer
    # @return: An integer
    def numTrees(self, n):
        if n < 0:
            return -1

        count = [0] * (n + 1)
        count[0] = 1
        for i in xrange(1, n + 1):
            for j in xrange(i):
                count[i] += count[j] * count[i - j - 1]

        return count[n]
    
```

C++

```

class Solution {
public:
    /**
     * @param n: An integer
     * @return: An integer
     */
    int numTrees(int n) {
        if (n < 0) {
            return -1;
        }

        vector<int> count(n + 1);
        count[0] = 1;
        for (int i = 1; i != n + 1; ++i) {
            for (int j = 0; j != i; ++j) {
                count[i] += count[j] * count[i - j - 1];
            }
        }

        return count[n];
    }
};

```

Java

```

public class Solution {
    /**
     * @param n: An integer
     * @return: An integer
     */
    public int numTrees(int n) {
        if (n < 0) {
            return -1;
        }

        int[] count = new int[n + 1];
        count[0] = 1;
        for (int i = 1; i < n + 1; ++i) {
            for (int j = 0; j < i; ++j) {
                count[i] += count[j] * count[i - j - 1];
            }
        }

        return count[n];
    }
}

```

1. n 0
2. n + 1 0 count[0] 1.
3. for count[i]
4. count[n]

count[0] 10

$n + 1, O(n + 1)$. for $n^2/2, O(n^2)$. Catalan number $O(n)$

[Wikipedia](#)

Reference

- fisherlei : [LeetCode] Unique Binary Search Trees, Solution ↵
- [Unique Binary Search Trees |](#)
- [Catalan number - Wikipedia, the free encyclopedia](#)

Update Bits

Source

- CTCI: [\(179\) Update Bits](#)

Given two 32-bit numbers, N and M, and two bit positions, i and j. Write a method to set all bits between i and j in N equal to M (e.g., M becomes a substring of N located at i and starting at j)

Example

Given $N=(10000000000)_2$, $M=(10101)_2$, $i=2$, $j=6$

```
return  $N=(10001010100)_2$ 
```

Note

In the function, the numbers N and M will given in decimal, you should also return a decimal number.

Challenge

Minimum number of operations?

Clarification

You can assume that the bits j through i have enough space to fit all of M.

That is, if $M=10011$

you can assume that there are at least 5 bits between j and i.

You would not, for example, have $j=3$ and $i=2$,

because M could not fully fit between bit 3 and bit 2.

Cracking The Coding Interview M N i j

1. i j 01 mask
2. mask N N i j
3. M i M N
4. N | M

mask CTCI (1111...000...111)

C++

```
class Solution {
public:
    /**
     *@param n, m: Two integer
     *@param i, j: Two bit positions
     *return: An integer
```

```

*/
int updateBits(int n, int m, int i, int j) {
    int ones = ~0;
    int left = ones << (j + 1);
    int right = ((1 << i) - 1);
    int mask = left | right;

    return (n & mask) | (m << i);
}
};

```

[-521, 0, 31, 31] WA, bug m = 0, i = 31, j = 31 left left ones , j 31 j +
 1 32 left int

C++

```

class Solution {
public:
    /**
     *@param n, m: Two integer
     *@param i, j: Two bit positions
     *return: An integer
     */
    int updateBits(int n, int m, int i, int j) {
        int ones = ~0;
        int mask = 0;
        if (j < 31) {
            int left = ones << (j + 1);
            int right = ((1 << i) - 1);
            mask = left | right;
        } else {
            mask = (1 << i) - 1;
        }

        return (n & mask) | (m << i);
    }
};

```

$\sim 0 \ 1 \quad j == 31 \quad \text{left } 1 (1 << i) - 1, \ i \ j \ M,$

$O(1)$.

C++

```

class Solution {
public:
    /**
     *@param n, m: Two integer
     *@param i, j: Two bit positions
     *return: An integer
     */
    int updateBits(int n, int m, int i, int j) {
        // get the bit width of input integer
        int bitwidth = 8 * sizeof(n);
        int ones = ~0;
        // use unsigned for logical shift
        unsigned int mask = ones << (bitwidth - (j - i + 1));
        mask = mask >> (bitwidth - 1 - j);

        return (n & (~mask)) | (m << i);
    }
};

```

`if` `mask` C/C++ `mask .`
`int 32` `sizeof`

$O(1)$.

Reference

- [c++ - logical shift right on signed data - Stack Overflow](#)
- [Update Bits |](#)
- *CTCI 5th Chapter 9.5 p163*

Fast Power

Source

- lintcode: [\(140\) Fast Power](#)

```
(a * b) % p = ((a % p) * (b % p)) % p
```

$a \ b \ p \ a, b \ p \ p$

$a^n \dots \quad a^n$

$$a^n = a^{n/2} \cdot a^{n/2} = a^{n/4} \cdot a^{n/4} \cdot a^{n/4} \cdot a^{n/4} \cdots$$

Python

```
class Solution:
    """
    @param a, b, n: 32bit integers
    @return: An integer
    """
    def fastPower(self, a, b, n):
        if n == 1:
            return a % b
        elif n == 0:
            # do not use `1` instead `1 % b` because `b = 1`
            return 1 % b
        elif n < 0:
            return -1

        # (a * b) % p = ((a % p) * (b % p)) % p
        product = self.fastPower(a, b, n / 2)
        product = (product * product) % b
        if n % 2 == 1:
            product = (product * a) % b

        return product
```

C++

```

class Solution {
public:
    /*
     * @param a, b, n: 32bit integers
     * @return: An integer
     */
    int fastPower(int a, int b, int n) {
        if (1 == n) {
            return a % b;
        } else if (0 == n) {
            // do not use 1 instead (1 % b)! b = 1
            return 1 % b;
        } else if (0 > n) {
            return -1;
        }

        // (a * b) % p = ((a % p) * (b % p)) % p
        // use long long to prevent overflow
        long long product = fastPower(a, b, n / 2);
        product = (product * product) % b;
        if (1 == n % 2) {
            product = (product * a) % b;
        }

        // cast long long to int
        return (int) product;
    }
};

```

Java

```

class Solution {
    /*
     * @param a, b, n: 32bit integers
     * @return: An integer
     */
    public int fastPower(int a, int b, int n) {
        if (n == 1) {
            return a % b;
        } else if (n == 0) {
            return 1 % b;
        } else if (n < 0) {
            return -1;
        }

        // (a * b) % p = ((a % p) * (b % p)) % p
        // use long to prevent overflow
        long product = fastPower(a, b, n / 2);
        product = (product * product) % b;
        if (n % 2 == 1) {
            product = (product * a) % b;
        }

        // cast long to int
        return (int) product;
    }
}

```

```
};
```

```
n      n == 0  a0 11  b == 1 0  1 % b .
```

```
n a,          long    int
```

product $O(1)$, $\log n$, $O(\log n)$, $O(\log n)$.

Reference

- [Lintcode: Fast Power - Yu's Garden -](#)
- [Fast Power Java/C++/Python](#)

Count 1 in Binary

Source

- lintcode: [\(365\) Count 1 in Binary](#)

Count how many 1 in binary representation of a 32-bit integer.

Example

Given 32, return 1

Given 5, return 2

Given 1023, return 9

Challenge

If the integer is n bits with m 1 bits. Can you do it in $O(m)$ time?

[O1 Check Power of 2](#) $x \& (x - 1) \neq x$

Java

```
public class Solution {
    /**
     * @param num: an integer
     * @return: an integer, the number of ones in num
     */
    public int countOnes(int num) {
        int count = 0;
        while (num != 0) {
            num = num & (num - 1);
            count++;
        }
        return count;
    }
}
```

1 $O(m)$. $O(1)$.

Reference

- [Number of 1 bits | LeetCode](#) -

Fibonacci

Source

- lintcode: ([366](#)) Fibonacci

Find the Nth number in Fibonacci sequence.

A Fibonacci sequence is defined as follow:

The first two numbers are 0 and 1.

The i th number is the sum of $i-1$ th number and $i-2$ th number.

The first ten numbers in Fibonacci sequence is:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34 ...

Example

Given 1, return 0

Given 2, return 1

Given 10, return 34

Note

The Nth fibonacci number won't exceed the max value of signed 32-bit integer in the test cases.

Java

```
class Solution {
    /**
     * @param n: an integer
     * @return an integer f(n)
     */
    public int fibonacci(int n) {
        if (n < 0) return -1;
        if (n == 1) return 0;
        if (n == 2) return 1;

        int fn = 0, fn1 = 1, fn2 = 1;
        for (int i = 3; i <= n; i++) {
            fn = fn1 + fn2;
            fn2 = fn1;
            fn1 = fn;
        }
    }
}
```

```
        return fn;  
    }  
}
```

1. corner cases
2. fn, fn1, fn2,
3. fn, fn2, fn1

$O(n)$, $O(1)$.

A plus B Problem

Source

- lintcode: [\(1\) A + B Problem](#)

Write a function that add two numbers A and B.
You should not use + or any arithmetic operators.

Example

Given a=1 and b=2 return 3

Note

There is no need to read data from standard input stream.
Both parameters are given in function aplusb,
you job is to calculate the sum and return it.

Challenge

Of course you can just return a + b to get accepted.
But Can you challenge not do it like that?

Clarification

Are a and b both 32-bit integers?

Yes.

Can I use bit operation?

Sure you can.

Java

```
class Solution {
    /*
     * param a: The first integer
     * param b: The second integer
     * return: The sum of a and b
     */
    public int aplusb(int a, int b) {
        int result = a ^ b;
        int carry = a & b;
        carry <= 1;
        if (carry != 0) {
            result = aplusb(result, carry);
        }

        return result;
    }
}
```

00

$O(1)$. $O(1)$.

Print Numbers by Recursion

Source

- lintcode: [\(371\) Print Numbers by Recursion](#)

Print numbers from 1 to the largest number with N digits by recursion.

Example

Given N = 1, return [1,2,3,4,5,6,7,8,9].

Given N = 2, return [1,2,3,4,5,6,7,8,9,10,11,12,...,99].

Note

It's pretty easy to do recursion like:

```
recursion(i) {
    if i > largest number:
        return
    results.add(i)
    recursion(i + 1)
}
```

however this cost a lot of recursion memory as the recursion depth maybe very large.

Can you do it in another way to recursive with at most N depth?

Challenge

Do it in recursion, not for-loop.

N `recursion(i)` , N 0~9() N
N=110 N,

`DFS` N=2
`DFS`

Java

```
public class Solution {
    /**
     * @param n: An integer.
     * @return : An array storing 1 to the largest number with n digits.
     */
    public List<Integer> numbersByRecursion(int n) {
        List<Integer> result = new ArrayList<Integer>();
        if (n <= 0) {
            return result;
        }
        helper(n, result);
        return result;
    }

    private void helper(int n, List<Integer> ret) {
```

```

if (n == 0) return;
helper(n - 1, ret);
// current base such as 10, 20, 30...
int base = (int)Math.pow(10, n - 1);
// get List size before for loop
int size = ret.size();
for (int i = 1; i < 10; i++) {
    // add 10, 100, 1000...
    ret.add(i * base);
    for (int j = 0; j < size; j++) {
        // add 11, 12, 13...
        ret.add(ret.get(j) + base * i);
    }
}
}
}
}

```

`n == 0 , N-1 base 10 size for`

$10^n \quad O(10^n), \quad O(1)$.

Reference

- [Lintcode: Print Numbers by Recursion | codesolutiony](#)

Majority Number

Source

- leetcode: Majority Element | LeetCode OJ
- lintcode: (46) Majority Number

Given an array of integers, the majority number is the number that occurs more than half of the size of the array. Find it.

Example

Given [1, 1, 1, 1, 2, 2, 2], return 1

Challenge

$O(n)$ time and $O(1)$ extra space

PK

10

pair<int, int> .

Java

```
public class Solution {
    /**
     * @param nums: a list of integers
     * @return: find a majority number
     */
    public int majorityNumber(ArrayList<Integer> nums) {
        if (nums == null || nums.isEmpty()) return -1;

        // pair<key, count>
        int key = -1, count = 0;
        for (int num : nums) {
            // re-initialize
            if (count == 0) {
                key = num;
                count = 1;
                continue;
            }
            // increment/decrement count
            if (key == num) {
                count++;
            } else {
                count--;
            }
        }

        return key;
    }
}
```

```
count = 0 , count == 0
```

$O(n)$, $O(1)$.

Majority Number II

Source

- leetcode: Majority Element II | LeetCode OJ
- lintcode: (47) Majority Number II

Given an array of integers,
the majority number is the number that occurs more than $1/3$ of the size of the array.

Find it.

Example

Given [1, 2, 1, 2, 1, 3, 3], return 1.

Note

There is only one majority number in the array.

Challenge

$O(n)$ time and $O(1)$ extra space.

Majority Number

Java

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @return: The majority number that occurs more than 1/3
     */
    public int majorityNumber(ArrayList<Integer> nums) {
        if (nums == null || nums.isEmpty()) return -1;

        // pair
        int key1 = -1, key2 = -1;
        int count1 = 0, count2 = 0;
        for (int num : nums) {
            if (count1 == 0) {
                key1 = num;
                count1 = 1;
                continue;
            } else if (count2 == 0 && key1 != num) {
                key2 = num;
                count2 = 1;
                continue;
            }
            if (key1 == num) {
                count1++;
            }
            if (key2 == num) {
                count2++;
            }
        }
        return count1 >= count2 ? key1 : key2;
    }
}
```

```
        } else if (key2 == num) {
            count2++;
        } else {
            count1--;
            count2--;
        }
    }

count1 = 0;
count2 = 0;
for (int num : nums) {
    if (key1 == num) {
        count1++;
    } else if (key2 == num) {
        count2++;
    }
}
return count1 > count2 ? key1 : key2;
}
}
```

```
count == 0    count2 == 0 && key1 = num , count1 count2
```

$O(n)$, $O(2 \times 2) = O(1)$.

Reference

- [Majority Number II Java/C++/Python](#)

Majority Number III

Source

- lintcode: [\(48\) Majority Number III](#)

Given an array of integers and a number k ,
the majority number is the number that occurs more than $1/k$ of the size of the array.

Find it.

Example

Given [3,1,2,3,2,3,3,4,4,4] and $k=3$, return 3.

Note

There is only one majority number in the array.

Challenge

$O(n)$ time and $O(k)$ extra space

[Majority Number II](#) K-1 key1, key2... K K-11. size

$\approx K$ key.

Java

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @param k: As described
     * @return: The majority number
     */
    public int majorityNumber(ArrayList<Integer> nums, int k) {
        HashMap<Integer, Integer> hash = new HashMap<Integer, Integer>();
        if (nums == null || nums.isEmpty()) return -1;

        // update HashMap
        for (int num : nums) {
            if (!hash.containsKey(num)) {
                hash.put(num, 1);
                if (hash.size() >= k) {
                    removeZeroCount(hash);
                }
            } else {
                hash.put(num, hash.get(num) + 1);
            }
        }

        // reset
    }

    private void removeZeroCount(HashMap<Integer, Integer> hash) {
        for (Map.Entry<Integer, Integer> entry : hash.entrySet()) {
            if (entry.getValue() == 0) {
                hash.remove(entry.getKey());
            }
        }
    }
}
```

```

        for (int key : hash.keySet()) {
            hash.put(key, 0);
        }
        for (int key : nums) {
            if (hash.containsKey(key)) {
                hash.put(key, hash.get(key) + 1);
            }
        }

        // find max
        int maxKey = -1, maxCount = 0;
        for (int key : hash.keySet()) {
            if (hash.get(key) > maxCount) {
                maxKey = key;
                maxCount = hash.get(key);
            }
        }

        return maxKey;
    }

    private void removeZeroCount(HashMap<Integer, Integer> hash) {
        Set<Integer> keySet = hash.keySet();
        for (int key : keySet) {
            hash.put(key, hash.get(key) - 1);
        }

        /* solution 1 */
        Iterator<Map.Entry<Integer, Integer>> it = hash.entrySet().iterator();
        while (it.hasNext()) {
            Map.Entry<Integer, Integer> entry = it.next();
            if(entry.getValue() == 0) {
                it.remove();
            }
        }

        /* solution 2 */
        // List<Integer> removeList = new ArrayList<>();
        // for (int key : keySet) {
        //     hash.put(key, hash.get(key) - 1);
        //     if (hash.get(key) == 0) {
        //         removeList.add(key);
        //     }
        // }
        // for (Integer key : removeList) {
        //     hash.remove(key);
        // }

        /* solution3 lambda expression for Java8 */
    }
}

```

Java

$O(n)$, $O(k)$.

Reference

- [Majority Number III Java/C++/Python](#)

Digit Counts

Source

- leetcode: Number of Digit One | LeetCode OJ
- lintcode: (3) Digit Counts

Count the number of k's between 0 and n. k can be 0 - 9.

Example

if n=12, k=1 in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
we have FIVE 1's (1, 10, 11, 12)

leetcode Lintcode 0 n k i k

Java

```
class Solution {
    /*
     * param k : As description.
     * param n : As description.
     * return: An integer denote the count of digit k in 1..n
     */
    public int digitCounts(int k, int n) {
        int count = 0;
        char kChar = (char)(k + '0');
        for (int i = k; i <= n; i++) {
            char[] iChars = Integer.toString(i).toCharArray();
            for (char iChar : iChars) {
                if (kChar == iChar) count++;
            }
        }
        return count;
    }
}
```

$$O(n \times L), \text{L } n \quad O(L).$$

Ugly Number

Source

- leetcode: [Ugly Number | LeetCode OJ](#)
- lintcode: [\(4\) Ugly Number](#)

Ugly number is a number that only have factors 3, 5 and 7.

Design an algorithm to find the Kth ugly number.

The first 5 ugly numbers are 3, 5, 7, 9, 15 ...

Example

If K=4, return 9.

Challenge

$O(K \log K)$ or $O(K)$ time.

1 - TLE

Lintcode Leetcode Lintcode K 357 K

Java

```
class Solution {
    /**
     * @param k: The number k.
     * @return: The kth prime number as description.
     */
    public long kthPrimeNumber(int k) {
        if (k <= 0) return -1;

        int count = 0;
        long num = 1;
        while (count < k) {
            num++;
            if (isUgly(num)) {
                count++;
            }
        }
        return num;
    }

    private boolean isUgly(long num) {
        while (num % 3 == 0) {
            num /= 3;
        }
        while (num % 5 == 0) {
            num /= 5;
        }
    }
}
```

```

        while (num % 7 == 0) {
            num /= 7;
        }

        return num == 1;
    }
}

```

3573,5,71 num

$$O(n^3)$$

2 -

3, 5, 7, 3, 5, 7

$$U_k = 3^{x_3} \cdot 5^{x_5} \cdot 7^{x_7}, \quad U_{k+1} = U_k \cdot x_3, x_5, x_7 \quad U_{k+1} = U_k \cdot \frac{3^{y_3} \cdot 5^{y_5} \cdot 7^{y_7}}{3^{z_3} \cdot 5^{z_5} \cdot 7^{z_7}}, \quad O(K)$$

$$y, z : ($$

3, 5, 7 N

Java

```

class Solution {
    /**
     * @param k: The number k.
     * @return: The kth prime number as description.
     */
    public long kthPrimeNumber(int k) {
        if (k <= 0) return -1;

        ArrayList<Long> nums = new ArrayList<Long>();
        nums.add(1L);
        for (int i = 0; i < k; i++) {
            long minNextUgly = Math.min(nextUgly(nums, 3), nextUgly(nums, 5));
            minNextUgly = Math.min(minNextUgly, nextUgly(nums, 7));
            nums.add(minNextUgly);
        }

        return nums.get(nums.size() - 1);
    }

    private long nextUgly(ArrayList<Long> nums, int factor) {
        int size = nums.size();
        int start = 0, end = size - 1;
        while (start + 1 < end) {
            int mid = start + (end - start) / 2;
        }
    }
}

```

```

        if (nums.get(mid) * factor > nums.get(size - 1)) {
            end = mid;
        } else {
            start = mid;
        }
    }
    if (nums.get(start) * factor > nums.get(size - 1)) {
        return nums.get(start) * factor;
    }

    return nums.get(end) * factor;
}
}

```

nextUgly factor nums.add(1L) 1 L Long,

$O(\log K)$, K $O(K \log K)$, $O(K)$.

3 -

TBD

Reference

- Offer
- [Ugly Numbers - GeeksforGeeks](#)

Linked List -

Remove Duplicates from Sorted List

Source

- leetcode: Remove Duplicates from Sorted List | LeetCode OJ
- lintcode: (112) Remove Duplicates from Sorted List

Given a sorted linked list,
delete all duplicates such that each element appear only once.

Example

Given 1->1->2, return 1->2.
Given 1->1->2->3->3, return 1->2->3.

next next ,

Python

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    # @param {ListNode} head
    # @return {ListNode}
    def deleteDuplicates(self, head):
        if head is None:
            return None

        node = head
        while node.next is not None:
            if node.val == node.next.val:
                node.next = node.next.next
            else:
                node = node.next

        return head
```

C++

```
/*
 * Definition of ListNode
 * class ListNode {

```

```

* public:
*     int val;
*     ListNode *next;
*     ListNode(int val) {
*         this->val = val;
*         this->next = NULL;
*     }
* }
*/
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: head node
     */
    ListNode *deleteDuplicates(ListNode *head) {
        if (head == NULL) {
            return NULL;
        }

        ListNode *node = head;
        while (node->next != NULL) {
            if (node->val == node->next->val) {
                ListNode *temp = node->next;
                node->next = node->next->next;
                delete temp;
            } else {
                node = node->next;
            }
        }

        return head;
    }
};

```

Java

```

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
*/
public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null) return null;

        ListNode node = head;
        while (node.next != null) {
            if (node.val == node.next.val) {
                node.next = node.next.next;
            } else {
                node = node.next;
            }
        }
    }
}

```

```
        return head;
    }
}
```

1. headNULL
2. node->val == node->next->val node->next (C/C++)
3. else

```
while node != null && node->next != null , head
```

$O(n)$, $O(1)$.

Reference

- Remove Duplicates from Sorted List |

Remove Duplicates from Sorted List II

Source

- leetcode: Remove Duplicates from Sorted List II | LeetCode OJ
- lintcode: (113) Remove Duplicates from Sorted List II

Given a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list.

Example

Given 1->2->3->3->4->4->5, return 1->2->5.

Given 1->1->1->2->3, return 2->3.

()if **—dummy node.**

```
ListNode *dummy = new ListNode(0);
dummy->next = head;
ListNode *node = dummy;
```

dummy nextheaddummyhead A->B->C BACAnextCheadnodenode

```
dummy  node->val == node->next->val
1.                   next
2.  val node->next  node->next->next
```

C++ - Wrong

```
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution{
public:
```

```

/**
 * @param head: The first node of linked list.
 * @return: head node
 */
ListNode * deleteDuplicates(ListNode *head) {
    if (head == NULL || head->next == NULL) {
        return NULL;
    }

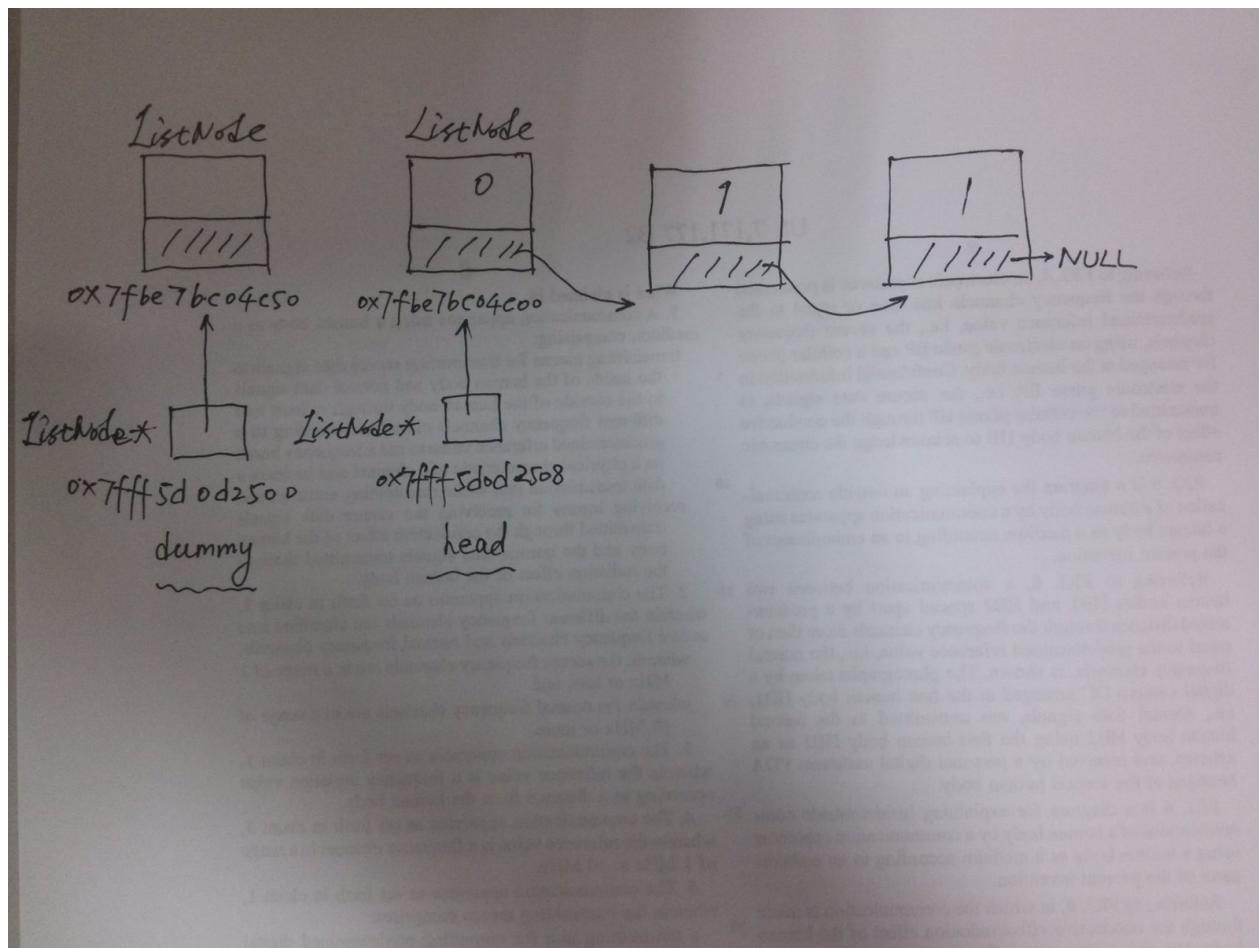
    ListNode *dummy;
    dummy->next = head;
    ListNode *node = dummy;

    while (node->next != NULL && node->next->next != NULL) {
        if (node->next->val == node->next->next->val) {
            int val = node->next->val;
            while (node->next != NULL && val == node->next->val) {
                ListNode *temp = node->next;
                node->next = node->next->next;
                delete temp;
            }
        } else {
            node->next = node->next->next;
        }
    }

    return dummy->next;
}
};

```

1. dummy new
2. else node->next = node->next->next; dummy->next dummy->next node = node->next; node->next



`ListNode` `dummy` `0x7fff5d0d25000` `x7fbe7bc04c50`. `head` `0x7fff5d0d25080` `0x7fbe7bc04c00`.

Python

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    # @param {ListNode} head
    # @return {ListNode}
    def deleteDuplicates(self, head):
        if head is None:
            return None

        dummy = ListNode(0)
        dummy.next = head
        node = dummy
        while node.next is not None and node.next.next is not None:
            if node.next.val == node.next.next.val:
                val_prev = node.next.val
                while node.next is not None and node.next.val == val_prev:
                    node.next = node.next.next
```

```

    else:
        node = node.next

    return dummy.next

```

C++

```

/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (head == NULL) return NULL;

        ListNode *dummy = new ListNode(0);
        dummy->next = head;
        ListNode *node = dummy;
        while (node->next != NULL && node->next->next != NULL) {
            if (node->next->val == node->next->next->val) {
                int val_prev = node->next->val;
                // remove ListNode node->next
                while (node->next != NULL && val_prev == node->next->val) {
                    ListNode *temp = node->next;
                    node->next = node->next->next;
                    delete temp;
                }
            } else {
                node = node->next;
            }
        }

        return dummy->next;
    }
};

```

Java

```

/*
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null) return null;

```

```

ListNode dummy = new ListNode(0);
dummy.next = head;
ListNode node = dummy;
while(node.next != null && node.next.next != null) {
    if (node.next.val == node.next.next.val) {
        int val_prev = node.next.val;
        while (node.next != null && node.next.val == val_prev) {
            node.next = node.next.next;
        }
    } else {
        node = node.next;
    }
}
return dummy.next;
}
}

```

1. head NULL NULL
2. newdummy dummy->next
3. nodedummy
4. valwhile
5. dummy->next

Python is not None

(node.next node.next.next) $O(2n)$. dummy $O(1)$.

Reference

- [Remove Duplicates from Sorted List II |](#)

Remove Duplicates from Unsorted List

Source

- Remove duplicates from an unsorted linked list - GeeksforGeeks

Write a `removeDuplicates()` function which takes a list and deletes any duplicate nodes from the list. The list is not sorted.

For example if the linked list is `12->11->12->21->41->43->21`, then `removeDuplicates()` should convert the list to `12->11->21->41->43`.

If temporary buffer is not allowed, how to solve it?

1 -

Remove Duplicates

CTCI 2.1

Python

```
"""
Definition of ListNode
class ListNode(object):
    def __init__(self, val, next=None):
        self.val = val
        self.next = next
"""

class Solution:
    """
    @param head: A ListNode
    @return: A ListNode
    """
    def deleteDuplicates(self, head):
        if head is None:
            return None

        curr = head
        while curr is not None:
            inner = curr
            while inner.next is not None:
                if inner.next.val == curr.val:
                    inner.next = inner.next.next
                else:
                    inner = inner.next
            curr = curr.next

        return head
```

C++

```


/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: head node
     */
    ListNode *deleteDuplicates(ListNode *head) {
        if (head == NULL) return NULL;

        ListNode *curr = head;
        while (curr != NULL) {
            ListNode *inner = curr;
            while (inner->next != NULL) {
                if (inner->next->val == curr->val) {
                    inner->next = inner->next->next;
                } else {
                    inner = inner->next;
                }
            }
            curr = curr->next;
        }

        return head;
    }
};


```

Java

```


/**
 * Definition for ListNode
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    /**


```

```

    * @param ListNode head is the head of the linked list
    * @return: ListNode head of linked list
    */
public static ListNode deleteDuplicates(ListNode head) {
    if (head == null) return null;

    ListNode curr = head;
    while (curr != null) {
        ListNode inner = curr;
        while (inner.next != null) {
            if (inner.next.val == curr.val) {
                inner.next = inner.next.next;
            } else {
                inner = inner.next;
            }
        }
        curr = curr.next;
    }

    return head;
}
}

```

```
node.next     inner = inner.next  inner.next = inner.next.next
```

$O(\frac{1}{2}n^2)$, $O(1)$.

2 - hashtable

Python

```

"""
Definition of ListNode
class ListNode(object):
    def __init__(self, val, next=None):
        self.val = val
        self.next = next
"""

class Solution:
    """
    @param head: A ListNode
    @return: A ListNode
    """

    def deleteDuplicates(self, head):
        if head is None:
            return None

```

```

hash = {}
hash[head.val] = True
curr = head
while curr.next is not None:
    if hash.has_key(curr.next.val):
        curr.next = curr.next.next
    else:
        hash[curr.next.val] = True
        curr = curr.next

return head

```

C++

```

/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: head node
     */
    ListNode *deleteDuplicates(ListNode *head) {
        if (head == NULL) return NULL;

        // C++ 11 use unordered_map
        // unordered_map<int, bool> hash;
        map<int, bool> hash;
        hash[head->val] = true;
        ListNode *curr = head;
        while (curr->next != NULL) {
            if (hash.find(curr->next->val) != hash.end()) {
                ListNode *temp = curr->next;
                curr->next = curr->next->next;
                delete temp;
            } else {
                hash[curr->next->val] = true;
                curr = curr->next;
            }
        }

        return head;
    }
};

```

Java

```

/**
 * Definition for ListNode
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param ListNode head is the head of the linked list
     * @return: ListNode head of linked list
     */
    public static ListNode deleteDuplicates(ListNode head) {
        if (head == null) return null;

        ListNode curr = head;
        HashMap<Integer, Boolean> hash = new HashMap<Integer, Boolean>();
        hash.put(curr.val, true);
        while (curr.next != null) {
            if (hash.containsKey(curr.next.val)) {
                curr.next = curr.next.next;
            } else {
                hash.put(curr.next.val, true);
                curr = curr.next;
            }
        }

        return head;
    }
}

```

`while`

$$O(n), \quad O(n).$$

Reference

- Remove duplicates from an unsorted linked list - GeeksforGeeks
- ctcI/Question.java at master · gaylemcd/ctci

Partition List

Source

- leetcode: Partition List | LeetCode OJ
- lintcode: ([96](#)) Partition List

Given a linked list and a value x , partition it such that all nodes less than x come before nodes greater than or equal to x .

You should preserve the original relative order of the nodes in each of the two partitions.

For example,
Given $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 2 \rightarrow \text{null}$ and $x = 3$,
return $1 \rightarrow 2 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 5 \rightarrow \text{null}$.

CTCI 2.4xxxx

xxdummy

Python

```
"""
Definition of ListNode
class ListNode(object):

    def __init__(self, val, next=None):
        self.val = val
        self.next = next
"""

class Solution:
    """
    @param head: The first node of linked list.
    @param x: an integer
    @return: a ListNode
    """
    def partition(self, head, x):
        if head is None:
            return None

        leftDummy = ListNode(0)
        left = leftDummy
        rightDummy = ListNode(0)
        right = rightDummy
        node = head
        while node is not None:
            if node.val < x:
```

```

        left.next = node
        left = left.next
    else:
        right.next = node
        right = right.next
    node = node.next
    # post-processing
    right.next = None
    left.next = rightDummy.next

    return leftDummy.next

```

C++

```

/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        if (head == NULL) return NULL;

        ListNode *leftDummy = new ListNode(0);
        ListNode *left = leftDummy;
        ListNode *rightDummy = new ListNode(0);
        ListNode *right = rightDummy;
        ListNode *node = head;
        while (node != NULL) {
            if (node->val < x) {
                left->next = node;
                left = left->next;
            } else {
                right->next = node;
                right = right->next;
            }
            node = node->next;
        }
        // post-processing
        right->next = NULL;
        left->next = rightDummy->next;

        return leftDummy->next;
    }
};

```

Java

```

/*
 * Definition for singly-linked list.
 */

```

```

* public class ListNode {
*     int val;
*     ListNode next;
*     ListNode(int x) { val = x; }
* }
*/
public class Solution {
    public ListNode partition(ListNode head, int x) {
        if (head == null) return null;

        ListNode leftDummy = new ListNode(0);
        ListNode left = leftDummy;
        ListNode rightDummy = new ListNode(0);
        ListNode right = rightDummy;
        ListNode node = head;
        while (node != null) {
            if (node.val < x) {
                left.next = node;
                left = left.next;
            } else {
                right.next = node;
                right = right.next;
            }
            node = node.next;
        }
        // post-processing
        right.next = null;
        left.next = rightDummy.next;

        return leftDummy.next;
    }
}

```

- 1.
2. **dummyleftright**
- 3.
4. **right->next next**

$$O(n), \text{ dummy} \quad O(1).$$

Two Lists Sum

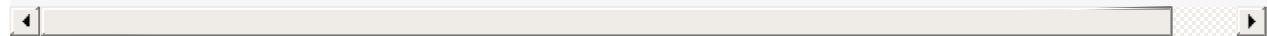
Source

- CC150 - [\(167\) Two Lists Sum](#)

You have two numbers represented by a linked list, where each node contains a single digit. The digits are stored in reverse order, such that the 1's digit is at the head of the list. Write a function that adds the two numbers and returns the sum as a linked list.

Example

Given two lists, 3->1->5->null and 5->9->2->null, return 8->0->8->null



C++ - Iteration

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    /**
     * @param l1: the first list
     * @param l2: the second list
     * @return: the sum list of l1 and l2
     */
    ListNode *addLists(ListNode *l1, ListNode *l2) {
        if (NULL == l1 && NULL == l2) {
            return NULL;
        }

        ListNode *sumlist = new ListNode(0);
        ListNode *templist = sumlist;

        int carry = 0;
        while ((NULL != l1) || (NULL != l2) || (0 != carry)) {
            // padding for NULL
            int l1_val = (NULL == l1) ? 0 : l1->val;
            int l2_val = (NULL == l2) ? 0 : l2->val;
            int sum_val = l1_val + l2_val + carry;
            carry = sum_val / 10;
            sumlist->val = sum_val % 10;
            templist = sumlist;
            if (l1 != NULL) l1 = l1->next;
            if (l2 != NULL) l2 = l2->next;
            sumlist = sumlist->next;
        }
    }
}
```

```

templist->val = (carry + l1_val + l2_val) % 10;
carry = (carry + l1_val + l2_val) / 10;

if (NULL != l1) l1 = l1->next;
if (NULL != l2) l2 = l2->next;

// return sumlist before generating new ListNode
if ((NULL == l1) && (NULL == l2) && (0 == carry)) {
    return sumlist;
}
templist->next = new ListNode(0);
templist = templist->next;
}

return sumlist;
}
};


```

1. (NULL != l1) || (NULL != l2) || (0 != carry) ,
2. - int l1_val = (NULL == l1) ? 0 : l1->val;
3. - (NULL == l1) && (NULL == l2) && (0 == carry) , 0

$O(\max(L1, L2))$.

C++ - Recursion

To-be done.

Reference

- CC150 Chapter 9.2 2.5 p123
- [Add two numbers represented by linked lists | Set 1 - GeeksforGeeks](#)

Two Lists Sum Advanced

Source

- CC150 - [Add two numbers represented by linked lists | Set 2 - GeeksforGeeks](#)

Given two numbers represented by two linked lists, write a function that returns sum list
The sum list is linked list representation of addition of two input numbers.

Example

Input:

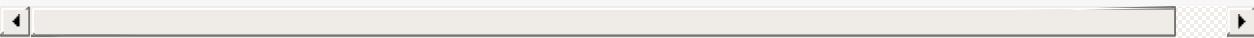
```
First List: 5->6->3 // represents number 563
Second List: 8->4->2 // represents number 842
```

Output

```
Resultant list: 1->4->0->5 // represents number 1405
```

Challenge

Not allowed to modify the lists.
Not allowed to use explicit extra space.



1 -

[Two Lists Sum | Data Structure and Algorithm](#) Two Lists Sum

Reference

- [Add two numbers represented by linked lists | Set 2 - GeeksforGeeks](#)

Remove Nth Node From End of List

Source

- lintcode: [\(174\) Remove Nth Node From End of List](#)

Given a linked list, remove the nth node from the end of list and return its head.

Note

The minimum number of nodes in list is n.

Example

Given linked list: 1->2->3->4->5->null, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5->null.

Challenge

$O(n)$ time

n node->next

C++

```
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @param n: An integer.
     * @return: The head of linked list.
     */
    ListNode *removeNthFromEnd(ListNode *head, int n) {
        if (NULL == head || n < 0) {
            return NULL;
        }

        ListNode *preN = head;
```

```

ListNode *tail = head;
// slow fast pointer
int index = 0;
while (index < n) {
    if (NULL == tail) {
        return NULL;
    }
    tail = tail->next;
    ++index;
}

if (NULL == tail) {
    return head->next;
}

while (tail->next) {
    tail = tail->next;
    preN = preN->next;
}
preN->next = preN->next->next;

return head;
}
};

dummy

```

C++ dummy node

```

/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @param n: An integer.
     * @return: The head of linked list.
     */
    ListNode *removeNthFromEnd(ListNode *head, int n) {
        if (NULL == head || n < 1) {
            return NULL;
        }

        ListNode *dummy = new ListNode(0);
        dummy->next = head;
        ListNode *preDel = dummy;

```

```
for (int i = 0; i != n; ++i) {
    if (NULL == head) {
        return NULL;
    }
    head = head->next;
}

while (head) {
    head = head->next;
    preDel = preDel->next;
}
preDel->next = preDel->next->next;

return dummy->next;
}
};
```

```
dummy head preDel
```

Linked List Cycle

Source

- leetcode: [Linked List Cycle | LeetCode OJ](#)
- lintcode: [\(102\) Linked List Cycle](#)

Given a linked list, determine if it has a cycle in it.

Example

Given -21->10->4->5, tail connects to node index 1, return true

Challenge

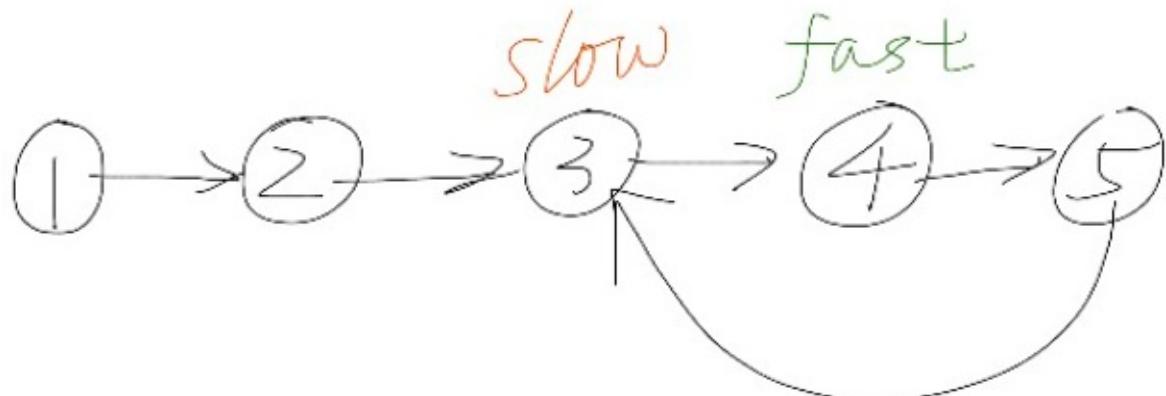
Follow up:

Can you solve it without using extra space?

-

0 NULL false .

NULL



i k 1

C++

```
/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {

```

```

*         this->val = val;
*         this->next = NULL;
*     }
* }
*/
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: True if it has a cycle, or false
     */
    bool hasCycle(ListNode *head) {
        if (NULL == head || NULL == head->next) {
            return false;
        }

        ListNode *slow = head, *fast = head->next;
        while (NULL != fast && NULL != fast->next) {
            fast = fast->next->next;
            slow = slow->next;
            if (slow == fast) return true;
        }

        return false;
    }
};

```

1. `head->next`
2. `head , head`

1. $O(n/2)$.
2. $O(n)$. n

$O(n)$.

Reference

- [Linked List Cycle |](#)

Linked List Cycle II

Source

- leetcode: [Linked List Cycle II | LeetCode OJ](#)
- lintcode: [\(103\) Linked List Cycle II](#)

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Example

Given -21->10->4->5, tail connects to node index 1return node 10

Challenge

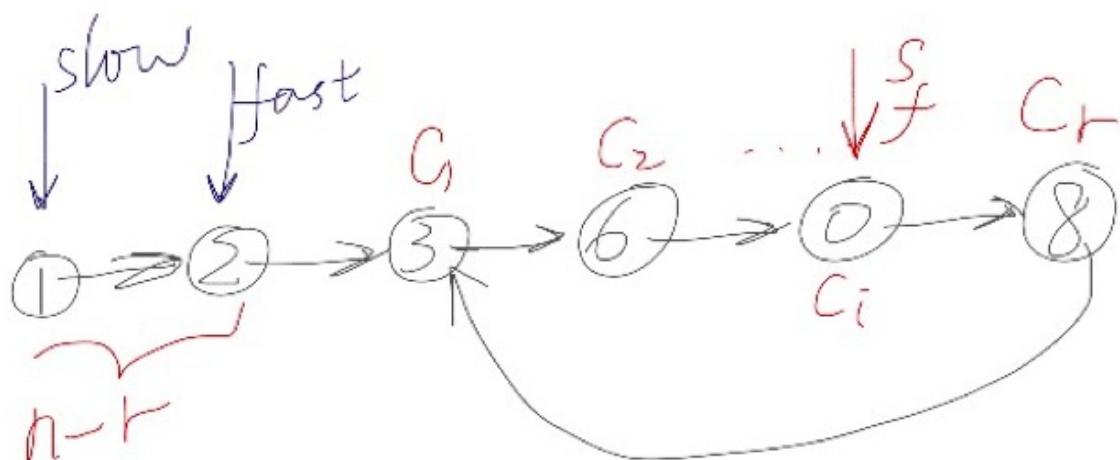
Follow up:

Can you solve it without using extra space?

[Linked List Cycle | Data Structure and Algorithm](#)

$r, n.$

1. ,
- 2.



$$\begin{aligned} \text{slow } \text{fast } 1 & 2 \ 0, \ i \quad C_i, \quad n - r - 1 + i \quad 2 \cdot (n - r - 1 + i) \quad 1: \\ n - r - 1 + i + 1 = l \cdot r. \quad (\quad i \ 1) \ 12 & \end{aligned}$$

$$i \quad 2: (l \cdot r - i + 1) \quad (\quad l) \quad 12 \quad 1 \quad n - r = l \cdot r - i. \quad n - r$$

C++

```

/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: The node where the cycle begins.
     *          if there is no cycle, return null
     */
    ListNode *detectCycle(ListNode *head) {
        if (NULL == head || NULL == head->next) {
            return NULL;
        }

        ListNode *slow = head, *fast = head;
        while (NULL != fast && NULL != fast->next) {
            fast = fast->next->next;
            slow = slow->next;
            if (slow == fast) {
                fast = head;
                while (slow != fast) {
                    fast = fast->next;
                    slow = slow->next;
                }
                return slow;
            }
        }

        return NULL;
    }
};

```

1.

2.

3. **fast**

$O(n)$, $O(n)$. $O(n)$, $O(1)$.

Reference

- [Linked List Cycle II](#) |

Reverse Linked List

Source

- leetcode: [Reverse Linked List | LeetCode OJ](#)
- lintcode: [\(35\) Reverse Linked List](#)

Reverse a linked list.

Example

For linked list 1->2->3, the reversed linked list is 3->2->1

Challenge

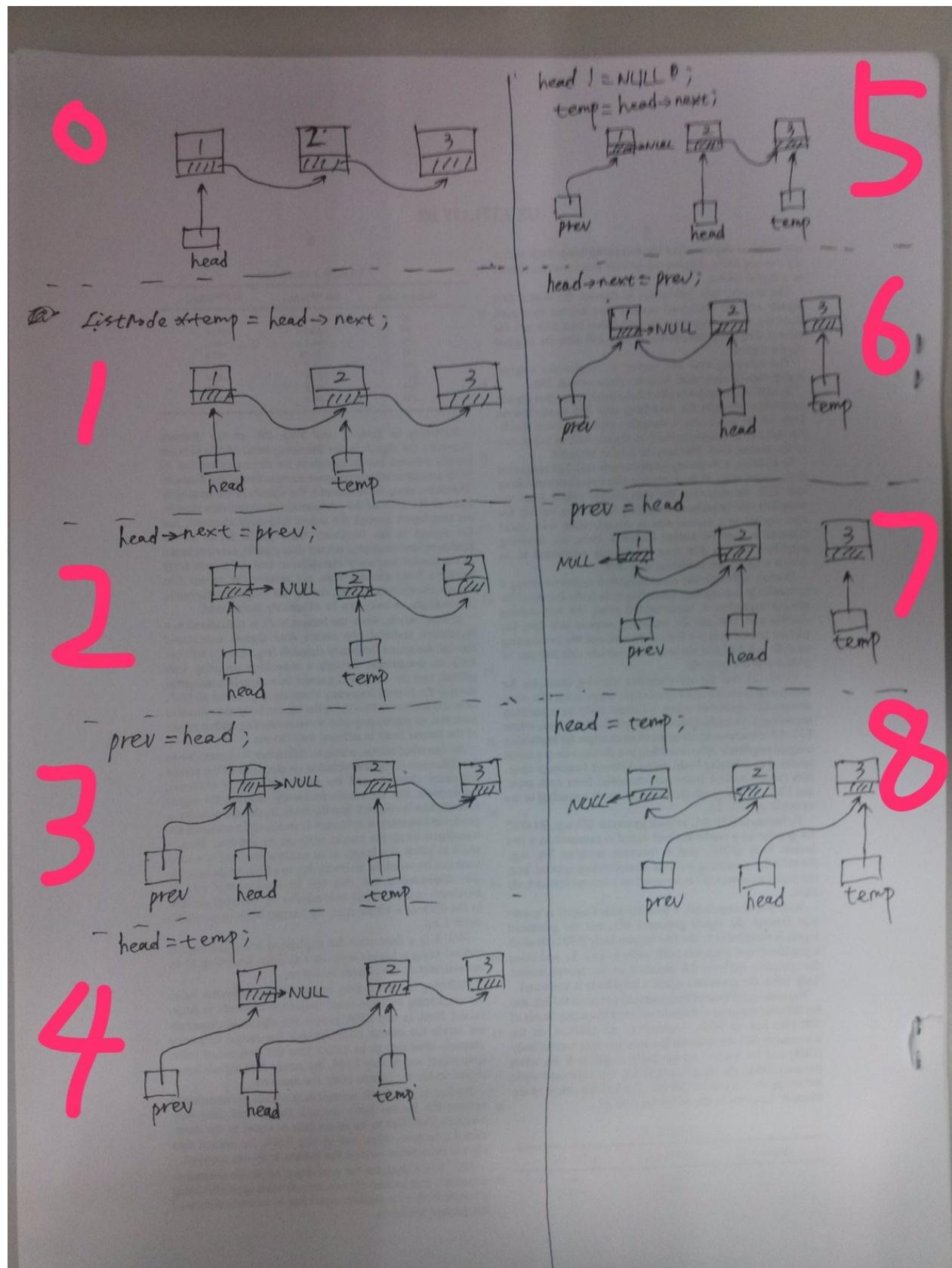
Reverse it in-place and in one-pass

1 -

1->2->3 3->2->1 112 1->2 2->1 12

```
temp = head->next;
head->next = prev;
prev = head;
head = temp;
```

prev head ,



1. head
2. headprev
3. prevhead
4. head

Python

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    # @param {ListNode} head
    # @return {ListNode}
    def reverseList(self, head):
        prev = None
        curr = head
        while curr is not None:
            temp = curr.next
            curr.next = prev
            prev = curr
            curr = temp
        # fix head
        head = prev

        return head
```

C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverse(ListNode* head) {
        ListNode *prev = NULL;
        ListNode *curr = head;
        while (curr != NULL) {
            ListNode *temp = curr->next;
            curr->next = prev;
            prev = curr;
            curr = temp;
        }
        // fix head
        head = prev;

        return head;
    }
};
```

Java

```

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode prev = null;
        ListNode curr = head;
        while (curr != null) {
            ListNode temp = curr.next;
            curr.next = prev;
            prev = curr;
            curr = temp;
        }
        // fix head
        head = prev;

        return head;
    }
}

```

prev

 $O(n)$, $O(1)$.**2 -**

- 1.
- 2.
- 3.

()

Python

```

"""
Definition of ListNode

class ListNode(object):

    def __init__(self, val, next=None):

```

```

        self.val = val
        self.next = next
    """
class Solution:
    """
        @param head: The first node of the linked list.
        @return: You should return the head of the reversed linked list.
                Reverse it in-place.
    """
    def reverse(self, head):
        # case1: empty list
        if head is None:
            return head
        # case2: only one element list
        if head.next is None:
            return head
        # case3: reverse from the rest after head
        newHead = self.reverse(head.next)
        # reverse between head and head->next
        head.next.next = head
        # unlink list from the rest
        head.next = None

        return newHead

```

C++

```

/*
 * Definition of ListNode
 *
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: The new head of reversed linked list.
     */
    ListNode *reverse(ListNode *head) {
        // case1: empty list
        if (head == NULL) return head;
        // case2: only one element list
        if (head->next == NULL) return head;
        // case3: reverse from the rest after head
        ListNode *newHead = reverse(head->next);
        // reverse between head and head->next
        head->next->next = head;
        // unlink list from the rest

```

```

        head->next = NULL;
    }
}
};
```

Java

```

/*
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode reverse(ListNode head) {
        // case1: empty list
        if (head == null) return head;
        // case2: only one element list
        if (head.next == null) return head;
        // case3: reverse from the rest after head
        ListNode newHead = reverse(head.next);
        // reverse between head and head->next
        head.next.next = head;
        // unlink list from the rest
        head.next = null;

        return newHead;
    }
}
```

case1 case2 case3

$O(n)$, $O(n)$, () $O(1)$.

Reference

- - -
- [data structures - Reversing a linked list in Java, recursively - Stack Overflow](#)
- [C++ |](#)
- [iteratively and recursively Java Solution - Leetcode Discuss](#)

Reverse Linked List II

Source

- lintcode: [\(36\) Reverse Linked List II](#)

Reverse a linked list from position m to n .

Note

Given m, n satisfy the following condition: $1 \leq m \leq n \leq \text{length of list}$.

Example

Given $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow \text{NULL}$, $m = 2$ and $n = 4$, return $1 \rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow \text{NULL}$.

Challenge

Reverse it in-place and in one-pass

dummytricky

- $m-1n+1$
- m for $n-m$
- $m-1n+1$
- $\text{dummy} \rightarrow \text{next}$

C++

```
/*
 * Definition of singly-linked-list:
 *
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The head of linked list.
     * @param m: The start position need to reverse.
     * @param n: The end position need to reverse.
     * @return: The new head of partial reversed linked list.
     */
}
```

```

ListNode *reverseBetween(ListNode *head, int m, int n) {
    if (head == NULL || m > n) {
        return NULL;
    }

    ListNode *dummy = new ListNode(0);
    dummy->next = head;
    ListNode *node = dummy;

    for (int i = 1; i != m; ++i) {
        if (node == NULL) {
            return NULL;
        } else {
            node = node->next;
        }
    }

    ListNode *premNode = node;
    ListNode *mNode = node->next;
    ListNode *nNode = mNode, *postnNode = nNode->next;
    for (int i = m; i != n; ++i) {
        if (postnNode == NULL) {
            return NULL;
        }

        ListNode *temp = postnNode->next;
        postnNode->next = nNode;
        nNode = postnNode;
        postnNode = temp;
    }
    premNode->next = nNode;
    mNode->next = postnNode;

    return dummy->next;
}
};

```

- 1.
2. dummy
3. premNode——m
4. nNodepostnNodenpostnNode
5. premNodenNode premNode->next = nNode;
6. mNodepostnNode mNode->next = postnNode;

node **node->next** **node** **node->next**

Merge Two Sorted Lists

Source

- lintcode: ([165](#)) Merge Two Sorted Lists
- leetcode: Merge Two Sorted Lists | LeetCode OJ

Merge two sorted linked lists and return it as a new list.
The new list should be made by splicing together the nodes of the first two lists.

Example

Given `1->3->8->11->15->null`, `2->null` , return `1->2->3->8->11->15->null`

`dummy next next next NULL ...`

`dummy dummy lastNode l1 l2 NULL while lastNode->next`

C++

```
/*
 * Definition for singly-linked list.
 */
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        ListNode *dummy = new ListNode(0);
        ListNode *lastNode = dummy;
        while ((NULL != l1) && (NULL != l2)) {
            if (l1->val < l2->val) {
                lastNode->next = l1;
                l1 = l1->next;
            } else {
                lastNode->next = l2;
                l2 = l2->next;
            }

            lastNode = lastNode->next;
        }

        // do not forget this line!
        lastNode->next = (NULL != l1) ? l1 : l2;
    }
}
```

```
        return dummy->next;
    }
};
```

1. dummy->next
2. dummy lastNode lastNode dummy
3. l1,l2|l1/l2 lastNode->next lastNode
4. l1/l2while lastNode->next
5. dummy->next

lastNode dummy->next lastNode dummy

1. 2.

la

$$O(1). \quad \text{lastNode} \quad O(l_1 + l_2). \quad O(1).$$

Reference

- [Merge Two Sorted Lists](#) |

Merge k Sorted Lists

Source

- leetcode: Merge k Sorted Lists | LeetCode OJ
- lintcode: (104) Merge k Sorted Lists

1 - (TLE)

Merge Two Sorted Lists | Data Structure and Algorithm k lastNode->next ()
lastNode k NULL , dummy->next .

TLE.

C++

```
/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */

class Solution {
public:
    /**
     * @param lists: a list of ListNode
     * @return: The head of one sorted list.
     */
    ListNode *mergeKLists(vector<ListNode *> &lists) {
        if (lists.empty()) {
            return NULL;
        }

        ListNode *dummy = new ListNode(INT_MAX);
        ListNode *last = dummy;

        while (true) {
            int count = 0;
            int index = -1, tempVal = INT_MAX;
            for (int i = 0; i != lists.size(); ++i) {
                if (NULL == lists[i]) {
                    ++count;
                    if (count == lists.size()) {
                        last->next = NULL;
                        return dummy->next;
                    }
                } else if (lists[i]->val < tempVal) {
                    tempVal = lists[i]->val;
                    index = i;
                }
            }
            if (-1 == index) {
                break;
            }
            last->next = lists[index];
            last = last->next;
            lists[index] = lists.back();
            lists.pop_back();
        }
    }
}
```

```

        return dummy->next;
    }
    continue;
}

// choose the min value in non-NULL ListNode
if (NULL != lists[i] && lists[i]->val <= tempVal) {
    tempVal = lists[i]->val;
    index = i;
}
}

last->next = lists[index];
last = last->next;
lists[index] = lists[index]->next;
}
}
};


```

1. dummy
2. last
3. count NULL vector
4. tempVal vector index

for $O(k \cdot \sum_{i=1}^k l_i)$. $O(1)$.

2 - Merge Two Sorted Lists (TLE)

12 Merge Two Sorted Lists | Data Structure and Algorithm. 123 vector

soulmachine

C++

```


/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */

class Solution {
public:


```

```

/**
 * @param lists: a list of ListNode
 * @return: The head of one sorted list.
 */
ListNode *mergeKLists(vector<ListNode *> &lists) {
    if (lists.empty()) {
        return NULL;
    }

    ListNode *head = lists[0];
    for (int i = 1; i != lists.size(); ++i) {
        head = merge2Lists(head, lists[i]);
    }

    return head;
}

private:
    ListNode *merge2Lists(ListNode *left, ListNode *right) {
        ListNode *dummy = new ListNode(0);
        ListNode *last = dummy;

        while (NULL != left && NULL != right) {
            if (left->val < right->val) {
                last->next = left;
                left = left->next;
            } else {
                last->next = right;
                right = right->next;
            }
            last = last->next;
        }

        last->next = (NULL != left) ? left : right;

        return dummy->next;
    }
};

```

mergeKLists lists[0] , for head lists[i] .

$$O(l_1 + l_2), \quad l_1 + l_1 + l_2 + \dots + l_1 + l_2 + \dots + l_k = O(\sum_{i=1}^k (k - i) \cdot l_i). \text{121}$$

...

3 - Merge Two Sorted Lists

2 merge2Lists mergeKLists for i l_i $k - i$ $\log k$, $\log k \cdot \sum_{i=1}^k l_i$.

C++

```

/**
 * Definition of ListNode
 */
class ListNode {
public:
    int val;
    ListNode *next;
    ListNode(int val) {
        this->val = val;
        this->next = NULL;
    }
};

class Solution {
public:
    /**
     * @param lists: a list of ListNode
     * @return: The head of one sorted list.
     */
    ListNode *mergeKLists(vector<ListNode *> &lists) {
        if (lists.empty()) {
            return NULL;
        }

        return helper(lists, 0, lists.size() - 1);
    }

private:
    ListNode *helper(vector<ListNode *> &lists, int start, int end) {
        if (start == end) {
            return lists[start];
        } else if (start + 1 == end) {
            return merge2Lists(lists[start], lists[end]);
        }

        ListNode *left = helper(lists, start, start + (end - start) / 2);
        ListNode *right = helper(lists, start + (end - start) / 2 + 1, end);

        return merge2Lists(left, right);
    }

    ListNode *merge2Lists(ListNode *left, ListNode *right) {
        ListNode *dummy = new ListNode(0);
        ListNode *last = dummy;

        while (NULL != left && NULL != right) {
            if (left->val < right->val) {
                last->next = left;
                left = left->next;
            } else {
                last->next = right;
                right = right->next;
            }
            last = last->next;
        }
        last->next = (NULL != left) ? left : right;

        return dummy->next;
    }
};

```

```
    }
};

helper    helper

1. start == end start + 1 == end .
2. helper , left right
3. merge2Lists    lists[start] start ...

mergeKLists helper end lists.size() - 1 lists.size() .
```

$$\log k \cdot \sum_{i=1}^k l_i, \quad O(1).$$

2500+ms 40ms

Reference

- soulmachine. [soulmachineLeetCode](#) ↵

Reorder List

Source

- leetcode: [Reorder List | LeetCode OJ](#)
- lintcode: [\(99\) Reorder List](#)

Given a singly linked list L: L₀→L₁→...→L_{n-1}→L_n,
reorder it to: L₀→L_n→L₁→L_{n-1}→L₂→L_{n-2}→...

You must do this in-place without altering the nodes' values.

Example

For example,

Given 1->2->3->4->null, reorder it to 1->4->2->3->null.

1 - (TLE)

```
n      i , n - 2*i (),
```

TLE

C++ - TLE

```
/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: void
     */
    void reorderList(ListNode *head) {
        if (NULL == head || NULL == head->next || NULL == head->next->next) {
            return;
        }

        ListNode *last = head;
        int length = 0;
```

```

        while (NULL != last) {
            last = last->next;
            ++length;
        }

        last = head;
        for (int i = 1; i < length - i; ++i) {
            ListNode *beforeTail = last;
            for (int j = i; j < length - i; ++j) {
                beforeTail = beforeTail->next;
            }

            ListNode *temp = last->next;
            last->next = beforeTail->next;
            last->next->next = temp;
            beforeTail->next = NULL;
            last = temp;
        }
    }
};


```

- 1.
- 2.
3. last beforeTail
4. last

1. $O(n)$.
2. $\text{for } (n - 2) + (n - 4) + \dots + 2 = O(\frac{1}{2} \cdot n^2)$.
3. $O(n^2)$,

i j $i < \text{length} + 1 - i$,

2 -

11 for j i j ... —

C++

```

/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *     }
 * };
 */


```

```

        this->next = NULL;
    }
}
*/
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: void
     */
    void reorderList(ListNode *head) {
        if (NULL == head || NULL == head->next || NULL == head->next->next) {
            return;
        }

        ListNode *middle = findMiddle(head);
        ListNode *right = reverse(middle->next);
        middle->next = NULL;

        merge(head, right);
    }

private:
    void merge(ListNode *left, ListNode *right) {
        ListNode *dummy = new ListNode(0);
        while (NULL != left && NULL != right) {
            dummy->next = left;
            left = left->next;
            dummy = dummy->next;
            dummy->next = right;
            right = right->next;
            dummy = dummy->next;
        }

        dummy->next = (NULL != left) ? left : right;
        //delete dummy; /* bug, delete the tail node */
    }

    ListNode *reverse(ListNode *head) {
        ListNode *newHead = NULL;
        while (NULL != head) {
            ListNode *temp = head->next;
            head->next = newHead;
            newHead = head;
            head = temp;
        }

        return newHead;
    }

    ListNode *findMiddle(ListNode *head) {
        if (NULL == head || NULL == head->next) {
            return head;
        }

        ListNode *slow = head, *fast = head->next;
        while (NULL != fast && NULL != fast->next) {
            fast = fast->next->next;
            slow = slow->next;
        }
    }
}

```

```
    }

    return slow;
}

};
```

12

1. head->next
- 2.
3. left , right dummy
4. new

$O(n)$. $O(n/2)$. $O(n/2)$. $O(n)$.

Reference

- [Reorder List](#) |

Copy List with Random Pointer

Source

- leetcode: [Copy List with Random Pointer | LeetCode OJ](#)
- lintcode: [\(105\) Copy List with Random Pointer](#)

A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

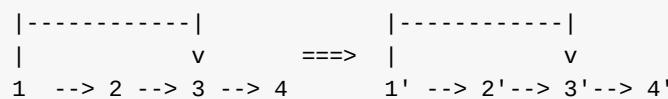
Return a deep copy of the list.

1 - ()

random random

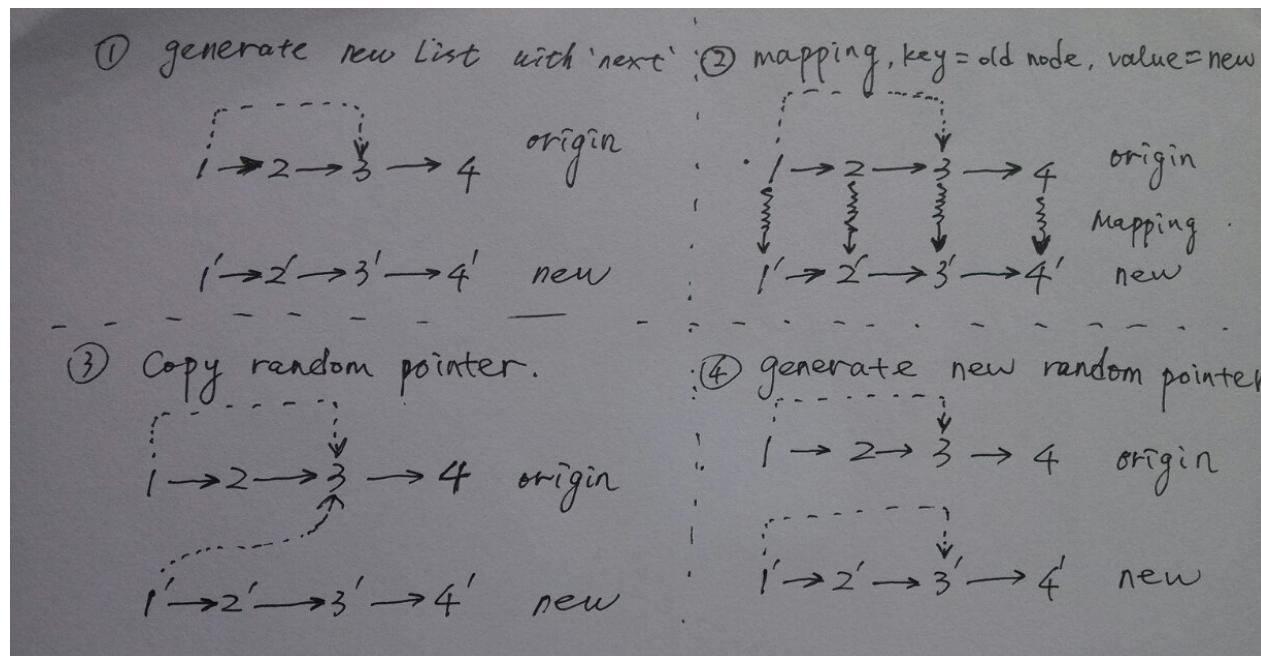
next next

hash table hash table key value random



1. next
- 2.
3. random
4. random

1, 2, 3



Python

```

# Definition for singly-linked list with a random pointer.
# class RandomListNode:
#     def __init__(self, x):
#         self.label = x
#         self.next = None
#         self.random = None
class Solution:
    # @param head: A RandomListNode
    # @return: A RandomListNode
    def copyRandomList(self, head):
        dummy = RandomListNode(0)
        curNode = dummy
        randomMap = {}

        while head is not None:
            # link newNode to new List
            newNode = RandomListNode(head.label)
            curNode.next = newNode
            # map old node head to newNode
            randomMap[head] = newNode
            # copy old node random pointer
            newNode.random = head.random
            #
            head = head.next
            curNode = curNode.next

        # re-mapping old random node to new node
        curNode = dummy.next
        while curNode is not None:
            if curNode.random is not None:
                curNode.random = randomMap[curNode.random]
            curNode = curNode.next

        return dummy.next
    
```

C++

```

/**
 * Definition for singly-linked list with a random pointer.
 * struct RandomListNode {
 *     int label;
 *     RandomListNode *next, *random;
 *     RandomListNode(int x) : label(x), next(NULL), random(NULL) {}
 * };
 */
class Solution {
public:
    /**
     * @param head: The head of linked list with a random pointer.
     * @return: A new head of a deep copy of the list.
     */
    RandomListNode *copyRandomList(RandomListNode *head) {
        if (head == NULL) return NULL;

        RandomListNode *dummy = new RandomListNode(0);
        RandomListNode *curNode = dummy;
        unordered_map<RandomListNode *, RandomListNode *> randomMap;
        while(head != NULL) {
            // link newNode to new List
            RandomListNode *newNode = new RandomListNode(head->label);
            curNode->next = newNode;
            // map old node head to newNode
            randomMap[head] = newNode;
            // copy old node random pointer
            newNode->random = head->random;

            head = head->next;
            curNode = curNode->next;
        }

        // re-mapping old random node to new node
        curNode = dummy->next;
        while (curNode != NULL) {
            if (curNode->random != NULL) {
                curNode->random = randomMap[curNode->random];
            }
            curNode = curNode->next;
        }

        return dummy->next;
    }
};

```

Java

```

/**
 * Definition for singly-linked list with a random pointer.
 * class RandomListNode {
 *     int label;
 *     RandomListNode next, random;
 *

```

```

*      RandomListNode(int x) { this.label = x; }
* };
*/
public class Solution {
    /**
     * @param head: The head of linked list with a random pointer.
     * @return: A new head of a deep copy of the list.
     */
    public RandomListNode copyRandomList(RandomListNode head) {
        if (head == null) return null;

        RandomListNode dummy = new RandomListNode(0);
        RandomListNode curNode = dummy;
        HashMap<RandomListNode, RandomListNode> randomMap = new HashMap<RandomListNode, R
        while (head != null) {
            // link newNode to new List
            RandomListNode newNode = new RandomListNode(head.label);
            curNode.next = newNode;
            // map old node head to newNode
            randomMap.put(head, newNode);
            // copy old node random pointer
            newNode.random = head.random;
            //
            head = head.next;
            curNode = curNode.next;
        }

        // re-mapping old random node to new node
        curNode = dummy.next;
        while(curNode != null) {
            if (curNode.random != null) {
                curNode.random = randomMap.get(curNode.random);
            }
            curNode = curNode.next;
        }

        return dummy.next;
    }
}

```

1. dummy random
2. random random "" node
3. node node""

$$O(2n) = O(n), \text{ TLE (Python). } O(n).$$

2 - ()

1 random 1 random random random () random

Python

```

# Definition for singly-linked list with a random pointer.
# class RandomListNode:
#     def __init__(self, x):
#         self.label = x
#         self.next = None
#         self.random = None
class Solution:
    # @param head: A RandomListNode
    # @return: A RandomListNode
    def copyRandomList(self, head):
        dummy = RandomListNode(0)
        curNode = dummy
        hash_map = {}

        while head is not None:
            # link newNode to new List
            if head in hash_map.keys():
                newNode = hash_map[head]
            else:
                newNode = RandomListNode(head.label)
                hash_map[head] = newNode
            curNode.next = newNode
            # map old node head to newNode
            hash_map[head] = newNode
            # copy old node random pointer
            if head.random is not None:
                if head.random in hash_map.keys():
                    newNode.random = hash_map[head.random]
                else:
                    newNode.random = RandomListNode(head.random.label)
                    hash_map[head.random] = newNode.random
            #
            head = head.next
            curNode = curNode.next

        return dummy.next

```

C++

```

/**
 * Definition for singly-linked list with a random pointer.
 */
struct RandomListNode {
    int label;
    RandomListNode *next, *random;
    RandomListNode(int x) : label(x), next(NULL), random(NULL) {}
};

class Solution {
public:
    /**
     * @param head: The head of linked list with a random pointer.
     * @return: A new head of a deep copy of the list.
     */

```

```

RandomListNode *copyRandomList(RandomListNode *head) {
    RandomListNode *dummy = new RandomListNode(0);
    RandomListNode *curNode = dummy;
    unordered_map<RandomListNode *, RandomListNode *> hash_map;
    while(head != NULL) {
        // link newNode to new List
        RandomListNode *newNode = NULL;
        if (hash_map.count(head) > 0) {
            newNode = hash_map[head];
        } else {
            newNode = new RandomListNode(head->label);
            hash_map[head] = newNode;
        }
        curNode->next = newNode;
        // re-mapping old random node to new node
        if (head->random != NULL) {
            if (hash_map.count(head->random) > 0) {
                newNode->random = hash_map[head->random];
            } else {
                newNode->random = new RandomListNode(head->random->label);
                hash_map[head->random] = newNode->random;
            }
        }
        head = head->next;
        curNode = curNode->next;
    }

    return dummy->next;
}
};

```

Java

```

/**
 * Definition for singly-linked list with a random pointer.
 */
class RandomListNode {
    int label;
    RandomListNode next, random;
    RandomListNode(int x) { this.label = x; }
}
public class Solution {
    /**
     * @param head: The head of linked list with a random pointer.
     * @return: A new head of a deep copy of the list.
     */
    public RandomListNode copyRandomList(RandomListNode head) {
        RandomListNode dummy = new RandomListNode(0);
        RandomListNode curNode = dummy;
        HashMap<RandomListNode, RandomListNode> hash_map = new HashMap<RandomListNode, Ra
        while (head != null) {
            // link newNode to new List
            RandomListNode newNode = null;
            if (hash_map.containsKey(head)) {
                newNode = hash_map.get(head);
            } else {

```

```

        newNode = new RandomListNode(head.label);
        hash_map.put(head, newNode);
    }
    curNode.next = newNode;
    // re-mapping old random node to new node
    if (head.random != null) {
        if (hash_map.containsKey(head.random)) {
            newNode.random = hash_map.get(head.random);
        } else {
            newNode.random = new RandomListNode(head.random.label);
            hash_map.put(head.random, newNode.random);
        }
    }
    //
    head = head.next;
    curNode = curNode.next;
}
}

return dummy.next;
}
}

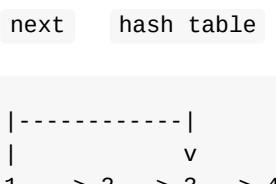
```

key C++ C++ 11 unordered_map map count key find find iterator

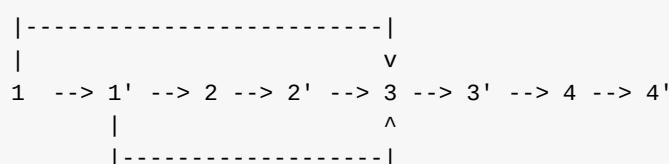
key $O(n)$, $O(n)$.

3 -

hash table random hash table

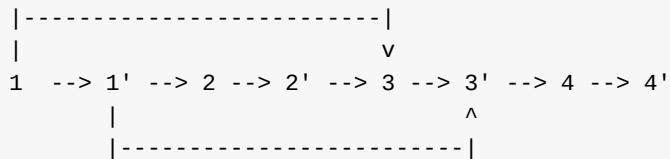


1 random 3 next



```
random random 11'3
```

```
random 1' random 3' random next
```



```
next          node->next->random = node->random->next    node->next
```

Python

```

# Definition for singly-linked list with a random pointer.
# class RandomListNode:
#     def __init__(self, x):
#         self.label = x
#         self.next = None
#         self.random = None
class Solution:
    # @param head: A RandomListNode
    # @return: A RandomListNode
    def copyRandomList(self, head):
        if head is None:
            return None

        curr = head
        # step1: generate new List with node
        while curr is not None:
            newNode = RandomListNode(curr.label)
            newNode.next = curr.next
            curr.next = newNode
            curr = curr.next.next

        # step2: copy random pointer
        curr = head
        while curr is not None:
            if curr.random is not None:
                curr.next.random = curr.random.next
            curr = curr.next.next
        # step3: split original and new List
        newHead = head.next
        curr = head
        while curr is not None:
            newNode = curr.next
            curr.next = curr.next.next
            if newNode.next is not None:
                newNode.next = newNode.next.next
            curr = curr.next

```

```
    return newHead
```

C++

```
/*
 * Definition for singly-linked list with a random pointer.
 */
class Solution {
public:
    /**
     * @param head: The head of linked list with a random pointer.
     * @return: A new head of a deep copy of the list.
     */
    RandomListNode *copyRandomList(RandomListNode *head) {
        if (head == NULL) return NULL;

        RandomListNode *curr = head;
        // step1: generate new List with node
        while (curr != NULL) {
            RandomListNode *newNode = new RandomListNode(curr->label);
            newNode->next = curr->next;
            curr->next = newNode;
            //
            curr = curr->next->next;
        }
        // step2: copy random
        curr = head;
        while (curr != NULL) {
            if (curr->random != NULL) {
                curr->next->random = curr->random->next;
            }
            curr = curr->next->next;
        }
        // step3: split original and new List
        RandomListNode *newHead = head->next;
        curr = head;
        while (curr != NULL) {
            RandomListNode *newNode = curr->next;
            curr->next = curr->next->next;
            curr = curr->next;
            if (newNode->next != NULL) {
                newNode->next = newNode->next->next;
            }
        }
        return newHead;
    }
};
```

Java

```

/**
 * Definition for singly-linked list with a random pointer.
 */
class RandomListNode {
    int label;
    RandomListNode next, random;
    RandomListNode(int x) { this.label = x; }
}
public class Solution {
    /**
     * @param head: The head of linked list with a random pointer.
     * @return: A new head of a deep copy of the list.
     */
    public RandomListNode copyRandomList(RandomListNode head) {
        if (head == null) return null;

        RandomListNode curr = head;
        // step1: generate new List with node
        while (curr != null) {
            RandomListNode newNode = new RandomListNode(curr.label);
            newNode.next = curr.next;
            curr.next = newNode;
            //
            curr = curr.next.next;
        }
        // step2: copy random pointer
        curr = head;
        while (curr != null) {
            if (curr.random != null) {
                curr.next.random = curr.random.next;
            }
            curr = curr.next.next;
        }
        // step3: split original and new List
        RandomListNode newHead = head.next;
        curr = head;
        while (curr != null) {
            RandomListNode newNode = curr.next;
            curr.next = curr.next.next;
            curr = curr.next;
            if (newNode.next != null) {
                newNode.next = newNode.next.next;
            }
        }
        return newHead;
    }
}

```

.next.next

$O(n)$ hash table $O(1)$

Reference

- [Copy List with Random Pointer - siddontang's leetcode Solution Book](#)
- [Copy List with Random Pointer Java/C++/Python](#)
- [Copy List with Random Pointer - Code Ganker](#)

Sort List

Source

- leetcode: Sort List | LeetCode OJ
- lintcode: (98) Sort List

Sort a linked list in $O(n \log n)$ time using constant space complexity.

1 - ()

$O(n \log n)$

$O(n)$

next

- 1.
 2. [Merge Two Sorted Lists | Data Structure and Algorithm](#)
-

```
/**
 * Definition of ListNode
 */
class ListNode {
public:
    int val;
    ListNode *next;
    ListNode(int val) {
        this->val = val;
        this->next = NULL;
    }
}
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: You should return the head of the sorted linked list,
              using constant space complexity.
     */
    ListNode *sortList(ListNode *head) {
        if (NULL == head) {
            return NULL;
        }

        // get the length of List
        int len = 0;
        ListNode *node = head;
```

```

        while (NULL != node) {
            node = node->next;
            ++len;
        }

        return sortListHelper(head, len);
    }

private:
    ListNode *sortListHelper(ListNode *head, const int length) {
        if ((NULL == head) || (0 >= length)) {
            return head;
        }

        ListNode *midNode = head;

        int count = 1;
        while (count < length / 2) {
            midNode = midNode->next;
            ++count;
        }

        ListNode *rList = sortListHelper(midNode->next, length - length / 2);
        midNode->next = NULL;
        ListNode *lList = sortListHelper(head, length / 2);

        return mergeList(lList, rList);
    }

    ListNode *mergeList(ListNode *l1, ListNode *l2) {
        ListNode *dummy = new ListNode(0);
        ListNode *lastNode = dummy;
        while ((NULL != l1) && (NULL != l2)) {
            if (l1->val < l2->val) {
                lastNode->next = l1;
                l1 = l1->next;
            } else {
                lastNode->next = l2;
                l2 = l2->next;
            }
            lastNode = lastNode->next;
        }

        lastNode->next = (NULL != l1) ? l1 : l2;

        return dummy->next;
    }
};

```

1. [Merge Two Sorted Lists | Data Structure and Algorithm.](#)
2. [Convert Sorted List to Binary Search Tree | Data Structure and Algorithm](#)
3. `next NULL , midNode midNode->next`

count`count 1 length / 2 ,`

4. ()

$O(n)$,	$\log(n)$	$n/2$	$n/2 \cdot O(\log n)$	0,	$O(n/2)$	$O(n)$
$O(n \log n)$,						

2 - ()**C++**

```


/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: You should return the head of the sorted linked list,
              using constant space complexity.
     */
    ListNode *sortList(ListNode *head) {
        if (NULL == head || NULL == head->next) {
            return head;
        }

        ListNode *midNode = findMiddle(head);
        ListNode *rList = sortList(midNode->next);
        midNode->next = NULL;
        ListNode *lList = sortList(head);

        return mergeList(lList, rList);
    }

private:
    ListNode *findMiddle(ListNode *head) {
        if (NULL == head || NULL == head->next) {
            return head;
        }

        ListNode *slow = head, *fast = head->next;
        while(NULL != fast && NULL != fast->next) {
            fast = fast->next->next;


```

```

        slow = slow->next;
    }

    return slow;
}

ListNode *mergeList(ListNode *l1, ListNode *l2) {
    ListNode *dummy = new ListNode(0);
    ListNode *lastNode = dummy;
    while ((NULL != l1) && (NULL != l2)) {
        if (l1->val < l2->val) {
            lastNode->next = l1;
            l1 = l1->next;
        } else {
            lastNode->next = l2;
            l2 = l2->next;
        }

        lastNode = lastNode->next;
    }

    lastNode->next = (NULL != l1) ? l1 : l2;

    return dummy->next;
}
};

```

1. head , head->next ,
2. fast head->next 1->2->null fast_slow_pointer
 - sortList head->next
3. merge

Reference

- Sort List |
- fast_slow_pointer. [LeetCode: Sort List - Yu's Garden - ↵](#)

Insertion Sort List

Source

- leetcode: [Insertion Sort List | LeetCode OJ](#)
- lintcode: [\(173\) Insertion Sort List](#)

Sort a linked list using insertion sort.

Example

Given $1 \rightarrow 3 \rightarrow 2 \rightarrow 0 \rightarrow \text{null}$, return $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \text{null}$.

1 -

Insertion Sort

dummy next dummy->next dummy->next

Python

```
"""
Definition of ListNode
class ListNode(object):

    def __init__(self, val, next=None):
        self.val = val
        self.next = next
"""

class Solution:
    """
    @param head: The first node of linked list.
    @return: The head of linked list.
    """
    def insertionSortList(self, head):
        dummy = ListNode(0)
        cur = head
        while cur is not None:
            pre = dummy
            while pre.next is not None and pre.next.val < cur.val:
                pre = pre.next
            temp = cur.next
            cur.next = pre.next
            pre.next = cur
            cur = temp
        return dummy.next
```

C++

```

/**
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: The head of linked list.
     */
    ListNode *insertionSortList(ListNode *head) {
        ListNode *dummy = new ListNode(0);
        ListNode *cur = head;
        while (cur != NULL) {
            ListNode *pre = dummy;
            while (pre->next != NULL && pre->next->val < cur->val) {
                pre = pre->next;
            }
            ListNode *temp = cur->next;
            cur->next = pre->next;
            pre->next = cur;
            cur = temp;
        }

        return dummy->next;
    }
};

```

Java

```

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode insertionSortList(ListNode head) {
        ListNode dummy = new ListNode(0);
        ListNode cur = head;
        while (cur != null) {
            ListNode pre = dummy;
            while (pre.next != null && pre.next.val < cur.val) {
                pre = pre.next;
            }
            ListNode temp = cur.next;

```

```

        cur.next = pre.next;
        pre.next = cur;
        cur = temp;
    }

    return dummy.next;
}
}

```

1. dummy
2. cur dummy cur cur .
3. pre cur
4. pre pre->next cur->next cur pre->next , cur
5. dummy->next

Python lintcode [TLE](#), leetcode `if A is not None: if A:`

Stack

[Overflow](#)

i , $1/2O(n^2) + O(n)$, dummy pre, $O(1)$.

$1/2O(n^2)$, $O(n^2)$, $O(1)$.

2 -

1 $O(n^2)$ $O(n)$ 1

Python

```

"""
Definition of ListNode
class ListNode(object):

    def __init__(self, val, next=None):
        self.val = val
        self.next = next
"""

class Solution:
    """
    @param head: The first node of linked list.
    @return: The head of linked list.
    """
    def insertionSortList(self, head):
        dummy = ListNode(0)
        dummy.next = head
        cur = head
        while cur is not None:

```

```

    if cur.next is not None and cur.next.val < cur.val:
        # find insert position for smaller(cur->next)
        pre = dummy
        while pre.next is not None and pre.next.val < cur.next.val:
            pre = pre.next
        # insert cur->next after pre
        temp = pre.next
        pre.next = cur.next
        cur.next = cur.next.next
        pre.next.next = temp
    else:
        cur = cur.next
return dummy.next

```

C++

```

/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: The head of linked list.
     */
    ListNode *insertionSortList(ListNode *head) {
        ListNode *dummy = new ListNode(0);
        dummy->next = head;
        ListNode *cur = head;
        while (cur != NULL) {
            if (cur->next != NULL && cur->next->val < cur->val) {
                ListNode *pre = dummy;
                // find insert position for smaller(cur->next)
                while (pre->next != NULL && pre->next->val <= cur->next->val) {
                    pre = pre->next;
                }
                // insert cur->next after pre
                ListNode *temp = pre->next;
                pre->next = cur->next;
                cur->next = cur->next->next;
                pre->next->next = temp;
            } else {
                cur = cur->next;
            }
        }
        return dummy->next;
    }
}

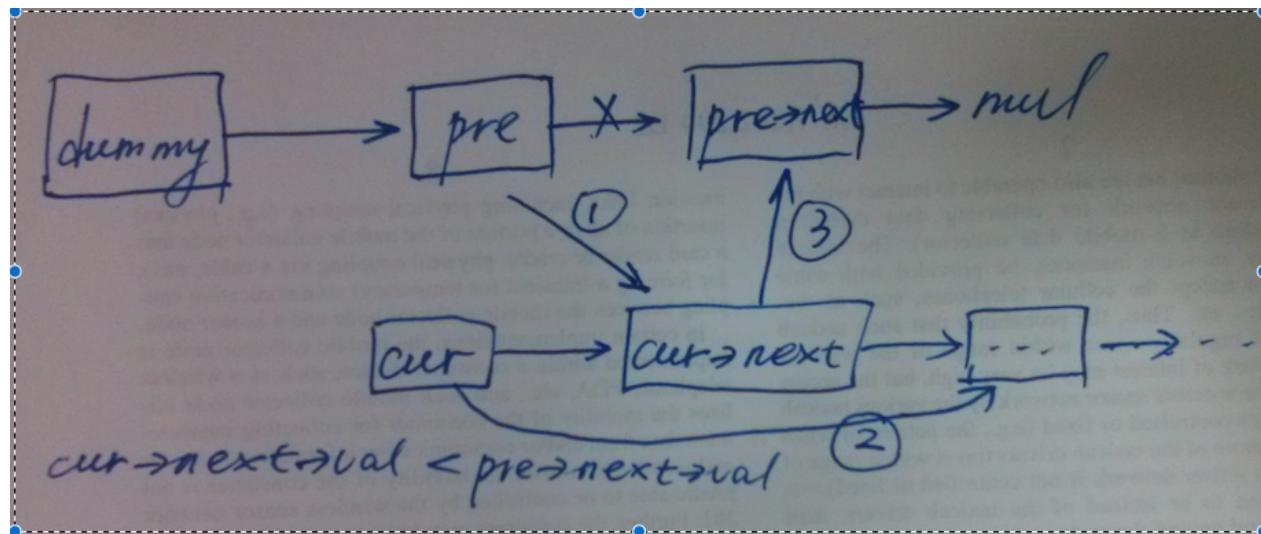
```

};

Java

```
/*
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
public class Solution {
    public ListNode insertionSortList(ListNode head) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode cur = head;
        while (cur != null) {
            if (cur.next != null && cur.next.val < cur.val) {
                // find insert position for smaller(cur->next)
                ListNode pre = dummy;
                while (pre.next != null && pre.next.val < cur.next.val) {
                    pre = pre.next;
                }
                // insert cur->next after pre
                ListNode temp = pre.next;
                pre.next = cur.next;
                cur.next = cur.next.next;
                pre.next.next = temp;
            } else {
                cur = cur.next;
            }
        }
        return dummy.next;
    }
}
```

1. dummy next head
- 2.
3. (cur->next cur)
4. cur->next pre



`cur->next pre 3`

$O(n)$, 1.

Reference

- Explained C++ solution (24ms) - Leetcode Discuss
- Insertion Sort List -

Check if a singly linked list is palindrome

- tags: [palindrome, linked_list]

Source

- Function to check if a singly linked list is palindrome - GeeksforGeeks

Given a singly linked list of characters, write a function that returns true if the given list is palindrome, else false.

1 -

(FILO)()

Java

```
/*
 * Definition for singly-linked list.
 */
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class Solution {
    public static boolean isPalindrome(ListNode head) {
        ListNode fast = head;
        ListNode slow = head;
        Stack<Integer> stack = new Stack<Integer>();

        // push node before mid
        while (fast != null && fast.next != null) {
            stack.push(slow.val);
            slow = slow.next;
            fast = fast.next.next;
        }

        // skip mid node for odd size
        if (fast != null) {
            slow = slow.next;
        }

        while (slow != null) {
            int top = stack.pop();
            // compare top with slow.val
            if (top != slow.val) {
                return false;
            }
        }
    }
}
```

```

        slow = slow.next;
    }

    return true;
}

public static void main (String[] args) {
    int len = 9;
    ListNode head = new ListNode(0);
    ListNode node = head;
    for (int i = 1; i < 9; i++) {
        int temp = (i >= len / 2) ? (len - i - 1) : i;
        node.next = new ListNode(temp);
        node = node.next;
    }

    System.out.println(isPalindrome(head));
}
}

```

$$O(\frac{1}{2}n), \quad O(n).$$

2 -

1

- 1.
- 2.
- 3.
- 4.

Java

```

/*
 * Definition for singly-linked list.
 */
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class Solution {
    public static boolean isPalindrome(ListNode head) {
        ListNode fast = head;

```

```

ListNode slow = head;
// push node before mid
while (fast != null && fast.next != null) {
    slow = slow.next;
    fast = fast.next.next;
}
// skip mid node for odd number
if (fast != null) {
    slow = slow.next;
}

ListNode rightHead = reverse(slow);
ListNode rCurr = rightHead;
ListNode lCurr = head;
while (rCurr != null) {
    if (rCurr.val != lCurr.val) {
        return false;
    }
    lCurr = lCurr.next;
    rCurr = rCurr.next;
}
// recover list
rightHead = reverse(rightHead);

return true;
}

public static ListNode reverse (ListNode head) {
    ListNode prev = null;
    ListNode curr = head;
    while (curr != null) {
        ListNode temp = curr.next;
        curr.next = prev;
        prev = curr;
        curr = temp;
    }

    return prev;
}

public static void main (String[] args) {
    int len = 9;
    ListNode head = new ListNode(0);
    ListNode node = head;
    for (int i = 1; i < 9; i++) {
        int temp = (i >= len / 2) ? (len - i - 1) : i;
        node.next = new ListNode(temp);
        node = node.next;
    }

    System.out.println(isPalindrome(head));
}
}

```

$O(n)$, $O(1)$.

3 -

i n-i i n-i ()() Java C/C++

Java

```
/*
 * Definition for singly-linked list.
 */
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) { val = x; }
}

public class Solution {
    private class Result {
        ListNode node;
        boolean isp;
        Result(ListNode aNode, boolean ret) {
            isp = ret;
            node = aNode;
        }
    }

    public Result helper(ListNode left, ListNode right) {
        Result result = new Result(left, true);

        if (right == null) return result;

        result = helper(left, right.next);
        boolean isp = (right.val == result.node.val);
        if (!isp) {
            result.isp = false;
        }
        result.node = result.node.next;

        return result;
    }

    public boolean isPalindrome(ListNode head) {
        Result ret = helper(head, head);
        return ret.isp;
    }

    public static void main (String[] args) {
        int len = 9;
        ListNode head = new ListNode(0);
        ListNode node = head;
        for (int i = 1; i < 9; i++) {
            int temp = (i >= len / 2) ? (len - i - 1) : i;
            node.next = new ListNode(temp);
            node = node.next;
        }
    }
}
```

```
        node.next = new ListNode(temp);
        node = node.next;
    }

    Solution ret = new Solution();
    System.out.println(ret.isPalindrome(head));
}
}
```

Result result `result.node = result.node.next ,`

n $O(n)$, $O(1)$.

Reference

- Function to check if a singly linked list is palindrome - GeeksforGeeks
- | The-Art-Of-Programming-By-July/01.04.md
- ctci/QuestionB.java at master · gaylemcd/ctci

Delete Node in the Middle of Singly Linked List

Source

- lintcode: [\(372\) Delete Node in the Middle of Singly Linked List](#)

```
Implement an algorithm to delete a node in the middle of a singly linked list,
given only access to that node.
```

Example

Given 1->2->3->4, and node 3. return 1->2->4

:)

Java

```
/*
 * Definition for ListNode.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int val) {
 *         this.val = val;
 *         this.next = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param node: the node in the list should be deleted
     * @return: nothing
     */
    public void deleteNode(ListNode node) {
        if (node == null) return;
        if (node.next == null) node = null;

        node.val = node.next.val;
        node.next = node.next.next;
    }
}
```

$O(1)$.

Maximum Depth of Binary Tree# Binary Tree -

[Binary Tree | Algorithm](#)

Binary Tree Preorder Traversal

Source

- leetcode: [Binary Tree Preorder Traversal | LeetCode OJ](#)
- lintcode: [\(66\) Binary Tree Preorder Traversal](#)

Given a binary tree, return the preorder traversal of its nodes' values.

Note

Given binary tree {1,#,2,3},

```

1
 \
 2
 /
3

```

return [1,2,3].

Example

Challenge

Can you do it without recursion?

1 -

() null vector

Python - Divide and Conquer

```

"""
Definition of TreeNode:
class TreeNode:
    def __init__(self, val):
        this.val = val
        this.left, this.right = None, None
"""

class Solution:
    """
    @param root: The root of binary tree.
    @return: Preorder in ArrayList which contains node values.
    """
    def preorderTraversal(self, root):
        if root == None:
            return []
        return [root.val] + self.preorderTraversal(root.left) \
               + self.preorderTraversal(root.right)

```

```
+ self.preorderTraversal(root.right)
```

C++ - Divide and Conquer

```
/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */

class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: Preorder in vector which contains node values.
     */
    vector<int> preorderTraversal(TreeNode *root) {
        vector<int> result;
        if (root != NULL) {
            // Divide ()
            vector<int> left = preorderTraversal(root->left);
            vector<int> right = preorderTraversal(root->right);
            // Merge
            result.push_back(root->val);
            result.insert(result.end(), left.begin(), left.end());
            result.insert(result.end(), right.begin(), right.end());
        }
        return result;
    }
};
```

C++ - Traversal

```
/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
```

```

class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: Preorder in vector which contains node values.
     */
    vector<int> preorderTraversal(TreeNode *root) {
        vector<int> result;
        traverse(root, result);

        return result;
    }

private:
    void traverse(TreeNode *root, vector<int> &ret) {
        if (root != NULL) {
            ret.push_back(root->val);
            traverse(root->left, ret);
            traverse(root->right, ret);
        }
    }
};

```

Java - Divide and Conquer

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        if (root != null) {
            // Divide
            List<Integer> left = preorderTraversal(root.left);
            List<Integer> right = preorderTraversal(root.right);
            // Merge
            result.add(root.val);
            result.addAll(left);
            result.addAll(right);
        }

        return result;
    }
}

```

traverse vector C++ vector, vectorvectorpush_back, insert Java

addAll .

$O(n)$,**2 -**

()(NULL)

Python

```

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def preorderTraversal(self, root):
        if root is None:
            return []

        result = []
        s = []
        s.append(root)
        while s:
            root = s.pop()
            result.append(root.val)
            if root.right is not None:
                s.append(root.right)
            if root.left is not None:
                s.append(root.left)

        return result

```

C++

```

/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */

```

```

class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: Preorder in vector which contains node values.
     */
    vector<int> preorderTraversal(TreeNode *root) {
        vector<int> result;
        if (root == NULL) return result;

        stack<TreeNode *> s;
        s.push(root);
        while (!s.empty()) {
            TreeNode *node = s.top();
            s.pop();
            result.push_back(node->val);
            if (node->right != NULL) {
                s.push(node->right);
            }
            if (node->left != NULL) {
                s.push(node->left);
            }
        }

        return result;
    }
};

```

Java

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        if (root == null) return result;

        Stack<TreeNode> s = new Stack<TreeNode>();
        s.push(root);
        while (!s.empty()) {
            TreeNode node = s.pop();
            result.add(node.val);
            if (node.right != null) s.push(node.right);
            if (node.left != null) s.push(node.left);
        }

        return result;
    }
}

```

1. root
2. root
3. s
4. (pop)
- 5.
- 6.

4,5,6

$$O(n), \quad O(n).$$

Binary Tree Inorder Traversal

Source

- leetcode: [Binary Tree Inorder Traversal | LeetCode OJ](#)
- lintcode: [\(67\) Binary Tree Inorder Traversal](#)

Given a binary tree, return the inorder traversal of its nodes' values.

Example

Given binary tree {1,#,2,3},

```

1
 \
 2
 /
3

```

return [1,3,2].

Challenge

Can you do it without recursion?

1 -

Python

```

"""
Definition of TreeNode:
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left, self.right = None, None
"""

class Solution:
    """
    @param root: The root of binary tree.
    @return: Inorder in ArrayList which contains node values.
    """
    def inorderTraversal(self, root):
        if root is None:
            return []
        else:
            return [root.val] + self.inorderTraversal(root.left) \
                   + self.inorderTraversal(root.right)

```

Python - with helper

```

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def inorderTraversal(self, root):
        result = []
        self.helper(root, result)
        return result

    def helper(self, root, ret):
        if root is not None:
            self.helper(root.left, ret)
            ret.append(root.val)
            self.helper(root.right, ret)

```

C++

```

/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        helper(root, result);
        return result;
    }

private:
    void helper(TreeNode *root, vector<int> &ret) {
        if (root != NULL) {
            helper(root->left, ret);
            ret.push_back(root->val);
            helper(root->right, ret);
        }
    }
};

```

Java

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        helper(root, result);
        return result;
    }

    private void helper(TreeNode root, List<Integer> ret) {
        if (root != null) {
            helper(root.left, ret);
            ret.add(root.val);
            helper(root.right, ret);
        }
    }
}

```

Python $O(n)$.**2** -

- 1.
- 2.
- 3.
4. p

2,3,42

Python

```

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None

```

```

#         self.right = None

class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def inorderTraversal(self, root):
        result = []
        s = []
        while root is not None or s:
            if root is not None:
                s.append(root)
                root = root.left
            else:
                root = s.pop()
                result.append(root.val)
                root = root.right

        return result

```

C++

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: Inorder in vector which contains node values.
     */
public:
    vector<int> inorderTraversal(TreeNode *root) {
        vector<int> result;
        stack<TreeNode *> s;

        while (!s.empty() || NULL != root) {
            if (root != NULL) {
                s.push(root);
                root = root->left;
            } else {
                root = s.top();
                s.pop();
                result.push_back(root->val);
                root = root->right;
            }
        }

        return result;
    }
}

```

```
};
```

Java

```
/*
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        Stack<TreeNode> s = new Stack<TreeNode>();
        while (root != null || !s.empty()) {
            if (root != null) {
                s.push(root);
                root = root.left;
            } else {
                root = s.pop();
                result.add(root.val);
                root = root.right;
            }
        }
        return result;
    }
}
```

$O(n)$, $O(n)$.

Reference

Binary Tree Postorder Traversal

Source

- leetcode: [Binary Tree Postorder Traversal | LeetCode OJ](#)
- lintcode: [\(68\) Binary Tree Postorder Traversal](#)

Given a binary tree, return the postorder traversal of its nodes' values.

Example

Given binary tree {1,#,2,3},

```

1
 \
 2
 /
3

```

return [3,2,1].

Challenge

Can you do it without recursion?

1 -

Python - Divide and Conquer

```

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def postorderTraversal(self, root):
        if root is None:
            return []
        else:
            return self.postorderTraversal(root.left) +\
                   self.postorderTraversal(root.right) + [root.val]

```

C++ - Traversal

```

/**
 * Definition of TreeNode:
 * class TreeNode {
 *     public:
 *         int val;
 *         TreeNode *left, *right;
 *         TreeNode(int val) {
 *             this->val = val;
 *             this->left = this->right = NULL;
 *         }
 *     }
 */
class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: Postorder in vector which contains node values.
     */
public:
    vector<int> postorderTraversal(TreeNode *root) {
        vector<int> result;

        traverse(root, result);

        return result;
    }

private:
    void traverse(TreeNode *root, vector<int> &ret) {
        if (root == NULL) {
            return;
        }

        traverse(root->left, ret);
        traverse(root->right, ret);
        ret.push_back(root->val);
    }
};

```

Java - Divide and Conquer

```

/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
public class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        if (root != null) {
            List<Integer> left = postorderTraversal(root.left);
            result.addAll(left);
            List<Integer> right = postorderTraversal(root.right);
            result.addAll(right);
        }
        return result;
    }
}

```

```

        result.addAll(right);
        result.add(root.val);
    }

    return result;
}
}

```

$O(n)$.

2 -

0

Python

```

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def postorderTraversal(self, root):
        result = []
        if root is None:
            return result
        s = []
        # previously traversed node
        prev = None
        s.append(root)
        while s:
            curr = s[-1]
            noChild = curr.left is None and curr.right is None
            childVisited = (prev is not None) and \
                (curr.left == prev or curr.right == prev)
            if noChild or childVisited:
                result.append(curr.val)
                s.pop()
                prev = curr
            else:
                if curr.right is not None:
                    s.append(curr.right)
                if curr.left is not None:

```

```
s.append(curr.left)

return result
```

C++

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {
        vector<int> result;
        if (root == NULL) return result;

        TreeNode *prev = NULL;
        stack<TreeNode *> s;
        s.push(root);
        while (!s.empty()) {
            TreeNode *curr = s.top();
            bool noChild = false;
            if (curr->left == NULL && curr->right == NULL) {
                noChild = true;
            }
            bool childVisited = false;
            if (prev != NULL && (curr->left == prev || curr->right == prev)) {
                childVisited = true;
            }

            // traverse
            if (noChild || childVisited) {
                result.push_back(curr->val);
                s.pop();
                prev = curr;
            } else {
                if (curr->right != NULL) s.push(curr->right);
                if (curr->left != NULL) s.push(curr->left);
            }
        }

        return result;
    }
};
```

Java

```
/*
 * Definition for a binary tree node.
```

```

* public class TreeNode {
*     int val;
*     TreeNode left;
*     TreeNode right;
*     TreeNode(int x) { val = x; }
* }
*/
public class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();
        if (root == null) return result;

        Stack<TreeNode> s = new Stack<TreeNode>();
        s.push(root);
        TreeNode prev = null;
        while (!s.empty()) {
            TreeNode curr = s.peek();
            boolean noChild = false;
            if (curr.left == null && curr.right == null) {
                noChild = true;
            }
            boolean childVisited = false;
            if (prev != null && (curr.left == prev || curr.right == prev)) {
                childVisited = true;
            }

            // traverse
            if (noChild || childVisited) {
                result.add(curr.val);
                s.pop();
                prev = curr;
            } else {
                if (curr.right != null) s.push(curr.right);
                if (curr.left != null) s.push(curr.left);
            }
        }

        return result;
    }
}

```

prev == null prev null.

$O(n)$, $O(n)$.

3 - Iterative

C++

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 *     public:
 *         int val;
 *         TreeNode *left, *right;
 *         TreeNode(int val) {
 *             this->val = val;
 *             this->left = this->right = NULL;
 *         }
 *     }
 */
class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: Postorder in vector which contains node values.
     */
public:
    vector<int> postorderTraversal(TreeNode *root) {
        vector<int> result;
        if (root == NULL) return result;

        stack<TreeNode*> s;
        s.push(root);
        while (!s.empty()) {
            TreeNode *node = s.top();
            s.pop();
            result.push_back(node->val);
            // root, right, left => left, right, root
            if (node->left != NULL) s.push(node->left);
            if (node->right != NULL) s.push(node->right);
        }
        // reverse
        std::reverse(result.begin(), result.end());
        return result;
    }
};

```

Reference

- [\[leetcode\]Binary Tree Postorder Traversal @ Python - -](#)

• -

Binary Tree Level Order Traversal

Source

- lintcode: [\(69\) Binary Tree Level Order Traversal](#)

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

Example

Given binary tree {3,9,20,#,#,15,7},

```

      3
     / \
    9  20
   /   \
  15   7
  
```

return its level order traversal as:

```
[
  [3],
  [9,20],
  [15,7]
]
```

Challenge

Using only 1 queue to implement it.

-

for

C++

```
/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */

class Solution {
    /**
     * @param root: The root of binary tree.
     */
```

```

    * @return: Level order a list of lists of integer
    */
public:
    vector<vector<int>> levelOrder(TreeNode *root) {
        vector<vector<int>> result;

        if (NULL == root) {
            return result;
        }

        queue<TreeNode *> q;
        q.push(root);
        while (!q.empty()) {
            vector<int> list;
            int size = q.size(); // keep the queue size first
            for (int i = 0; i != size; ++i) {
                TreeNode * node = q.front();
                q.pop();
                list.push_back(node->val);
                if (node->left) {
                    q.push(node->left);
                }
                if (node->right) {
                    q.push(node->right);
                }
            }
            result.push_back(list);
        }

        return result;
    }
};

```

Java

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: Level order a list of lists of integer
     */
    public ArrayList<ArrayList<Integer>> levelOrder(TreeNode root) {
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        if (root == null) return result;

```

```
Queue<TreeNode> q = new LinkedList<TreeNode>();
q.offer(root);
while (!q.isEmpty()) {
    int qLen = q.size();
    ArrayList<Integer> aList = new ArrayList<Integer>();
    for (int i = 0; i < qLen; i++) {
        TreeNode node = q.poll();
        aList.add(node.val);
        if (node.left != null) q.offer(node.left);
        if (node.right != null) q.offer(node.right);
    }
    result.add(aList);
}

return result;
}
}
```

- 1.
2. STL queue root
- 3.
4. list

$$O(n), \quad O(n).$$

Binary Tree Level Order Traversal II

Source

- leetcode: [Binary Tree Level Order Traversal II | LeetCode OJ](#)
- lintcode: [\(70\) Binary Tree Level Order Traversal II](#)

Given a binary tree, return the bottom-up level order traversal of its nodes' values. (ie, from left to right, level by level from leaf to root).

Example

Given binary tree {3,9,20,#,#,15,7},

```

3
/
9  20
 / \
15   7

```

return its bottom-up level order traversal as:

```
[
  [15, 7],
  [9, 20],
  [3]
]
```

BFS

Java - Stack

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: bottom-up level order a list of lists of integer
     */
}

```

```

/*
public ArrayList<ArrayList<Integer>> levelOrderBottom(TreeNode root) {
    ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
    if (root == null) return result;

    Stack<ArrayList<Integer>> s = new Stack<ArrayList<Integer>>();
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.offer(root);
    while (!q.isEmpty()) {
        int qLen = q.size();
        ArrayList<Integer> aList = new ArrayList<Integer>();
        for (int i = 0; i < qLen; i++) {
            TreeNode node = q.poll();
            aList.add(node.val);
            if (node.left != null) q.offer(node.left);
            if (node.right != null) q.offer(node.right);
        }
        s.push(aList);
    }

    while (!s.empty()) {
        result.add(s.pop());
    }
    return result;
}
}

```

Java - Reverse

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: bottom-up level order a list of lists of integer
     */
    public ArrayList<ArrayList<Integer>> levelOrderBottom(TreeNode root) {
        ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
        if (root == null) return result;

        Queue<TreeNode> q = new LinkedList<TreeNode>();
        q.offer(root);
        while (!q.isEmpty()) {
            int qLen = q.size();
            ArrayList<Integer> aList = new ArrayList<Integer>();
            for (int i = 0; i < qLen; i++) {

```

```
        TreeNode node = q.poll();
        aList.add(node.val);
        if (node.left != null) q.offer(node.left);
        if (node.right != null) q.offer(node.right);
    }
    result.add(aList);
}

Collections.reverse(result);
return result;
}
}
```

Java Queue LinkedList

$O(n)$, $O(n)$.

Maximum Depth of Binary Tree

Source

- lintcode: (97) Maximum Depth of Binary Tree

Given a binary tree, find its maximum depth.

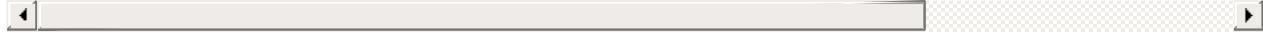
The maximum depth is the number of nodes along the longest path from the root node down to a leaf node.

Example

Given a binary tree as follow:



The maximum depth is 3



1 NULL 0

`maxDepth` $O(n)$, $n = \log_2 n$, $O(\log n)$, $O(n)$

C++ Recursion

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root: The root of binary tree.
     */
    int maxDepth(TreeNode* root) {
        if (root == NULL) {
            return 0;
        } else {
            int leftDepth = maxDepth(root->left);
            int rightDepth = maxDepth(root->right);
            return 1 + max(leftDepth, rightDepth);
        }
    }
}
  
```

```

    * @return: An integer
    */
int maxDepth(TreeNode *root) {
    if (NULL == root) {
        return 0;
    }

    int left_depth = maxDepth(root->left);
    int right_depth = maxDepth(root->right);

    return max(left_depth, right_depth) + 1;
}

```

Java Recursion

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: An integer.
     */
    public int maxDepth(TreeNode root) {
        // write your code here
        if (root == null) {
            return 0;
        }
        return Math.max(maxDepth(root.left), maxDepth(root.right)) + 1;
    }
}

```

- ()

00

max_depth

C++ Iterative with stack

```

/**
 * Definition of TreeNode:
 * class TreeNode {

```

```

* public:
*     int val;
*     TreeNode *left, *right;
*     TreeNode(int val) {
*         this->val = val;
*         this->left = this->right = NULL;
*     }
* }
*/
class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: An integer
     */
    int maxDepth(TreeNode *root) {
        if (NULL == root) {
            return 0;
        }

        TreeNode *curr = NULL, *prev = NULL;
        stack<TreeNode *> s;
        s.push(root);

        int max_depth = 0;

        while(!s.empty()) {
            curr = s.top();
            if (!prev || prev->left == curr || prev->right == curr) {
                if (curr->left) {
                    s.push(curr->left);
                } else if (curr->right){
                    s.push(curr->right);
                }
            } else if (curr->left == prev) {
                if (curr->right) {
                    s.push(curr->right);
                }
            } else {
                s.pop();
            }

            prev = curr;

            if (s.size() > max_depth) {
                max_depth = s.size();
            }
        }

        return max_depth;
    }
};

```

- ()

/ ++max_depth

C++ Iterative with queue

```

/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * };
 */
class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: An integer
     */
    int maxDepth(TreeNode *root) {
        if (NULL == root) {
            return 0;
        }

        queue<TreeNode *> q;
        q.push(root);

        int max_depth = 0;
        while(!q.empty()) {
            int size = q.size();
            for (int i = 0; i != size; ++i) {
                TreeNode *node = q.front();
                q.pop();

                if (node->left) {
                    q.push(node->left);
                }
                if (node->right) {
                    q.push(node->right);
                }
            }

            ++max_depth;
        }

        return max_depth;
    }
};

```

Java Iterative with queue

```

/**
 * Definition of TreeNode:
 * public class TreeNode {

```

```

*     public int val;
*     public TreeNode left, right;
*     public TreeNode(int val) {
*         this.val = val;
*         this.left = this.right = null;
*     }
* }
*/
public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: An integer.
     */
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }

        int level = 0;
        LinkedList<TreeNode> queue = new LinkedList<TreeNode>();
        queue.add(root);
        int curNum = 1; //num of nodes left in current level
        int nextNum = 0; //num of nodes in next level

        while(!queue.isEmpty()) {
            TreeNode n = queue.poll();
            curNum--;
            if (n.left != null) {
                queue.add(n.left);
                nextNum++;
            }
            if (n.right != null) {
                queue.add(n.right);
                nextNum++;
            }
            if (curNum == 0) {
                curNum = nextNum;
                nextNum = 0;
                level++;
            }
        }
        return level;
    }
}

```

Balanced Binary Tree

Source

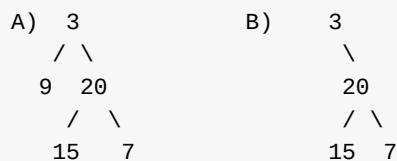
- lintcode: [\(93\) Balanced Binary Tree](#)

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as a binary tree in which the

Example

Given binary tree A={3,9,20,#,#,15,7}, B={3,#,20,15,7}



The binary tree A is a height-balanced binary tree, but B is not.



1 Maximum Depth of Binary Tree | Algorithm NULL == root 0 1
— INT_MAX or INT_MIN bool bool

C++ Recursion with extra bool variable

```

/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: True if this Binary tree is Balanced, or false.
     */
    bool isBalanced(TreeNode *root) {
        if (NULL == root) {
            return true;
        }
        if (abs(maxDepth(root->left) - maxDepth(root->right)) >= 2) {
            return false;
        }
        return isBalanced(root->left) && isBalanced(root->right);
    }

    int maxDepth(TreeNode *root) {
        if (NULL == root) {
            return 0;
        }
        return 1 + max(maxDepth(root->left), maxDepth(root->right));
    }
}
  
```

```

        return true;
    }

    bool result = true;
    maxDepth(root, result);

    return result;
}

private:
    int maxDepth(TreeNode *root, bool &isBalanced) {
        if (NULL == root) {
            return 0;
        }

        int leftDepth = maxDepth(root->left, isBalanced);
        int rightDepth = maxDepth(root->right, isBalanced);
        if (abs(leftDepth - rightDepth) > 1) {
            isBalanced = false;
            // speed up the recursion process
            return INT_MAX;
        }

        return max(leftDepth, rightDepth) + 1;
    }
};

```

1 INT_MAX

```

...     abs(leftDepth - rightDepth) > 1      -1      max(leftDepth, rightDepth) + 1
max...      -1 leftDepth rightDepth 1      -1 -1

```

C++ Recursion without extra bool variable

```

/*
* forked from http://www.jiuzhang.com/solutions/balanced-binary-tree/
* Definition of TreeNode:
* class TreeNode {
* public:
*     int val;
*     TreeNode *left, *right;
*     TreeNode(int val) {
*         this->val = val;
*         this->left = this->right = NULL;
*     }
* }
*/
class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: True if this Binary tree is Balanced, or false.
     */

```

```
bool isBalanced(TreeNode *root) {
    return (-1 != maxDepth(root));
}

private:
    int maxDepth(TreeNode *root) {
        if (NULL == root) {
            return 0;
        }

        int leftDepth = maxDepth(root->left);
        int rightDepth = maxDepth(root->right);
        if (leftDepth == -1 || rightDepth == -1 || \
            abs(leftDepth - rightDepth) > 1) {
            return -1;
        }

        return max(leftDepth, rightDepth) + 1;
    }
};
```

Binary Tree Maximum Path Sum

Source

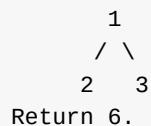
- lintcode: [\(94\) Binary Tree Maximum Path Sum](#)

Given a binary tree, find the maximum path sum.

The path may start and end at any node in the tree.

Example

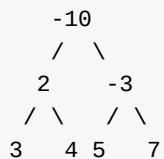
Given the below binary tree,



Return 6.

1 -

`()/()=++`



`-10 4 -> 2 -> -10 <- -3 <- 7 , 3 -> 2 4 -> 2 , 3 -> 2 <- 4 .`

// $f(\text{root})$ root $\text{root}()$
 $f(\text{root}) = \text{root} \rightarrow \text{val} + \max(f(\text{root} \rightarrow \text{left}), f(\text{root} \rightarrow \text{right}))$

$g(\text{root})$ root
 $g(\text{root}) = \text{root} \rightarrow \text{val} + f(\text{root} \rightarrow \text{left}) + f(\text{root} \rightarrow \text{right})$

$g(\text{node})$

C++ Recursion + Iteration(Not Recommended)

Time Limit Exceeded

`/**`

```

* Definition of TreeNode:
* class TreeNode {
* public:
*     int val;
*     TreeNode *left, *right;
*     TreeNode(int val) {
*         this->val = val;
*         this->left = this->right = NULL;
*     }
* }
*/
class Solution {
public:
    /**
     * @param root: The root of binary tree.
     * @return: An integer
     */
    int maxPathSum(TreeNode *root) {
        if (NULL == root) {
            return 0;
        }

        int result = INT_MIN;
        stack<TreeNode *> s;
        s.push(root);
        while (!s.empty()) {
            TreeNode *node = s.top();
            s.pop();

            int temp_path_sum = node->val + singlePathSum(node->left) \
                + singlePathSum(node->right);

            if (temp_path_sum > result) {
                result = temp_path_sum;
            }

            if (NULL != node->right) s.push(node->right);
            if (NULL != node->left) s.push(node->left);
        }

        return result;
    }

private:
    int singlePathSum(TreeNode *root) {
        if (NULL == root) {
            return 0;
        }

        int path_sum = max(singlePathSum(root->left), singlePathSum(root->right));
        return max(0, (root->val + path_sum));
    }
};

```

singlePathSum path_sum 00

2 -

C++ using std::pair

C++

pair

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 *     public:
 *         int val;
 *         TreeNode *left, *right;
 *         TreeNode(int val) {
 *             this->val = val;
 *             this->left = this->right = NULL;
 *         }
 *     }
 */
class Solution {
private:
    pair<int, int> helper(TreeNode *root) {
        if (NULL == root) {
            return make_pair(0, INT_MIN);
        }

        pair<int, int> leftTree = helper(root->left);
        pair<int, int> rightTree = helper(root->right);

        int single_path_sum = max(leftTree.first, rightTree.first) + root->val;
        single_path_sum = max(0, single_path_sum);

        int max_sub_sum = max(leftTree.second, rightTree.second);
        int max_path_sum = root->val + leftTree.first + rightTree.first;
        max_path_sum = max(max_sub_sum, max_path_sum);

        return make_pair(single_path_sum, max_path_sum);
    }

public:
    /**
     * @param root: The root of binary tree.
     * @return: An integer
     */
    int maxPathSum(TreeNode *root) {
        if (NULL == root) {
            return 0;
        }

        return helper(root).second;
    }
};

```

```

pair C++...           ...private pair    class pair
single_path_sum max_path_sum pair_is_harmful.

max_path_sum = max(max_sub_sum, max_path_sum) ,

```

The path may start and end at any node in the tree.

2

- 1.
2. (0)

-10

C++ using self-defined class

```

/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
    class ResultType {
public:
    int singlePath, maxPath;
    ResultType(int s, int m):singlePath(s), maxPath(m) {}
};

private:
    ResultType helper(TreeNode *root) {
        if (root == NULL) {
            ResultType *nullResult = new ResultType(0, INT_MIN);
            return *nullResult;
        }
        // Divide
        ResultType left = helper(root->left);
        ResultType right = helper(root->right);

        // Conquer
        int singlePath = max(left.singlePath, right.singlePath) + root->val;
        singlePath = max(singlePath, 0);

        int maxPath = max(left.maxPath, right.maxPath);
        maxPath = max(maxPath, left.singlePath + right.singlePath + root->val);

        ResultType *result = new ResultType(singlePath, maxPath);
        return *result;
    }
}

```

```
public:  
    int maxPathSum(TreeNode *root) {  
        ResultType result = helper(root);  
        return result.maxPath;  
    }  
};
```

1. `ResultType *XXX = new ResultType ...` `return *xxx` class `new` operator
2. `...private class public class`

Reference

- [pair_is_harmful . std::pair considered harmful! << Modern Maintainable Code - pair ↵](#)
- [Binary Tree Maximum Path Sum |](#)

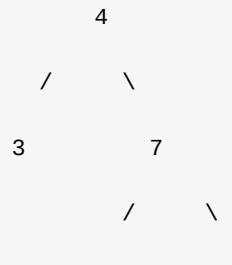
Lowest Common Ancestor

Source

- lintcode: [\(88\) Lowest Common Ancestor](#)

Given the root and two nodes in a Binary Tree. Find the lowest common ancestor(LCA) of the two nodes.

The lowest common ancestor is the node with largest depth which is the ancestor of both nodes.



For 3 and 5, the LCA is 4.

For 5 and 6, the LCA is 7.

For 6 and 7, the LCA is 7.



1 -

3 5 4 4 5 6 4 7 6 7 7 6 7 ——00

— A/B

1.

- o //
- o //
- o `NULL, NULL .//`

2.

C++ Recursion From Bottom to Top

```
/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 * 
```

```

        *      this->val = val;
        *      this->left = this->right = NULL;
        *
    }
*/
class Solution {
public:
    /**
     * @param root: The root of the binary search tree.
     * @param A and B: two nodes in a Binary.
     * @return: Return the least common ancestor(LCA) of the two nodes.
     */
    TreeNode *lowestCommonAncestor(TreeNode *root, TreeNode *A, TreeNode *B) {
        // return either A or B or NULL
        if (NULL == root || root == A || root == B) return root;

        TreeNode *left = lowestCommonAncestor(root->left, A, B);
        TreeNode *right = lowestCommonAncestor(root->right, A, B);

        // A and B are on both sides
        if ((NULL != left) && (NULL != right)) return root;

        // either left or right or NULL
        return (NULL != left) ? left : right;
    }
};

```

root == A || root == B ()

return (NULL != left) ? left : right; leetcode

A/B ...

A/B

```

public class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode A, TreeNode B) {
        if (root == null || root == A || root == B) {
            return root;
        }

        // Divide
        TreeNode left = lowestCommonAncestor(root.left, A, B);
        TreeNode right = lowestCommonAncestor(root.right, A, B);

        // Conquer
        if (left != null && right != null) {
            return root;
        }
        if (left != null) {
            return left;
        }
        if (right != null) {

```

```

        return right;
    }
    return null;
}
}

```

- ()

pair<TreeNode *, int> result(node, counter)	node LCA	counter A B 2
---	----------	---------------

C++ Post-order(counter)

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root: The root of the binary search tree.
     * @param A and B: two nodes in a Binary.
     * @return: Return the least common ancestor(LCA) of the two nodes.
     */
    TreeNode *lowestCommonAncestor(TreeNode *root, TreeNode *A, TreeNode *B) {
        if ((NULL == A) || (NULL == B)) return NULL;

        pair<TreeNode *, int> result = helper(root, A, B);

        if (A != B) {
            return (2 == result.second) ? result.first : NULL;
        } else {
            return (1 == result.second) ? result.first : NULL;
        }
    }

private:
    pair<TreeNode *, int> helper(TreeNode *root, TreeNode *A, TreeNode *B) {
        TreeNode * node = NULL;
        if (NULL == root) return make_pair(node, 0);

        pair<TreeNode *, int> left = helper(root->left, A, B);

```

```
pair<TreeNode *, int> right = helper(root->right, A, B);

// return either A or B
int count = max(left.second, right.second);
if (A == root || B == root) return make_pair(root, ++count);

// A and B are on both sides
if (NULL != left.first && NULL != right.first) return make_pair(root, 2);

// return either left or right or NULL
return (NULL != left.first) ? left : right;
}
};
```

A == B 12

Reference

- [leetcode](#). Lowest Common Ancestor of a Binary Tree Part I | LeetCode - ↵
- [Lowest Common Ancestor of a Binary Tree Part II | LeetCode](#) -
- [Lowest Common Ancestor of a Binary Search Tree \(BST\) | LeetCode](#) -
- [Lowest Common Ancestor | - Divide and Conquer parent](#)

Invert Binary Tree

Source

- leetcode: [Invert Binary Tree | LeetCode OJ](#)
- lintcode: [\(175\) Invert Binary Tree](#)

```
Invert a binary tree.
```

Example

```
      1      1
     / \    / \
2   3 => 3   2
     /       \
    4       4
```

Challenge

Do it in recursion is acceptable, can you do it without recursion?

1 - Recursive

C++ - return void

```
/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * };
 */
class Solution {
public:
    /**
     * @param root: a TreeNode, the root of the binary tree
     * @return: nothing
     */
    void invertBinaryTree(TreeNode *root) {
        if (root == NULL) return;

        TreeNode *temp = root->left;
        root->left = root->right;
        root->right = temp;

        invertBinaryTree(root->left);
    }
}
```

```

        invertBinaryTree(root->right);
    }
};

```

C++ - return TreeNode *

```

/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        if (root == NULL) return NULL;

        TreeNode *temp = root->left;
        root->left = invertTree(root->right);
        root->right = invertTree(temp);

        return root;
    }
};
```

$O(n)$, $O(1)$.

2 - Iterative

level-order

C++

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *     }
 * };
 */
```

```

*           this->left = this->right = NULL;
*       }
*   };
*/
class Solution {
public:
    /**
     * @param root: a TreeNode, the root of the binary tree
     * @return: nothing
     */
    void invertBinaryTree(TreeNode *root) {
        if (root == NULL) return;

        queue<TreeNode*> q;
        q.push(root);
        while (!q.empty()) {
            // pop out the front node
            TreeNode *node = q.front();
            q.pop();
            // swap between left and right pointer
            swap(node->left, node->right);
            // push non-NULL node
            if (node->left != NULL) q.push(node->left);
            if (node->right != NULL) q.push(node->right);
        }
    }
};

```

$O(n)$, $O(n)$.

Reference

- 0ms C++ Recursive/Iterative Solutions with Explanations - Leetcode Discuss

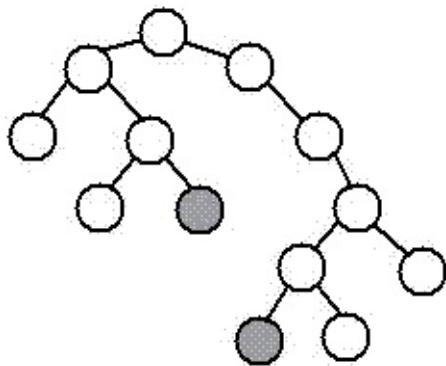
Diameter of a Binary Tree

Source

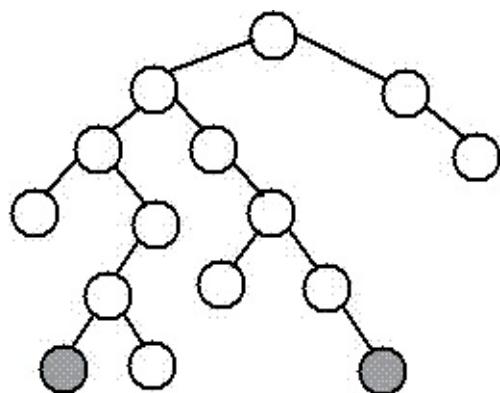
- [Diameter of a Binary Tree - GeeksforGeeks](#)

The diameter of a tree (sometimes called the width) is the number of nodes on the longest path between two leaves in the tree.

The diagram below shows two trees each with diameter nine, the leaves that form the ends of a longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).



diameter, 9 nodes, through root



diameter, 9 nodes, NOT through root

[Lowest Common Ancestor](#)

Java

```

class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class Solution {
    public int diameter(TreeNode root) {
        if (root == null) return 0;

        // left, right height
        int leftHeight = getHeight(root.left);
    }
}

```

```

        int rightHeight = getHeight(root.right);

        // left, right subtree diameter
        int leftDia = diameter(root.left);
        int rightDia = diameter(root.right);

        int maxSubDia = Math.max(leftDia, rightDia);
        return Math.max(maxSubDia, leftHeight + 1 + rightHeight);
    }

    private int getHeight(TreeNode root) {
        if (root == null) return 0;

        return 1 + Math.max(getHeight(root.left), getHeight(root.right));
    }

    public static void main(String[] args) {
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        root.left.right.left = new TreeNode(6);
        root.left.right.left.right = new TreeNode(7);
        root.left.left.left = new TreeNode(8);

        Solution sol = new Solution();
        int maxDistance = sol.diameter(root);
        System.out.println("Max Distance: " + maxDistance);
    }
}

```

Reference

- [Diameter of a Binary Tree - GeeksforGeeks](#)
- [Diameter of a Binary Tree | Algorithms](#)

Construct Binary Tree from Preorder and Inorder Traversal

Source

- leetcode: Construct Binary Tree from Preorder and Inorder Traversal | LeetCode OJ
- lintcode: (73) Construct Binary Tree from Preorder and Inorder Traversal

Given preorder and inorder traversal of a tree, construct the binary tree.

Example

Given in-order [1,2,3] and pre-order [2,1,3], return a tree:

```
2
 / \
1   3
```

Note

You may assume that duplicates do not exist in the tree.

1. preorder
2. inorder

inorder

Java

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     *@param preorder : A list of integers that preorder traversal of a tree
     *@param inorder : A list of integers that inorder traversal of a tree
     *@return : Root of a tree
     */
    public TreeNode buildTree(int[] preorder, int[] inorder) {
```

```

if (preorder == null || inorder == null) return null;
if (preorder.length == 0 || inorder.length == 0) return null;
if (preorder.length != inorder.length) return null;

TreeNode root = helper(preorder, 0, preorder.length - 1,
                      inorder, 0, inorder.length - 1);
return root;
}

private TreeNode helper(int[] preorder, int prestart, int preend,
                      int[] inorder, int instart, int inend) {
    // corner cases
    if (prestart > preend || instart > inend) return null;
    // build root TreeNode
    int root_val = preorder[prestart];
    TreeNode root = new TreeNode(root_val);
    // find index of root_val in inorder[]
    int index = findIndex(inorder, instart, inend, root_val);
    // build left subtree
    root.left = helper(preorder, prestart + 1, prestart + index - instart,
                       inorder, instart, index - 1);
    // build right subtree
    root.right = helper(preorder, prestart + index - instart + 1, preend,
                        inorder, index + 1, inend);
    return root;
}

private int findIndex(int[] nums, int start, int end, int target) {
    for (int i = start; i <= end; i++) {
        if (nums[i] == target) return i;
    }
    return -1;
}
}

```

`findIndex` $O(n)$, `helper` $O(n^2)$. $O(1)$.

Reference

- [Construct Binary Tree from Preorder and Inorder Traversal Java/C++/Python](#)

Construct Binary Tree from Inorder and Postorder Traversal

Source

- lintcode: (72) Construct Binary Tree from Inorder and Postorder Traversal

Given inorder and postorder traversal of a tree, construct the binary tree.

Example

Given inorder [1,2,3] and postorder [1,3,2], return a tree:

2

/ \

1 3

Note

You may assume that duplicates do not exist in the tree.

[Construct Binary Tree from Preorder and Inorder Traversal](#)

Java

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     *@param inorder : A list of integers that inorder traversal of a tree
     *@param postorder : A list of integers that postorder traversal of a tree
     *@return : Root of a tree
     */
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        if (inorder == null || postorder == null) return null;
        if (inorder.length == 0 || postorder.length == 0) return null;
        if (inorder.length != postorder.length) return null;

        TreeNode root = helper(inorder, 0, inorder.length - 1,
                               postorder, 0, postorder.length - 1);
        return root;
    }

    private TreeNode helper(int[] inorder, int inStart, int inEnd,
                           int[] postorder, int postStart, int postEnd) {
        if (inStart > inEnd || postStart > postEnd) return null;
        if (inStart == inEnd) return new TreeNode(inorder[inStart]);
        if (postStart == postEnd) return new TreeNode(postorder[postStart]);

        int rootVal = postorder[postEnd];
        int index = inStart;
        while (inorder[index] != rootVal) index++;

        TreeNode root = new TreeNode(rootVal);
        root.left = helper(inorder, inStart, index - 1,
                           postorder, postStart, postStart + index - inStart - 1);
        root.right = helper(inorder, index + 1, inEnd,
                            postorder, postStart + index - inStart, postEnd - 1);
        return root;
    }
}
```

```

}

private TreeNode helper(int[] inorder, int instart, int inend,
                      int[] postorder, int poststart, int postend) {
    // corner cases
    if (instart > inend || poststart > postend) return null;

    // build root TreeNode
    int root_val = postorder[postend];
    TreeNode root = new TreeNode(root_val);
    // find index of root_val in inorder[]
    int index = findIndex(inorder, instart, inend, root_val);
    // build left subtree
    root.left = helper(inorder, instart, index - 1,
                        postorder, poststart, poststart + index - instart - 1);
    // build right subtree
    root.right = helper(inorder, index + 1, inend,
                        postorder, poststart + index - instart, postend - 1);
    return root;
}

private int findIndex(int[] nums, int start, int end, int target) {
    for (int i = start; i <= end; i++) {
        if (nums[i] == target) return i;
    }
    return -1;
}
}

```

$$O(n), \quad O(n^2).$$

Subtree

Source

- lintcode: [\(245\) Subtree](#)

You have two every large binary trees: T1, with millions of nodes, and T2, with hundreds of nodes. Create an algorithm to decide if T2 is a subtree of T1.

Example

T2 is a subtree of T1 in the following case:

```

      1           3
     / \         /
T1 = 2   3       T2 =  4
      |
      4
  
```

T2 isn't a subtree of T1 in the following case:

```

      1           3
     / \         \
T1 = 2   3       T2 =    4
      |
      4
  
```

Note

A tree T2 is a subtree of T1 if there exists a node n in T1 such that the subtree of n is identical to T2.

That is, if you cut off the tree at node n, the two trees would be identical.

T2 T1 T1 T2

Java

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param T1, T2: The roots of binary tree.
     * @return: True if T2 is a subtree of T1, or false.
     */
}
  
```

```

/*
public boolean isSubtree(TreeNode T1, TreeNode T2) {
    if (T2 == null) return true;
    if (T1 == null) return false;
    return identical(T1, T2) || isSubtree(T1.left, T2) || isSubtree(T1.right, T2);
}

private boolean identical(TreeNode T1, TreeNode T2) {
    if (T1 == null && T2 == null) return true;
    if (T1 == null || T2 == null) return false;
    if (T1.val != T2.val) return false;
    return identical(T1.left, T2.left) && identical(T1.right, T2.right);
}
}

```

trick null identical isSubtree identical .val null, identical

identical $O(n)$, $O(m)$, $O(mn)$.

Reference

- LintCode: Subtree

Binary Tree Zigzag Level Order Traversal

Source

- leetcode: [Binary Tree Zigzag Level Order Traversal Java/C++/Python](#)
- lintcode: [\(71\) Binary Tree Zigzag Level Order Traversal](#)

Given a binary tree, return the zigzag level order traversal of its nodes' values. (ie, from left to right, then right to left for the next level and alternate between).

Example

Given binary tree {3,9,20,#,#,15,7},

```

      3
     / \
    9  20
      / \
     15   7
  
```

return its zigzag level order traversal as:

```
[
  [3],
  [20,9],
  [15,7]
]
```

1 -

Java

```

/*
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: A list of lists of integer include
  
```

```

/*
     the zigzag level order traversal of its nodes' values
*/
public ArrayList<ArrayList<Integer>> zigzagLevelOrder(TreeNode root) {
    ArrayList<ArrayList<Integer>> result = new ArrayList<ArrayList<Integer>>();
    if (root == null) return result;

    boolean odd = true;
    Queue<TreeNode> q = new LinkedList<TreeNode>();
    q.offer(root);
    while (!q.isEmpty()) {
        // level traversal
        int qLen = q.size();
        ArrayList<Integer> level = new ArrayList<Integer>();
        for (int i = 0; i < qLen; i++) {
            TreeNode node = q.poll();
            level.add(node.val);
            if (node.left != null) q.offer(node.left);
            if (node.right != null) q.offer(node.right);
        }
        // add level order reverse for even
        if (odd) {
            result.add(level);
        } else {
            Collections.reverse(level);
            result.add(level);
        }
        // flip odd and even
        odd = !odd;
    }

    return result;
}
}

```

reverse $n/2$, $O(n)$. $n/2$, reverse $O(n/2)$.

$O(n)$, $O(n)$.

Reference

- [Binary Tree Zigzag Level Order Traversal Java/C++/Python](#)
- [Printing a Binary Tree in Zig Zag Level-Order | LeetCode](#)

Binary Tree Serialization

Source

- lintcode: (7) Binary Tree Serialization

Design an algorithm and write code to serialize and deserialize a binary tree.
Writing the tree to a file is called 'serialization'
and reading back from the file to reconstruct
the exact same binary tree is 'deserialization'.

There is no limit of how you deserialize or serialize a binary tree,
you only need to make sure you can serialize a binary tree to a string
and deserialize this string to the original structure.

Have you met this question in a real interview? Yes

Example

An example of testdata: Binary tree {3,9,20,#,#,15,7},
denote the following structure:

```
3
/ \
9  20
 / \
15  7
```

Our data serialization use bfs traversal.

This is just for when you got wrong answer and want to debug the input.

You can use other method to do serializaiton and deserialization.

Java

```
/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
class Solution {
    /**
     * This method will be invoked first, you should design your own algorithm
     * to serialize a binary tree which denote by a root node to a string which
     * can be easily serialized by your own "deserialize" method later.
     */
}
```

```

public String serialize(TreeNode root) {
    StringBuilder sb = new StringBuilder();
    if (root == null) return sb.toString();

    seriaHelper(root, sb);

    return sb.substring(0, sb.length() - 1);
}

private void seriaHelper(TreeNode root, StringBuilder sb) {
    if (root == null) {
        sb.append("#,");
    } else {
        sb.append(root.val).append(",");
        seriaHelper(root.left, sb);
        seriaHelper(root.right, sb);
    }
}

/**
 * This method will be invoked second, the argument data is what exactly
 * you serialized at method "serialize", that means the data is not given by
 * system, it's given by your own serialize method. So the format of data is
 * designed by yourself, and deserialize it here as you serialize it in
 * "serialize" method.
 */
public TreeNode deserialize(String data) {
    if (data == null || data.length() == 0) return null;

    StringTokenizer st = new StringTokenizer(data, ",");
    return deseriaHelper(st);
}

private TreeNode deseriaHelper(StringTokenizer st) {
    if (!st.hasMoreTokens()) return null;

    String val = st.nextToken();
    if (val.equals("#")) {
        return null;
    }

    TreeNode root = new TreeNode(Integer.parseInt(val));
    root.left = deseriaHelper(st);
    root.right = deseriaHelper(st);

    return root;
}
}

```

Java StringTokenizer

deseriaHelper

Reference

- [Serialize and Deserialize a Binary Tree \(pre order\).](#)
- [Serialization/Deserialization of a Binary Tree | LeetCode](#)

Binary Search Tree -

[Binary Search Trees](#)

Insert Node in a Binary Search Tree

Source

- lintcode: [\(85\) Insert Node in a Binary Search Tree](#)

Given a binary search tree and a new tree node, insert the node into the tree. You should

Example

Given binary search tree as follow:

```
    2
   /   \
  1     4
      /
     3
```

after Insert node 6, the tree should be:

```
    2
   /   \
  1     4
      /   \
     3     6
```

Challenge

Do it without recursion



(

1. / -
2. / -

```
root->right = node  root->left = node . root->right/left =
func(...)
```

C++ Recursion

```
/*
 * forked from http://www.jiuzhang.com/solutions/insert-node-in-binary-search-tree/
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root: The root of the binary search tree.
     * @param node: insert this node into the binary search tree
     * @return: The root of the new binary search tree.
     */
    TreeNode* insertNode(TreeNode* root, TreeNode* node) {
        if (NULL == root) {
            return node;
        }

        if (node->val <= root->val) {
            root->left = insertNode(root->left, node);
        } else {
            root->right = insertNode(root->right, node);
        }

        return root;
    }
};
```

Java Recursion

```
public class Solution {
    /**
     * @param root: The root of the binary search tree.
     * @param node: insert this node into the binary search tree
     * @return: The root of the new binary search tree.
     */
    public TreeNode insertNode(TreeNode root, TreeNode node) {
        if (root == null) {
            return node;
        }

        if (root.val > node.val) {
            root.left = insertNode(root.left, node);
        } else {
            root.right = insertNode(root.right, node);
        }

        return root;
    }
}
```

```

    }
}
```

C++

```

/**
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root: The root of the binary search tree.
     * @param node: insert this node into the binary search tree
     * @return: The root of the new binary search tree.
     */
    TreeNode* insertNode(TreeNode* root, TreeNode* node) {
        if (NULL == root) {
            return node;
        }

        TreeNode* tempNode = root;
        while (NULL != tempNode) {
            if (node->val <= tempNode->val) {
                if (NULL == tempNode->left) {
                    tempNode->left = node;
                    return root;
                }
                tempNode = tempNode->left;
            } else {
                if (NULL == tempNode->right) {
                    tempNode->right = node;
                    return root;
                }
                tempNode = tempNode->right;
            }
        }

        return root;
    }
};
```

```
NULL == tempNode->right NULL == tempNode->left node root
```

Java Iterative

```
public class Solution {  
    /**  
     * @param root: The root of the binary search tree.  
     * @param node: insert this node into the binary search tree  
     * @return: The root of the new binary search tree.  
     */  
    public TreeNode insertNode(TreeNode root, TreeNode node) {  
        // write your code here  
        if (root == null) return node;  
        if (node == null) return root;  
  
        TreeNode rootcopy = root;  
        while (root != null) {  
            if (root.val <= node.val && root.right == null) {  
                root.right = node;  
                break;  
            }  
            else if (root.val > node.val && root.left == null) {  
                root.left = node;  
                break;  
            }  
            else if (root.val <= node.val) root = root.right;  
            else root = root.left;  
        }  
        return rootcopy;  
    }  
}
```

Validate Binary Search Tree

Source

- lintcode: [\(95\) Validate Binary Search Tree](#)

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node's key.
 The right subtree of a node contains only nodes with keys greater than the node's key.
 Both the left and right subtrees must also be binary search trees.

Example

An example:

```

1
/
2   3
 \
 4
 \
 5

```

The above binary tree is serialized as "{1,2,3,#,#,4,#,#,5}".

1. / -

2. / -

— key key false , true , [10, 5, 15, #, #, 6, 20] case

...

Java

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */

```

```

public class Solution {
    /**
     * @param root: The root of binary tree.
     * @return: True if the binary tree is BST, or false
     */
    public boolean isValidBST(TreeNode root) {
        if (root == null) return true;

        return helper(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean helper(TreeNode root, long lower, long upper) {
        if (root == null) return true;
        // System.out.println("root.val = " + root.val + ", lower = " + lower + ", upper
        // left node value < root node value < right node value
        if (root.val >= upper || root.val <= lower) return false;
        boolean isLeftValidBST = helper(root.left, lower, root.val);
        boolean isRightValidBST = helper(root.right, root.val, upper);

        return isLeftValidBST && isRightValidBST;
    }
}

```

long BST `root.val > upper`.

$O(n)$, $O(1)$.

Reference

- LeetCode: Validate Binary Search Tree - Yu's Garden - - 4

Search Range in Binary Search Tree

Source

- lintcode: [\(11\) Search Range in Binary Search Tree](#)

Given two values k_1 and k_2 (where $k_1 < k_2$) and a root pointer to a Binary Search Tree. Find all the keys of tree in range k_1 to k_2 . i.e. print all x such that $k_1 \leq x \leq k_2$ and x is a key in the tree. Return all the keys in ascending order.

Example

For example, if $k_1 = 10$ and $k_2 = 22$, then your function should print 12, 20 and 22.



OJ

C++ In-order Recursion

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param root: The root of the binary search tree.
     * @param k1 and k2: range k1 to k2.
     * @return: Return all keys that k1<=key<=k2 in ascending order.
     */
    vector<int> searchRange(TreeNode* root, int k1, int k2) {

```

```

        vector<int> result;
        inorder_dfs(result, root, k1, k2);

        return result;
    }

private:
    void inorder_dfs(vector<int> &ret, TreeNode *root, int k1, int k2) {
        if (NULL == root) {
            return;
        }

        inorder_dfs(ret, root->left, k1, k2);
        if ((root->val >= k1) && (root->val <= k2)) {
            ret.push_back(root->val);
        }
        inorder_dfs(ret, root->right, k1, k2);
    }
};

```

inorder_dfs

```

void inorder_dfs(vector<int> &ret, TreeNode *root, int k1, int k2) {
    if (NULL == root) {
        return;
    }

    if ((NULL != root->left) && (root->val > k1)) {
        inorder_dfs(ret, root->left, k1, k2);
    } // cut-off for left sub tree

    if ((root->val >= k1) && (root->val <= k2)) {
        ret.push_back(root->val);
    } // add valid value

    if ((NULL != root->right) && (root->val < k2)) {
        inorder_dfs(ret, root->right, k1, k2);
    } // cut-off for right sub tree
}

```

k1 k2

Convert Sorted Array to Binary Search Tree

Source

- leetcode - Convert Sorted Array to Binary Search Tree | LeetCode OJ

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

-

key key key -1

key key key key, OJ

start end

C++

```
/*
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode *sortedArrayToBST(vector<int> &num) {
        if (num.empty()) {
            return NULL;
        }

        return middleNode(num, 0, num.size() - 1);
    }

private:
    TreeNode *middleNode(vector<int> &num, const int start, const int end) {
        if (start > end) {
            return NULL;
        }

        TreeNode *root = new TreeNode(num[start + (end - start) / 2]);
        root->left = middleNode(num, start, start + (end - start) / 2 - 1);
        root->right = middleNode(num, start + (end - start) / 2 + 1, end);

        return root;
    }
};
```

start + 1 < end

middleNode key $O(n)$.

Reference

- [Convert Sorted Array to Binary Search Tree |](#)

Convert Sorted List to Binary Search Tree

Source

- leetcode - Convert Sorted List to Binary Search Tree | LeetCode OJ
- lintcode - (106) Convert Sorted List to Binary Search Tree

Given a singly linked list where elements are sorted in ascending order, convert it to a height balanced BST.

-

[Convert Sorted Array to Binary Search Tree | Data Structure and Algorithm](#)
AC,

C++

```
/*
 * Definition of ListNode
 * class ListNode {
 * public:
 *     int val;
 *     ListNode *next;
 *     ListNode(int val) {
 *         this->val = val;
 *         this->next = NULL;
 *     }
 * }
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param head: The first node of linked list.
     * @return: a tree node
     */
    TreeNode *sortedListToBST(ListNode *head) {
        if (NULL == head) {
            return NULL;
        }
    }
}
```

```

    // get the size of List
    ListNode *node = head;
    int len = 0;
    while (NULL != node) {
        node = node->next;
        ++len;
    }

    return buildBSTHelper(head, len);
}

private:
    TreeNode *buildBSTHelper(ListNode *head, int length) {
        if (NULL == head || length <= 0) {
            return NULL;
        }

        // get the middle ListNode as root TreeNode
        ListNode *lnode = head;
        int count = 0;
        while (count < length / 2) {
            lnode = lnode->next;
            ++count;
        }

        TreeNode *root = new TreeNode(lnode->val);
        root->left = buildBSTHelper(head, length / 2);
        root->right = buildBSTHelper(lnode->next, length - 1 - length / 2);

        return root;
    }
};

```

- 1.
- 2.
3. buildBSTHelper

buildBSTHelper NULL NULL . length 1 , 1->2 , 1->2->3 .

buildBSTHelper length —— head length length 0 NULL .

count count

1. 10.
2. 201.
3. 31
4. 4... 12.

count length / 2 length / 2 + 1 , length / 2 count < length / 2 , count 0. count
lnode TreeNode

count length / 2 length - 1 - length / 2 .

```

length - 1 - length / 2 != length / 2 - 1

length / 2 - 1 ?    TERMSIG= 11 ()...
length / 2 - 1 length - 1 - length / 2 . 13          1 / 2 0:-(

```

$O(n)$. $O(n)$, $O(n)$.

```

class Solution {
public:
    TreeNode *sortedListToBST(ListNode *head) {
        int length = 0;
        ListNode *curr = head;
        while (curr != NULL) {
            curr = curr->next;
            ++length;
        }
        return helper(head, length);
    }
private:
    TreeNode *helper(ListNode *&pos, int length) {
        if (length <= 0) {
            return NULL;
        }

        TreeNode *left = helper(pos, length / 2);
        TreeNode *root = new TreeNode(pos->val); // the sequence cannot be changed!
                                                // this is important difference of the s
        pos = pos->next;
        root->left = left;
        root->right = helper(pos, length - length / 2 - 1);
        return root;
    }
};

```

1. helper
2. n
3. new root new left

O(nlogn) length

```
/* *
```

```

* Definition for ListNode.
* public class ListNode {
*     int val;
*     ListNode next;
*     ListNode(int val) {
*         this.val = val;
*         this.next = null;
*     }
* }
* Definition of TreeNode:
* public class TreeNode {
*     public int val;
*     public TreeNode left, right;
*     public TreeNode(int val) {
*         this.val = val;
*         this.left = this.right = null;
*     }
* }
*/
public class Solution {
    /**
     * @param head: The first node of linked list.
     * @return: a tree node
     */
    public TreeNode sortedListToBST(ListNode head) {
        if (head == null) {
            return null;
        }
        return helper(head);
    }

    private TreeNode helper(ListNode head) {
        if (head == null) {
            return null;
        }
        if (head.next == null) {
            return new TreeNode(head.val);
        }

        ListNode pre = null;
        ListNode slow = head, fast = head;

        while (fast != null && fast.next != null) {
            pre = slow;
            slow = slow.next;
            fast = fast.next.next;
        }
        pre.next = null;

        TreeNode root = new TreeNode(slow.val);
        TreeNode L = helper(head);
        TreeNode R = helper(slow.next);
        root.left = L;
        root.right = R;

        return root;
    }
}

```

1. length
2. $O(n \log n)$ $O(n)$

Reference

- [Convert Sorted List to Binary Search Tree |](#)

Binary Search Tree Iterator

Source

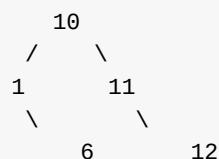
- lintcode: [\(86\) Binary Search Tree Iterator](#)

Design an iterator over a binary search tree with the following rules:

- Elements are visited in ascending order (i.e. an in-order traversal)
- next() and hasNext() queries run in O(1) time in average.

Example

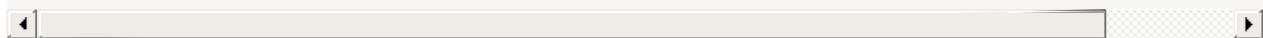
For the following binary search tree, in-order traversal by using iterator is [1, 6, 10,



Challenge

Extra memory usage $O(h)$, h is the height of the tree.

Super Star: Extra memory usage $O(1)$



Java

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 * Example of iterate a tree:
 * Solution iterator = new Solution(root);
 * while (iterator.hasNext()) {
 *     TreeNode node = iterator.next();
 *     do something for node
 * }
 */
public class Solution {

```

```
private Stack<TreeNode> stack = new Stack<>();
private TreeNode curt;

// @param root: The root of binary tree.
public Solution(TreeNode root) {
    curt = root;
}

//@return: True if there has next node, or false
public boolean hasNext() {
    return (curt != null || !stack.isEmpty()); //important to judge curt != null
}

//@return: return next node
public TreeNode next() {
    while (curt != null) {
        stack.push(curt);
        curt = curt.left;
    }

    curt = stack.pop();
    TreeNode node = curt;
    curt = curt.right;

    return node;
}
}
```

1. `hasNext()` `curt != null`
2. leetcode current

Exhaustive Search -

- 1.
- 2.
- 3.
4. ([DFS](#), Depth-First Search)
5. ([BFS](#), Breadth-First Search)

1, 2, 3

[DFS](#)

[DFS \(\)](#)

->()

- 1.
- 2.
- 3.

[BFS](#)

[BFS](#) ->->->... [BFS](#) $O(\text{states} \times \text{transfer_methods})$. [BFS](#)

Reference

- Chaper 2.1 p26 “”
- [Steven Skiena: Lecture15 - Backtracking](#)
- - -
- - -
- - Backtracking

Subsets -

Source

- leetcode: [Subsets | LeetCode OJ](#)
- lintcode: [\(17\) Subsets](#)

Given a set of distinct integers, return all possible subsets.

Note

Elements in a subset must be in non-descending order.

The solution set must not contain duplicate subsets.

Example

If S = [1,2,3], a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Combination [1, 2, 3]

DFS

1. [1] -> [1, 2] -> [1, 2, 3]
2. [2] -> [2, 3]
3. [3]

Python

```
class Solution:
    # @param {integer[]} nums
    # @return {integer[][]}
    def subsets(self, nums):
        if nums is None:
            return []
        result = []
        nums.sort()
```

```

        self.dfs(nums, 0, [], result)
        return result

def dfs(self, nums, pos, list_temp, ret):
    # append new object with []
    ret.append([] + list_temp)

    for i in xrange(pos, len(nums)):
        list_temp.append(nums[i])
        self.dfs(nums, i + 1, list_temp, ret)
        list_temp.pop()

```

C++

```

class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int> > result;
        if (nums.empty()) return result;

        sort(nums.begin(), nums.end());
        vector<int> list;
        dfs(nums, 0, list, result);

        return result;
    }

private:
    void dfs(vector<int>& nums, int pos, vector<int> &list,
             vector<vector<int> > &ret) {
        ret.push_back(list);

        for (int i = pos; i < nums.size(); ++i) {
            list.push_back(nums[i]);
            dfs(nums, i + 1, list, ret);
            list.pop_back();
        }
    }
};

```

Java

```

public class Solution {
    public List<List<Integer>> subsets(int[] nums) {
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        List<Integer> list = new ArrayList<Integer>();
        if (nums == null || nums.length == 0) {
            return result;
        }

        Arrays.sort(nums);
        dfs(nums, 0, list, result);

```

```

        return result;
    }

    private void dfs(int[] nums, int pos, List<Integer> list,
                     List<List<Integer>> ret) {

        // add temp result first
        ret.add(new ArrayList<Integer>(list));

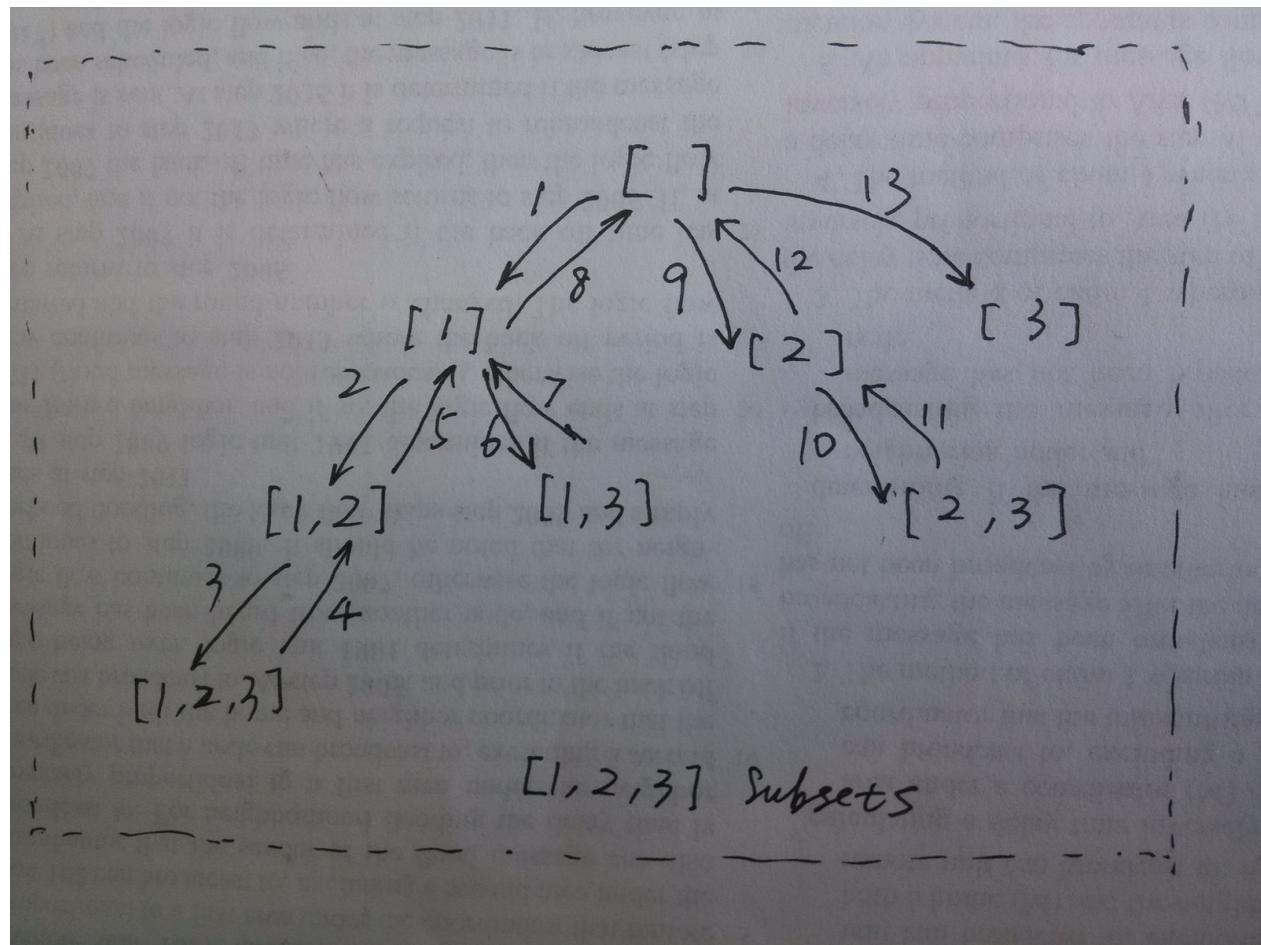
        for (int i = pos; i < nums.length; i++) {
            list.add(nums[i]);
            dfs(nums, i + 1, list, ret);
            list.remove(list.size() - 1);
        }
    }
}

```

Java Python list (Python [] +), list

Notice: backTrack(num, i + 1, list, ret);*i + 1*pos + 1 pos i subsets... :(

[1, 2, 3] list result list.add result.add list.remove



$O(n \log n)$. $O(2^n)$, $O(1)$, [1] -> [1, 2], $O(2^n)$.

list list $O(n)$.

Reference

- [\[NineChap 1.2\] Permutation - Woodstock Blog](#)
- [- subsets](#)
- [LeetCode: Subsets - Yu's Garden -](#)

Unique Subsets

Source

- lintcode: [\(18\) Unique Subsets](#)

Given a list of numbers that may has duplicate numbers, return all possible subsets

Note

Each element in a subset must be in non-descending order.

The ordering between two subsets is free.

The solution set must not contain duplicate subsets.

Example

If S = [1,2,2], a solution is:

```
[  
    [2],  
    [1],  
    [1,2,2],  
    [2,2],  
    [1,2],  
    []  
]
```

result.add list.add

n (n)()

[1, 2₁, 2₂] [], [1], [1, 2₁], [1, 2₁, 2₂], [1, 2₂], [2₁], [2₁, 2₂], [2₂]. [1, 2₂], [2₂].

Subsets for

pos i == pos i i != pos && s[i] :

C++

```
class Solution {  
public:  
    /**  
     * @param S: A set of numbers.  
     * @return: A list of lists. All valid subsets.  
     */  
    vector<vector<int>> subsetsWithDup(const vector<int> &S) {  
        vector<vector<int>> result;  
        if (S.empty()) {  
            return result;  
        }
```

```

vector<int> list;
vector<int> source(S);
sort(source.begin(), source.end());
backtrack(result, list, source, 0);

return result;
}

private:
void backtrack(vector<vector<int> > &ret, vector<int> &list,
               vector<int> &s, int pos) {

    ret.push_back(list);

    for (int i = pos; i != s.size(); ++i) {
        if (i != pos && s[i] == s[i - 1]) {
            continue;
        }
        list.push_back(s[i]);
        backtrack(ret, list, s, i + 1);
        list.pop_back();
    }
}
};


```

Reference

- [Subsets II](#) |

Unique Permutations

Source

- lintcode: [\(16\) Unique Permutations](#)

Given a list of numbers with duplicate number in it. Find all unique permutations.

Example

For numbers [1,2,2] the unique permutations are:

```
[  
    [1,2,2],  
    [2,1,2],  
    [2,2,1]  
]
```

Challenge

Do it without recursion.

[Unique Subsets](#)

[1, 2₁, 2₂] Permutations

1. [1, 2₁, 2₂]
2. [1, 2₂, 2₁]
3. [2₁, 1, 2₂]
4. [2₁, 2₂, 1]
5. [2₂, 1, 2₁]
6. [2₂, 2₁, 1]

```
1 2      5 3      6 4      22      21      [1, 21, 22]  [1, 22, 21]

[1, 21, 22]  list  [1, 21, 22],  list.pop_back()  list  [1],  list  22,  22
list          22  visited[1]  true ( 21)  22  visited[1]  false  list  [1, 21]
list.pop_back()  false

visited[i - 1] == false ,
```

C++

```

class Solution {
public:
    /**
     * @param nums: A list of integers.
     * @return: A list of unique permutations.
     */
    vector<vector<int>> permuteUnique(vector<int> &nums) {
        vector<vector<int>> ret;
        if (nums.empty()) {
            return ret;
        }

        // important! sort before call `backTrack`
        sort(nums.begin(), nums.end());
        vector<bool> visited(nums.size(), false);
        vector<int> list;
        backTrack(ret, list, visited, nums);

        return ret;
    }

private:
    void backTrack(vector<vector<int>> &result, vector<int> &list, \
                  vector<bool> &visited, vector<int> &nums) {
        if (list.size() == nums.size()) {
            result.push_back(list);
            // avoid unnecessary call for `for loop`, but not essential
            return;
        }

        for (int i = 0; i != nums.size(); ++i) {
            if (visited[i] || (i != 0 && nums[i] == nums[i - 1] \ 
                && !visited[i - 1])) {
                continue;
            }
            visited[i] = true;
            list.push_back(nums[i]);
            backTrack(result, list, visited, nums);
            list.pop_back();
            visited[i] = false;
        }
    }
};

```

Unique Subsets Unique Permutations

Unique Subsets Unique Permutations

Reference

- [Permutation II |](#)

Next Permutation

Source

- lintcode: [\(52\) Next Permutation](#)

Given a list of integers, which denote a permutation.

Find the next permutation in ascending order.

Example

For [1,3,2,3], the next permutation is [1,3,3,2]

For [4,3,2,1], the next permutation is [1,2,3,4]

Note

The list may contains duplicate integers.

C++ STL [Permutations](#)

1. `a[k] < a[k + 1]`,
2. `a[k] a[1]`, `a[k] < a[1]`.
3. `a[k] a[1]`.
4. `k + 1 ~ n`

`[4,3,2,1]` , `[1,2,3,4]` ,

Python

```
class Solution:
    # @param num : a list of integer
    # @return : a list of integer
    def nextPermutation(self, num):
        if num is None or len(num) <= 1:
            return num
        # step1: find num[i] < num[i + 1], Loop backwards
        i = 0
        for i in xrange(len(num) - 2, -1, -1):
            if num[i] < num[i + 1]:
                break
        elif i == 0:
            # reverse nums if reach maximum
            num = num[::-1]
            return num
        # step2: find num[i] < num[j], Loop backwards
        j = 0
```

```

for j in xrange(len(num) - 1, i, -1):
    if num[i] < num[j]:
        break
    # step3: swap between nums[i] and nums[j]
    num[i], num[j] = num[j], num[i]
    # step4: reverse between [i + 1, n - 1]
    num[i + 1:len(num)] = num[len(num) - 1:i:-1]

return num

```

C++

```

class Solution {
public:
    /**
     * @param nums: An array of integers
     * @return: An array of integers that's next permutation
     */
    vector<int> nextPermutation(vector<int> &nums) {
        if (nums.empty() || nums.size() <= 1) {
            return nums;
        }
        // step1: find nums[i] < nums[i + 1]
        int i = 0;
        for (i = nums.size() - 2; i >= 0; --i) {
            if (nums[i] < nums[i + 1]) {
                break;
            } else if (0 == i) {
                // reverse nums if reach maximum
                reverse(nums, 0, nums.size() - 1);
                return nums;
            }
        }
        // step2: find nums[i] < nums[j]
        int j = 0;
        for (j = nums.size() - 1; j > i; --j) {
            if (nums[i] < nums[j]) break;
        }
        // step3: swap between nums[i] and nums[j]
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
        // step4: reverse between [i + 1, n - 1]
        reverse(nums, i + 1, nums.size() - 1);

        return nums;
    }

private:
    void reverse(vector<int>& nums, int start, int end) {
        for (int i = start, j = end; i < j; ++i, --j) {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
    }
}

```

```
};
```

Java

```
public class Solution {
    /**
     * @param nums: an array of integers
     * @return: return nums in-place
     */
    public int[] nextPermutation(int[] nums) {
        if (nums == null || nums.length <= 1) {
            return nums;
        }
        // step1: find nums[i] < nums[i + 1]
        int i = 0;
        for (i = nums.length - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) {
                break;
            } else if (i == 0) {
                // reverse nums if reach maximum
                reverse(nums, 0, nums.length - 1);
                return nums;
            }
        }
        // step2: find nums[i] < nums[j]
        int j = 0;
        for (j = nums.length - 1; j > i; j--) {
            if (nums[i] < nums[j]) {
                break;
            }
        }
        // step3: swap between nums[i] and nums[j]
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
        // step4: reverse between [i + 1, n - 1]
        reverse(nums, i + 1, nums.length - 1);

        return nums;
    }

    private void reverse(int[] nums, int start, int end) {
        for (int i = start, j = end; i < j; i++, j--) {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
    }
}
```

Permutation step 1 $i == 0$ step1 step2

$O(n)$, temp $O(1)$.

Reference

- [Permutations](#)

Previous Permutation

Source

- lintcode: [\(51\) Previous Permutation](#)

Given a list of integers, which denote a permutation.

Find the previous permutation in ascending order.

Example

For [1,3,2,3], the previous permutation is [1,2,3,3]

For [1,2,3,4], the previous permutation is [4,3,2,1]

Note

The list may contains duplicate integers.

[Next Permutation](#)

1. $a[k] > a[k + 1]$,
2. $a[k] < a[1]$, $a[k] > a[1]$.
3. $a[k] < a[1]$.
4. $k + 1 \sim n$

Python

```
class Solution:
    # @param num : a list of integer
    # @return : a list of integer
    def previousPermutation(self, num):
        if num is None or len(num) <= 1:
            return num
        # step1: find num[i] > num[i + 1], Loop backwards
        i = 0
        for i in xrange(len(num) - 2, -1, -1):
            if num[i] > num[i + 1]:
                break
            elif i == 0:
                # reverse nums if reach maximum
                num = num[::-1]
                return num
        # step2: find num[i] > num[j], Loop backwards
        j = 0
        for j in xrange(len(num) - 1, i, -1):
```

```

        if num[i] > num[j]:
            break
    # step3: swap between nums[i] and nums[j]
    num[i], num[j] = num[j], num[i]
    # step4: reverse between [i + 1, n - 1]
    num[i + 1:len(num)] = num[len(num) - 1:i:-1]

    return num

```

C++

```

class Solution {
public:
    /**
     * @param nums: An array of integers
     * @return: An array of integers that's previous permutation
     */
    vector<int> previousPermutation(vector<int> &nums) {
        if (nums.empty() || nums.size() <= 1) {
            return nums;
        }
        // step1: find nums[i] > nums[i + 1]
        int i = 0;
        for (i = nums.size() - 2; i >= 0; --i) {
            if (nums[i] > nums[i + 1]) {
                break;
            } else if (0 == i) {
                // reverse nums if reach minimum
                reverse(nums, 0, nums.size() - 1);
                return nums;
            }
        }
        // step2: find nums[i] > nums[j]
        int j = 0;
        for (j = nums.size() - 1; j > i; --j) {
            if (nums[i] > nums[j]) break;
        }
        // step3: swap between nums[i] and nums[j]
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
        // step4: reverse between [i + 1, n - 1]
        reverse(nums, i + 1, nums.size() - 1);

        return nums;
    }

private:
    void reverse(vector<int>& nums, int start, int end) {
        for (int i = start, j = end; i < j; ++i, --j) {
            int temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
    }
};

```

Java

```

public class Solution {
    /**
     * @param nums: A list of integers
     * @return: A list of integers that's previous permuation
     */
    public ArrayList<Integer> previousPermutation(ArrayList<Integer> nums) {
        if (nums == null || nums.size() <= 1) {
            return nums;
        }
        // step1: find nums[i] > nums[i + 1]
        int i = 0;
        for (i = nums.size() - 2; i >= 0; i--) {
            if (nums.get(i) > nums.get(i + 1)) {
                break;
            } else if (i == 0) {
                // reverse nums if reach minimum
                reverse(nums, 0, nums.size() - 1);
                return nums;
            }
        }
        // step2: find nums[i] > nums[j]
        int j = 0;
        for (j = nums.size() - 1; j > i; j--) {
            if (nums.get(i) > nums.get(j)) {
                break;
            }
        }
        // step3: swap between nums[i] and nums[j]
        Collections.swap(nums, i, j);
        // step4: reverse between [i + 1, n - 1]
        reverse(nums, i + 1, nums.size() - 1);

        return nums;
    }

    private void reverse(List<Integer> nums, int start, int end) {
        for (int i = start, j = end; i < j; i++, j--) {
            Collections.swap(nums, i, j);
        }
    }
}

```

Permutation step 1 `i == 0` step1 step2

$O(n)$, temp $O(1)$.

Reference

- [Permutations](#)

Unique Binary Search Trees II

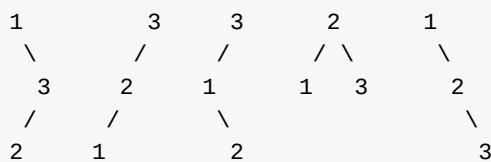
Source

- leetcode: Unique Binary Search Trees II | LeetCode OJ
- lintcode: (164) Unique Binary Search Trees II

Given n , generate all structurally unique BST's (binary search trees) that store values $1 \dots n$.

Example

Given $n = 3$, your program should return all 5 unique BST's shown below.



[Unique Binary Search Trees](#) i [1, i - 1][i + 1, n] 1n1n

```

i   i   i
      i i           i
                    start end .
i       start <= i <= end , start > end . start == end
  
```

```

helper(start, end) {
    result;
    if (start > end) {
        result.push_back(NULL);
        return;
    } else if (start == end) {
        result.push_back(TreeNode(i));
        return;
    }

    // dfs
    for (int i = start; i <= end; ++i) {
        leftTree = helper(start, i - 1);
        rightTree = helper(i + 1, end);
        // link left and right sub tree to the root i
        for (j in leftTree ){
            for (k in rightTree) {
                root = TreeNode(i);
                root->left = leftTree[j];
                root->right = rightTree[k];
                result.push_back(root);
            }
        }
    }
}
  
```

```

        }
    }

    return result;
}

```

[1, 2, 3]

1. helper(1,3)

- [leftTree]: helper(1, 0) ==> return NULL
- ---loop i = 2---
- [rightTree]: helper(2, 3)
 - i. [leftTree]: helper(2,1) ==> return NULL
 - ii. [rightTree]: helper(3,3) ==> return node(3)
 - iii. [for loop]: ==> return (2->3)
- ---loop i = 3---
 - i. [leftTree]: helper(2,2) ==> return node(2)
 - ii. [rightTree]: helper(4,3) ==> return NULL
 - iii. [for loop]: ==> return (3->2)

2. ...

```
start end      start == end      for :(
```

Python

```

"""
Definition of TreeNode:
class TreeNode:
    def __init__(self, val):
        this.val = val
        this.left, this.right = None, None
"""
class Solution:
    # @param n: An integer
    # @return: A list of root
    def generateTrees(self, n):
        return self.helper(1, n)

    def helper(self, start, end):
        result = []
        if start > end:
            result.append(None)
            return result

        for i in xrange(start, end + 1):
            # generate left and right sub tree
            leftTree = self.helper(start, i - 1)
            rightTree = self.helper(i + 1, end)
            # link left and right sub tree to root(i)
            for j in xrange(len(leftTree)):
                for k in xrange(len(rightTree)):
                    root = TreeNode(i)
                    root.left = leftTree[j]

```

```

        root.right = rightTree[k]
        result.append(root)

    return result
}

```

C++

```

/*
 * Definition of TreeNode:
 * class TreeNode {
 * public:
 *     int val;
 *     TreeNode *left, *right;
 *     TreeNode(int val) {
 *         this->val = val;
 *         this->left = this->right = NULL;
 *     }
 * }
 */
class Solution {
public:
    /**
     * @param n: An integer
     * @return: A list of root
     */
    vector<TreeNode *> generateTrees(int n) {
        return helper(1, n);
    }

private:
    vector<TreeNode *> helper(int start, int end) {
        vector<TreeNode *> result;
        if (start > end) {
            result.push_back(NULL);
            return result;
        }

        for (int i = start; i <= end; ++i) {
            // generate left and right sub tree
            vector<TreeNode *> leftTree = helper(start, i - 1);
            vector<TreeNode *> rightTree = helper(i + 1, end);
            // link left and right sub tree to root(i)
            for (int j = 0; j < leftTree.size(); ++j) {
                for (int k = 0; k < rightTree.size(); ++k) {
                    TreeNode *root = new TreeNode(i);
                    root->left = leftTree[j];
                    root->right = rightTree[k];
                    result.push_back(root);
                }
            }
        }

        return result;
    }
};


```

Java

```

/**
 * Definition of TreeNode:
 * public class TreeNode {
 *     public int val;
 *     public TreeNode left, right;
 *     public TreeNode(int val) {
 *         this.val = val;
 *         this.left = this.right = null;
 *     }
 * }
 */
public class Solution {
    /**
     * @param n: An integer
     * @return: A list of root
     */
    public List<TreeNode> generateTrees(int n) {
        return helper(1, n);
    }

    private List<TreeNode> helper(int start, int end) {
        List<TreeNode> result = new ArrayList<TreeNode>();
        if (start > end) {
            result.add(null);
            return result;
        }

        for (int i = start; i <= end; i++) {
            // generate left and right sub tree
            List<TreeNode> leftTree = helper(start, i - 1);
            List<TreeNode> rightTree = helper(i + 1, end);
            // link left and right sub tree to root(i)
            for (TreeNode lnode: leftTree) {
                for (TreeNode rnode: rightTree) {
                    TreeNode root = new TreeNode(i);
                    root.left = lnode;
                    root.right = rnode;
                    result.add(root);
                }
            }
        }

        return result;
    }
}

```

1. None/NULL/null.

2. start->end,

3. `for`

`DFS helper`

0 $O(1)$,

Reference

- [Code Ganker: Unique Binary Search Trees II -- LeetCode](#)
- [\[LeetCode\] Unique Binary Search Trees II, Solution](#)
- [Accepted Iterative Java solution. - Leetcode Discuss](#)
- [Unique Binary Search Trees II Java/C++/Python](#)

Permutation Index

Source

- lintcode: [\(197\) Permutation Index](#)

Given a permutation which contains no repeated number, find its index in all the permutations of these numbers, which are ordered in lexicographical order. The index begins at 1.

Example

Given [1,2,4], return 1.

next permutation

$O(n!)$, permutation (index), permutation

```
1, 2, 4      3!=6      [2, 4, 1] 1      2!=2 2      [2, 4, 1] 41, 2, 4312      2 * 1! =
2 10          [2, 4, 1], index 2! * (2 - 1) + 1! * (3 - 1) + 0! * (1 - 1) + 1 .
```

2142, index

Python

```
class Solution:
    # @param {int[]} A an integer array
    # @return {long} a long integer
    def permutationIndex(self, A):
        if A is None or len(A) == 0:
            return 0

        index = 1
        factor = 1
        for i in xrange(len(A) - 1, -1, -1):
            rank = 0
            for j in xrange(i + 1, len(A)):
                if A[i] > A[j]:
                    rank += 1

            index += rank * factor
            factor *= (len(A) - i)

        return index
```

C++

```
class Solution {
```

```

public:
    /**
     * @param A an integer array
     * @return a long integer
     */
    long long permutationIndex(vector<int>& A) {
        if (A.empty()) return 0;

        long long index = 1;
        long long factor = 1;
        for (int i = A.size() - 1; i >= 0; --i) {
            int rank = 0;
            for (int j = i + 1; j < A.size(); ++j) {
                if (A[i] > A[j]) ++rank;
            }
            index += rank * factor;
            factor *= (A.size() - i);
        }

        return index;
    }
};

```

Java

```

public class Solution {
    /**
     * @param A an integer array
     * @return a long integer
     */
    public long permutationIndex(int[] A) {
        if (A == null || A.length == 0) return 0;

        long index = 1;
        long factor = 1;
        for (int i = A.length - 1; i >= 0; i--) {
            int rank = 0;
            for (int j = i + 1; j < A.length; j++) {
                if (A[i] > A[j]) rank++;
            }
            index += rank * factor;
            factor *= (A.length - i);
        }

        return index;
    }
}

```

index factor rank index factor

for $O(n^2)$. $O(1)$.

Reference

- [Permutation Index](#)

Permutation Index II

Source

- lintcode: [\(198\) Permutation Index II](#)

Given a permutation which may contain repeated numbers, find its index in all the permutations of these numbers, which are ordered in lexicographical order. The index begins at 1.

Example

Given the permutation [1, 4, 2, 2], return 3.

Permutation Index

1!, 2!, 3!...

Python

```
class Solution:
    # @param {int[]} A an integer array
    # @return {long} a long integer
    def permutationIndexII(self, A):
        if A is None or len(A) == 0:
            return 0

        index = 1
        factor = 1
        for i in xrange(len(A) - 1, -1, -1):
            hash_map = {A[i]: 1}
            rank = 0
            for j in xrange(i + 1, len(A)):
                if A[j] in hash_map.keys():
                    hash_map[A[j]] += 1
                else:
                    hash_map[A[j]] = 1
                # get rank
                if A[i] > A[j]:
                    rank += 1

            index += rank * factor / self.dupPerm(hash_map)
            factor *= (len(A) - i)

        return index

    def dupPerm(self, hash_map):
        if hash_map is None or len(hash_map) == 0:
            return 0
        dup = 1
```

```

        for val in hash_map.values():
            dup *= self.factorial(val)

        return dup

    def factorial(self, n):
        r = 1
        for i in xrange(1, n + 1):
            r *= i

        return r

```

C++

```

class Solution {
public:
    /**
     * @param A an integer array
     * @return a long integer
     */
    long long permutationIndexII(vector<int>& A) {
        if (A.empty()) return 0;

        long long index = 1;
        long long factor = 1;
        for (int i = A.size() - 1; i >= 0; --i) {
            int rank = 0;
            unordered_map<int, int> hash;
            ++hash[A[i]];
            for (int j = i + 1; j < A.size(); ++j) {
                ++hash[A[j]];

                if (A[i] > A[j]) {
                    ++rank;
                }
            }
            index += rank * factor / dupPerm(hash);
            factor *= (A.size() - i);
        }

        return index;
    }

private:
    long long dupPerm(unordered_map<int, int> hash) {
        if (hash.empty()) return 1;

        long long dup = 1;
        for (auto it = hash.begin(); it != hash.end(); ++it) {
            dup *= fact(it->second);
        }

        return dup;
    }

    long long fact(int num) {

```

```

        long long val = 1;
        for (int i = 1; i <= num; ++i) {
            val *= i;
        }

        return val;
    }
};

```

Java

```

public class Solution {
    /**
     * @param A an integer array
     * @return a long integer
     */
    public long permutationIndexII(int[] A) {
        if (A == null || A.length == 0) return 0;

        long index = 1;
        long factor = 1;
        for (int i = A.length - 1; i >= 0; i--) {
            HashMap<Integer, Integer> hash = new HashMap<Integer, Integer>();
            hash.put(A[i], 1);
            int rank = 0;
            for (int j = i + 1; j < A.length; j++) {
                if (hash.containsKey(A[j])) {
                    hash.put(A[j], hash.get(A[j]) + 1);
                } else {
                    hash.put(A[j], 1);
                }

                if (A[i] > A[j]) {
                    rank++;
                }
            }
            index += rank * factor / dupPerm(hash);
            factor *= (A.length - i);
        }

        return index;
    }

    private long dupPerm(HashMap<Integer, Integer> hash) {
        if (hash == null || hash.isEmpty()) return 1;

        long dup = 1;
        for (int val : hash.values()) {
            dup *= fact(val);
        }

        return dup;
    }

    private long fact(int num) {
        long val = 1;
        for (int i = 1; i <= num; i++) {

```

```
        val *= i;
    }

    return val;
}

dup *= fact(val); , dup *= val; . A[i] - hash.put(A[i], 1); 2, 2, 1, 1
```

for $O(n^2)$, $O(n)$.

Permutation Sequence

Source

- leetcode: Permutation Sequence | LeetCode OJ
- lintcode: (388) Permutation Sequence

Given n and k , return the k -th permutation sequence.

Example

For $n = 3$, all permutations are listed as follows:

```
"123"
"132"
"213"
"231"
"312"
"321"
If k = 4, the fourth permutation is "231"
```

Note

n will be between 1 and 9 inclusive.

Challenge

$O(n^k)$ in time complexity is easy, can you do it in $O(n^2)$ or less?

Permutation Index

$1!, 2! \dots, n=3, k=4$

$k/(n-1)!, n=3, k=5$

$n=3, k=6$ $k \ 1$

$(k-1)!$

Python

```
class Solution:
    """
    @param n: n
    @param k: the k-th permutation
    @return: a string, the k-th permutation
    """
    def getPermutation(self, n, k):
        # generate factorial list
        factorial = [1]
        for i in xrange(1, n + 1):
            factorial.append(factorial[-1] * i)

        nums = range(1, n + 1)
        perm = []
        for i in xrange(n):
            rank = (k - 1) / factorial[n - i - 1]
```

```

        k = (k - 1) % factorial[n - i - 1] + 1
        # append and remove nums[rank]
        perm.append(nums[rank])
        nums.remove(nums[rank])
    # combine digits
    return "".join([str(digit) for digit in perm])

```

C++

```

class Solution {
public:
    /**
     * @param n: n
     * @param k: the kth permutation
     * @return: return the k-th permutation
     */
    string getPermutation(int n, int k) {
        // generate factorial list
        vector<int> factorial = vector<int>(n + 1, 1);
        for (int i = 1; i < n + 1; ++i) {
            factorial[i] = factorial[i - 1] * i;
        }
        // generate digits ranging from 1 to n
        vector<int> nums;
        for (int i = 1; i < n + 1; ++i) {
            nums.push_back(i);
        }

        vector<int> perm;
        for (int i = 0; i < n; ++i) {
            int rank = (k - 1) / factorial[n - i - 1];
            k = (k - 1) % factorial[n - i - 1] + 1;
            // append and remove nums[rank]
            perm.push_back(nums[rank]);
            nums.erase(std::remove(nums.begin(), nums.end(), nums[rank]), nums.end());
        }
        // transform a vector<int> to a string
        std::stringstream result;
        std::copy(perm.begin(), perm.end(), std::ostream_iterator<int>(result, ""));
        return result.str();
    }
};

```

Java

```

class Solution {
    /**
     * @param n: n
     * @param k: the kth permutation
     * @return: return the k-th permutation
     */
    public String getPermutation(int n, int k) {
        // generate factorial list

```

```

int[] factorial = new int[n + 1];
factorial[0] = 1;
for (int i = 1; i < n + 1; i++) {
    factorial[i] = factorial[i - 1] * i;
}
// generate digits ranging from 1 to n
ArrayList<Integer> nums = new ArrayList<Integer>(n);
for (int i = 0; i < n; i++) {
    nums.add(i + 1);
}

int[] perm = new int[n];
for (int i = 0; i < n; i++) {
    int rank = (k - 1) / factorial[n - i - 1];
    k = (k - 1) % factorial[n - i - 1] + 1;

    perm[i] = nums.get(rank);
    nums.remove(nums.get(rank));
}

StringBuilder result = new StringBuilder();
for (int digit : perm) {
    result.append(digit);
}
return result.toString();
}
}

```

- 1.
- 2.
- 3.

for $O(n)$, $n = O(n)$.

Reference

- [Permutation Sequence](#)
- [Permutation Sequence Java/C++/Python](#)
- [c++ - How to transform a vector into a string? - Stack Overflow](#)

Palindrome Partitioning

- tags: [palindrome]

Source

- leetcode: Palindrome Partitioning | LeetCode OJ
- lintcode: (136) Palindrome Partitioning

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example, given s = "aab",

Return

```
[  
    ["aa", "b"],  
    ["a", "a", "b"]  
]
```

1 - DFS

DFS. n n-1

$O(2^{n-1})$

$O(2^{n-1})$.

subsets

Python

```
class Solution:  
    # @param s, a string  
    # @return a list of lists of string  
    def partition(self, s):  
        result = []  
        if not s:  
            return result  
  
        palindromes = []  
        self.dfs(s, 0, palindromes, result)  
        return result  
  
    def dfs(self, s, pos, palindromes, ret):  
        if pos == len(s):  
            ret.append([] + palindromes)  
            return
```

```

        for i in xrange(pos + 1, len(s) + 1):
            if not self.isPalindrome(s[pos:i]):
                continue

            palindromes.append(s[pos:i])
            self.dfs(s, i, palindromes, ret)
            palindromes.pop()

    def isPalindrome(self, s):
        if not s:
            return False
        # reverse compare
        return s == s[::-1]

```

C++

```

class Solution {
public:
    /**
     * @param s: A string
     * @return: A list of lists of string
     */
    vector<vector<string>> partition(string s) {
        vector<vector<string>> result;
        if (s.empty()) return result;

        vector<string> palindromes;
        dfs(s, 0, palindromes, result);

        return result;
    }

private:
    void dfs(string s, int pos, vector<string> &palindromes,
             vector<vector<string>> &ret) {

        if (pos == s.size()) {
            ret.push_back(palindromes);
            return;
        }

        for (int i = pos + 1; i <= s.size(); ++i) {
            string substr = s.substr(pos, i - pos);
            if (!isPalindrome(substr)) {
                continue;
            }

            palindromes.push_back(substr);
            dfs(s, i, palindromes, ret);
            palindromes.pop_back();
        }
    }

    bool isPalindrome(string s) {
        if (s.empty()) return false;

        int n = s.size();

```

```

        for (int i = 0; i < n; ++i) {
            if (s[i] != s[n - i - 1]) return false;
        }

        return true;
    }
};

```

Java

```

public class Solution {
    /**
     * @param s: A string
     * @return: A list of lists of string
     */
    public List<List<String>> partition(String s) {
        List<List<String>> result = new ArrayList<List<String>>();
        if (s == null || s.isEmpty()) return result;

        List<String> palindromes = new ArrayList<String>();
        dfs(s, 0, palindromes, result);

        return result;
    }

    private void dfs(String s, int pos, List<String> palindromes,
                     List<List<String>> ret) {

        if (pos == s.length()) {
            ret.add(new ArrayList<String>(palindromes));
            return;
        }

        for (int i = pos + 1; i <= s.length(); i++) {
            String substr = s.substring(pos, i);
            if (!isPalindrome(substr)) {
                continue;
            }

            palindromes.add(substr);
            dfs(s, i, palindromes, ret);
            palindromes.remove(palindromes.size() - 1);
        }
    }

    private boolean isPalindrome(String s) {
        if (s == null || s.isEmpty()) return false;

        int n = s.length();
        for (int i = 0; i < n; i++) {
            if (s.charAt(i) != s.charAt(n - i - 1)) return false;
        }

        return true;
    }
}

```

Java ArrayList List Python result [], [] C++
n () Java

.substr(pos, n) pos

DFS $O(2^{n-1})$, $O(2^n)$, $O(n)$.

Reference

- [Palindrome Partitioning Java/C++/Python](#)
- [soulmachine Palindrome Partitioning](#)

Combinations

Source

- leetcode: Combinations | LeetCode OJ
- lintcode: (152) Combinations

```
Given two integers n and k,
return all possible combinations of k numbers out of 1 ... n.
```

Example

For example,

If n = 4 and k = 2, a solution is:

```
[[2,4],[3,4],[2,3],[1,2],[1,3],[1,4]]
```

Permutations

Java

```
public class Solution {
    /**
     * @param n: Given the range of numbers
     * @param k: Given the numbers of combinations
     * @return: All the combinations of k numbers out of 1..n
     */
    public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        List<Integer> list = new ArrayList<Integer>();
        if (n <= 0 || k <= 0) return result;

        helper(n, k, 1, list, result);
        return result;
    }

    private void helper(int n, int k, int pos,
                        List<Integer> list, List<List<Integer>> result) {

        if (list.size() == k) {
            result.add(new ArrayList<Integer>(list));
            return;
        }

        for (int i = pos; i <= n; i++) {
            list.add(i);
            helper(n, k, i + 1, list, result);
            list.remove(list.size() - 1);
        }
    }
}
```

```
}
```

```
helper(n, k, i + 1, list, result); i + 1 pos + 1
```

C_n^2 , $O(n^2)$. list $O(1)$.

Combination Sum

Source

- leetcode: [Combination Sum | LeetCode OJ](#)
- lintcode: [\(135\) Combination Sum](#)

Given a set of candidate numbers (C) and a target number (T),
 find all unique combinations in C where the candidate numbers sums to T.
 The same repeated number may be chosen from C unlimited number of times.

For example, given candidate set 2,3,6,7 and target 7,
 A solution set is:

[7]
 [2, 2, 3]

Have you met this question in a real interview? Yes

Example

given candidate set 2,3,6,7 and target 7,
 A solution set is:

[7]
 [2, 2, 3]

Note

- All numbers (including target) will be positive integers.
- Elements in a combination (a₁, a₂, ... , a_k) must be in non-descending order.
 (ie, a₁ ≤ a₂ ≤ ... ≤ a_k).
- The solution set must not contain duplicate combinations.

[Permutations](#) for

Java

```
public class Solution {
    /**
     * @param candidates: A list of integers
     * @param target:An integer
     * @return: A list of lists of integers
     */
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        List<Integer> list = new ArrayList<Integer>();
        if (candidates == null) return result;

        Arrays.sort(candidates);
        helper(candidates, 0, target, list, result);

        return result;
    }

    private void helper(int[] candidates, int index, int target, List<Integer> list, List<List<Integer>> result) {
        if (target < 0) return;
        if (target == 0) {
            result.add(new ArrayList<Integer>(list));
            return;
        }

        for (int i = index; i < candidates.length; i++) {
            list.add(candidates[i]);
            helper(candidates, i, target - candidates[i], list, result);
            list.remove(list.size() - 1);
        }
    }
}
```

```

}

private void helper(int[] candidates, int pos, int gap,
                   List<Integer> list, List<List<Integer>> result) {

    if (gap == 0) {
        // add new object for result
        result.add(new ArrayList<Integer>(list));
        return;
    }

    for (int i = pos; i < candidates.length; i++) {
        // cut invalid candidate
        if (gap < candidates[i]) {
            return;
        }
        list.add(candidates[i]);
        helper(candidates, i, gap - candidates[i], list, result);
        list.remove(list.size() - 1);
    }
}
}

```

target Soulmachine gap list result new

$O(n!)$, list $O(n)$.

Reference

- Soulmachine leetcode

Combination Sum II

Source

- leetcode: [Combination Sum II | LeetCode OJ](#)
- lintcode: [\(153\) Combination Sum II](#)

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T. Each number in C may only be used once in the combination.

Have you met this question in a real interview? Yes

Example

For example, given candidate set `[10,1,6,7,2,1,5]` and target `8`,

A solution set is:

`[1,7]`

`[1,2,5]`

`[2,6]`

`[1,1,6]`

Note

All numbers (including target) will be positive integers.

Elements in a combination (a_1, a_2, \dots, a_k) must be in non-descending order.
(ie, $a_1 \leq a_2 \leq \dots \leq a_k$).

The solution set must not contain duplicate combinations.

[Unique Subsets](#)

[Combination Sum](#)

Java

```
public class Solution {
    /**
     * @param num: Given the candidate numbers
     * @param target: Given the target number
     * @return: All the combinations that sum to target
     */
    public List<List<Integer>> combinationSum2(int[] num, int target) {
        List<List<Integer>> result = new ArrayList<List<Integer>>();
        List<Integer> list = new ArrayList<Integer>();
        if (num == null) return result;

        Arrays.sort(num);
        helper(num, 0, target, list, result);
    }

    private void helper(int[] num, int index, int target, List<Integer> list, List<List<Integer>> result) {
        if (target < 0) return;
        if (target == 0) {
            result.add(new ArrayList<Integer>(list));
            return;
        }

        for (int i = index; i < num.length; i++) {
            list.add(num[i]);
            helper(num, i + 1, target - num[i], list, result);
            list.remove(list.size() - 1);
        }
    }
}
```

```

        return result;
    }

private void helper(int[] nums, int pos, int gap,
                    List<Integer> list, List<List<Integer>> result) {

    if (gap == 0) {
        result.add(new ArrayList<Integer>(list));
        return;
    }

    for (int i = pos; i < nums.length; i++) {
        // ensure only the first same num is chosen, remove duplicate list
        if (i != pos && nums[i] == nums[i - 1]) {
            continue;
        }
        // cut invalid num
        if (gap < nums[i]) {
            return;
        }
        list.add(nums[i]);
        // i + 1 ==> only be used once
        helper(nums, i + 1, gap - nums[i], list, result);
        list.remove(list.size() - 1);
    }
}
}

```

Unique Subsets prev i + 1 .

$O(n)$, $O(n)$.

Dynamic Programming -

...

|

NOI

1. ()
2. ()

(0)

1. (State)
2. (Function)
3. (Initialization)
4. (Answer)

1. /
- 2.
3. ()
- 4.

(DP_Sequence)

i, , i...

1. State: $f[i]$ i//...
2. Function: $f[i] = f[i-1]$...
3. Initialization:
4. Answer: $f[n-1]$

(DP_Two_Sequence)

Distinct Subsequences

```
f[i+1][j+1] = f[i][j+1] + f[i][j] (if S[i] == T[j])
f[i+1][j+1] = f[i][j+1] (if S[i] != T[j])
```

$f[i+1][*]$ $f[i][*]$ $f[*][j]$ j, j $f[j+1]$ $f[j]$
 $(f[i+1][j])$ $f[i][j]$

1. $f[i+1][*]$ $f[i][*]$,
2. i, j

```
public class Solution {
    /**
     * @param S, T: Two string.
     * @return: Count the number of distinct subsequences
     */
    public int numDistinct(String S, String T) {
        if (S == null || T == null) return 0;
        if (S.length() < T.length()) return 0;
        if (T.length() == 0) return 1;

        int[] f = new int[T.length() + 1];
        f[0] = 1;
        for (int i = 0; i < S.length(); i++) {
            for (int j = T.length() - 1; j >= 0; j--) {
                if (S.charAt(i) == T.charAt(j)) {
                    f[j + 1] += f[j];
                }
            }
        }
        return f[T.length()];
    }
}
```

DP

Reference

1. -
2. - Topcoder

Triangle - Find the minimum path sum from top to bottom

Source

- lintcode: [\(109\) Triangle](#)

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

Note

Bonus point if you are able to do this using only $O(n)$ extra space, where n is the total number of rows in the triangle.

Example

For example, given the following triangle

```
[  
     [2],  
     [3,4],  
     [6,5,7],  
     [4,1,8,3]  
]
```

The minimum path sum from top to bottom is 11 (i.e., $2 + 3 + 5 + 1 = 11$).

triangle

Method 1 - Traverse without hashmap

$\text{---}^2 \quad O(2^n)$,

C++ Traverse without hashmap

```
class Solution {  
public:  
    /**  
     * @param triangle: a list of lists of integers.  
     * @return: An integer, minimum path sum.  
     */  
    int minimumTotal(vector<vector<int>> &triangle) {  
        if (triangle.empty()) {  
            return -1;  
        }  
  
        int result = INT_MAX;  
        dfs(0, 0, 0, triangle, result);  
  
        return result;  
    }  
}
```

```

private:
    void dfs(int x, int y, int sum, vector<vector<int>> &triangle, int &result) {
        const int n = triangle.size();
        if (x == n) {
            if (sum < result) {
                result = sum;
            }
            return;
        }

        dfs(x + 1, y, (sum + triangle[x][y]), triangle, result);
        dfs(x + 1, y + 1, (sum + triangle[x][y]), triangle, result);
    }
};

```

```
dfs() x == n x == n - 1 sum sum + triangle[x][y] yy+1xx+1x
```

nhashmap

Method 2 - Divide and Conquer without hashmap

C++ Divide and Conquer without hashmap

```

class Solution {
public:
    /**
     * @param triangle: a list of lists of integers.
     * @return: An integer, minimum path sum.
     */
    int minimumTotal(vector<vector<int>> &triangle) {
        if (triangle.empty()) {
            return -1;
        }

        int result = dfs(0, 0, triangle);

        return result;
    }

private:
    int dfs(int x, int y, vector<vector<int>> &triangle) {
        const int n = triangle.size();
        if (x == n) {
            return 0;
        }

        return min(dfs(x + 1, y, triangle), dfs(x + 1, y + 1, triangle)) + triangle[x][y];
    }
};

```



hashmaptriangle

Method 3 - Divide and Conquer with hashmap

trianglehashmap INT_MIN

C++ Divide and Conquer with hashmap

```

class Solution {
public:
    /**
     * @param triangle: a list of lists of integers.
     * @return: An integer, minimum path sum.
     */
    int minimumTotal(vector<vector<int>> &triangle) {
        if (triangle.empty()) {
            return -1;
        }

        vector<vector<int>> hashmap(triangle);
        for (int i = 0; i != hashmap.size(); ++i) {
            for (int j = 0; j != hashmap[i].size(); ++j) {
                hashmap[i][j] = INT_MIN;
            }
        }
        int result = dfs(0, 0, triangle, hashmap);

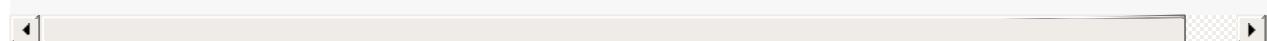
        return result;
    }

private:
    int dfs(int x, int y, vector<vector<int>> &triangle, vector<vector<int>> &hashmap)
    const int n = triangle.size();
    if (x == n) {
        return 0;
    }

    // INT_MIN means no value yet
    if (hashmap[x][y] != INT_MIN) {
        return hashmap[x][y];
    }
    int x1y = dfs(x + 1, y, triangle, hashmap);
    int x1y1 = dfs(x + 1, y + 1, triangle, hashmap);
    hashmap[x][y] = min(x1y, x1y1) + triangle[x][y];

    return hashmap[x][y];
}
};

```



$$O(2^n) \quad O(n^2), \quad 1 + 2 + \dots + n \approx O(n^2).$$

Method 4 - Dynamic Programming

$\text{triangle}[x][y] \rightarrow \text{triangle}[x+1][y]$
 $\text{triangle}[x][y] \rightarrow \text{triangle}[x+1][y+1]. \quad (x, y) \quad f(x, y)$

1. (x, y)
2. $(0, 0) \quad (x, y)$

$$f_1(x, y) = \min\{f_1(x+1, y), f_1(x+1, y+1)\} + \text{triangle}[x][y]$$

$$f_2(x, y) = \min\{f_2(x-1, y), f_2(x-1, y-1)\} + \text{triangle}[x][y]$$

$$f_1(n-1, y), 0 \leq y \leq n-1 \quad f_2(0, 0).$$

C++ From Bottom to Top

```
class Solution {
public:
    /**
     * @param triangle: a list of lists of integers.
     * @return: An integer, minimum path sum.
     */
    int minimumTotal(vector<vector<int>> &triangle) {
        if (triangle.empty()) {
            return -1;
        }

        vector<vector<int>> hashmap(triangle);

        // get the total row number of triangle
        const int N = triangle.size();
        for (int i = 0; i != N; ++i) {
            hashmap[N-1][i] = triangle[N-1][i];
        }

        for (int i = N - 2; i >= 0; --i) {
            for (int j = 0; j < i + 1; ++j) {
                hashmap[i][j] = min(hashmap[i + 1][j], hashmap[i + 1][j + 1]) + triangle[i][j];
            }
        }

        return hashmap[0][0];
    }
};
```

- 1.
2. hashmap

3. hashmap[N-1][i] ,
- 4.
5. hashmap[0][0]

trianglehashmaptriangle

C++ From Top to Bottom

```
class Solution {
public:
    /**
     * @param triangle: a list of lists of integers.
     * @return: An integer, minimum path sum.
     */
    int minimumTotal(vector<vector<int> > &triangle) {
        if (triangle.empty()) {
            return -1;
        }

        vector<vector<int> > hashmap(triangle);

        // get the total row number of triangle
        const int N = triangle.size();
        //hashmap[0][0] = triangle[0][0];
        for (int i = 0; i != N; ++i) {
            for (int j = 0; j <= i; ++j) {
                if (j == 0) {
                    hashmap[i][j] = hashmap[i - 1][j];
                }
                if (j == i) {
                    hashmap[i][j] = hashmap[i - 1][j - 1];
                }
                if ((j > 0) && (j < i)) {
                    hashmap[i][j] = min(hashmap[i - 1][j], hashmap[i - 1][j - 1]);
                }
                hashmap[i][j] += triangle[i][j];
            }
        }

        int result = INT_MAX;
        for (int i = 0; i != N; ++i) {
            result = min(result, hashmap[N - 1][i]);
        }
        return result;
    }
};
```

Backpack

Source

- lintcode: [\(92\) Backpack](#)

Given n items with size $A[i]$, an integer m denotes the size of a backpack. How full you can make the backpack?

Note
 You can not divide any item into small pieces.

Example
 If we have 4 items with size [2, 3, 5, 7], the backpack size is 11, we can select 2, 3 and 6.
 Your function should return the max size we can fill in the given backpack.

01 ###

1. : result[i][S] is
2. : $f[i][S] = f[i - 1][S - A[i]] \text{ or } f[i - 1][S]$ ($A[i] \leq S$)
 - o is
 - i. $f[i - 1][S - A[i]]: i - 1 \leq S - A[i]$
 - ii. $f[i - 1][S]: i - 1 \leq S$
3. : $f[1 \dots n][0] = true; f[0][1 \dots m] = false$. $1 \sim n$
4. : $f[n][S]$ trueS ($1 \leq S \leq m$)

C++ 2D vector for result

```
class Solution {
public:
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A: Given n items with size A[i]
     * @return: The maximum size
     */
    int backPack(int m, vector<int> A) {
        if (A.empty() || m < 1) {
            return 0;
        }

        const int N = A.size() + 1;
        const int M = m + 1;
        vector<vector<bool>> result;
        result.resize(N);
        for (vector<int>::size_type i = 0; i != N; ++i) {
```

```

        result[i].resize(M);
        std::fill(result[i].begin(), result[i].end(), false);
    }

    result[0][0] = true;
    for (int i = 1; i != N; ++i) {
        for (int j = 0; j != M; ++j) {
            if (j < A[i - 1]) {
                result[i][j] = result[i - 1][j];
            } else {
                result[i][j] = result[i - 1][j] || result[i - 1][j - A[i - 1]];
            }
        }
    }

    // return the largest i if true
    for (int i = M; i > 0; --i) {
        if (result[N - 1][i - 1]) {
            return (i - 1);
        }
    }
    return 0;
}
};

```

- 1.
2. **resize reserve**
3. **j < A[i - 1]**
4. **result[N - 1][i - 1] true**

```

for (int i = 1; i != N; ++i) {
    for (int j = 0; j != M; ++j) {
        result[i][j] = result[i - 1][j];
        if (j >= A[i - 1] && result[i - 1][j - A[i - 1]]) {
            result[i][j] = true;
        }
    }
}

```

——result resultii-1 **j result[i][S] result[i][S-A[i]]**

C++ 1D vector for result

```

class Solution {
public:
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A: Given n items with size A[i]
     */

```

```

    * @return: The maximum size
    */
int backPack(int m, vector<int> A) {
    if (A.empty() || m < 1) {
        return 0;
    }

    const int N = A.size();
    vector<bool> result;
    result.resize(m + 1);
    std::fill(result.begin(), result.end(), false);

    result[0] = true;
    for (int i = 0; i != N; ++i) {
        for (int j = m; j >= 0; --j) {
            if (j >= A[i] && result[j - A[i]]) {
                result[j] = true;
            }
        }
    }

    // return the largest i if true
    for (int i = m; i > 0; --i) {
        if (result[i]) {
            return i;
        }
    }
    return 0;
}

```

Backpack II

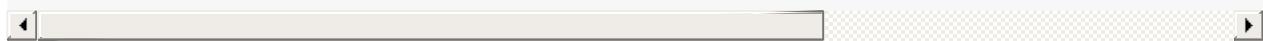
Source

- lintcode: [\(125\) Backpack II](#)

Given n items with size $A[i]$ and value $V[i]$, and a backpack with size m . What's the maximum value you can get?

Note
You cannot divide item into small pieces and the total size of items you choose should not exceed m .

Example
Given 4 items with size [2, 3, 5, 7] and value [1, 5, 2, 4], and a backpack with size 10.



$$K(i, w) \quad i \text{ size } w \quad K(i, w) = \max\{K(i - 1, w), K(i - 1, w - w_i) + v_i\}$$

C++ 2D vector for result

```

class Solution {
public:
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A & V: Given n items with size A[i] and value V[i]
     * @return: The maximum value
     */
    int backPackII(int m, vector<int> A, vector<int> V) {
        if (A.empty() || V.empty() || m < 1) {
            return 0;
        }
        const int N = A.size() + 1;
        const int M = m + 1;
        vector<vector<int>> result;
        result.resize(N);
        for (vector<int>::size_type i = 0; i != N; ++i) {
            result[i].resize(M);
            std::fill(result[i].begin(), result[i].end(), 0);
        }

        for (vector<int>::size_type i = 1; i != N; ++i) {
            for (int j = 0; j != M; ++j) {
                if (j < A[i - 1]) {
                    result[i][j] = result[i - 1][j];
                } else {
                    int temp = result[i - 1][j - A[i - 1]] + V[i - 1];
                    result[i][j] = max(temp, result[i - 1][j]);
                }
            }
        }
        return result[N - 1][M - 1];
    }
};

```

1. result
2. result——sizem

backpack result[j]

C++ 1D vector for result

```

class Solution {
public:
    /**
     * @param m: An integer m denotes the size of a backpack
     * @param A & V: Given n items with size A[i] and value V[i]
     * @return: The maximum value
     */

```

```

int backPackII(int m, vector<int> A, vector<int> V) {
    if (A.empty() || V.empty() || m < 1) {
        return 0;
    }

    const int M = m + 1;
    vector<int> result;
    result.resize(M);
    std::fill(result.begin(), result.end(), 0);

    for (vector<int>::size_type i = 0; i != A.size(); ++i) {
        for (int j = m; j >= 0; --j) {
            if (j < A[i]) {
                // result[j] = result[j];
            } else {
                int temp = result[j - A[i]] + V[i];
                result[j] = max(temp, result[j]);
            }
        }
    }

    return result[M - 1];
}

```

Reference

- [Lintcode: Backpack - neverlandly](#) -
- [Lintcode: Backpack II - neverlandly](#) -
- |
- [§ > 2.0 alpha1](#)

Minimum Path Sum

- tags: [DP_Matrix]

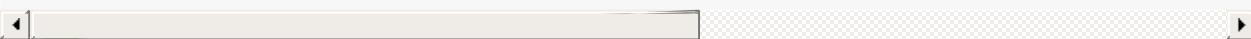
Source

- lintcode: [\(110\) Minimum Path Sum](#)

Given a $m \times n$ grid filled with non-negative numbers, find a path from top left to bottom right.

Note

You can only move either down or right at any point in time.



- State: $f[x][y]$ ($0,0$) (x,y)
- Function: $f[x][y] = (x, y) + \min\{f[x-1][y], f[x][y-1]\}$
- Initialization: $f[0][0] = A[0][0]$, $f[i][0] = \text{sum}(0,0 \rightarrow i,0)$, $f[0][i] = \text{sum}(0,0 \rightarrow 0,i)$
- Answer: $f[m-1][n-1]$

$f[m-1][n-1]f[m][n]$.

OJ [TLE](#) hashmap

C++ dfs without hashmap: Wrong answer

```
class Solution {
public:
    /**
     * @param grid: a list of lists of integers.
     * @return: An integer, minimizes the sum of all numbers along its path
     */
    int minPathSum(vector<vector<int>> &grid) {
        if (grid.empty()) {
            return 0;
        }

        const int m = grid.size() - 1;
        const int n = grid[0].size() - 1;

        return helper(grid, m, n);
    }

private:
    int helper(vector<vector<int>> &grid_in, int x, int y) {
        if (0 == x && 0 == y) {
            return grid_in[0][0];
        }
    }
}
```

```

    }
    if (0 == x) {
        return helper(grid_in, x, y - 1) + grid_in[x][y];
    }
    if (0 == y) {
        return helper(grid_in, x - 1, y) + grid_in[x][y];
    }

    return grid_in[x][y] + min(helper(grid_in, x - 1, y), helper(grid_in, x, y - 1));
}
};


```



C++ Iterative

```

class Solution {
public:
    /**
     * @param grid: a list of lists of integers.
     * @return: An integer, minimizes the sum of all numbers along its path
     */
    int minPathSum(vector<vector<int>> &grid) {
        if (grid.empty() || grid[0].empty()) {
            return 0;
        }

        const int M = grid.size();
        const int N = grid[0].size();
        vector<vector<int>> ret(M, vector<int>(N, 0));

        ret[0][0] = grid[0][0];
        for (int i = 1; i != M; ++i) {
            ret[i][0] = grid[i][0] + ret[i - 1][0];
        }
        for (int i = 1; i != N; ++i) {
            ret[0][i] = grid[0][i] + ret[0][i - 1];
        }

        for (int i = 1; i != M; ++i) {
            for (int j = 1; j != N; ++j) {
                ret[i][j] = grid[i][j] + min(ret[i - 1][j], ret[i][j - 1]);
            }
        }

        return ret[M - 1][N - 1];
    }
};


```

1. `gridgrid[0]`
2. `ret[0][0]ret[i-1]`

3. ij1
4. ret[M-1][N-1]0

: [LeetCode] Minimum Path Sum

$$f[x][y] = (x, y) + \min\{f[x-1][y], f[x][y-1]\} f[y]xy$$

C++ 1D vector

```
class Solution {
public:
    /**
     * @param grid: a list of lists of integers.
     * @return: An integer, minimizes the sum of all numbers along its path
     */
    int minPathSum(vector<vector<int> > &grid) {
        if (grid.empty() || grid[0].empty()) {
            return 0;
        }

        const int M = grid.size();
        const int N = grid[0].size();
        vector<int> ret(N, INT_MAX);

        ret[0] = 0;

        for (int i = 0; i != M; ++i) {
            ret[0] = ret[0] + grid[i][0];
            for (int j = 1; j != N; ++j) {
                ret[j] = grid[i][j] + min(ret[j], ret[j - 1]);
            }
        }

        return ret[N - 1];
    }
};
```

INT_MAX i = 0 ret[j] .

Unique Paths

- tags: [DP_Matrix]

Source

- lintcode: (114) Unique Paths

A robot is located at the top-left corner of a $m \times n$ grid (marked 'Start' in the diagram below).

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

Note

m and n will be at most 100.

$m \times n$

- State: $f[m][n]$ (m, n)
- Function: $f[m][n] = f[m-1][n] + f[m][n-1]$
- Initialization: $f[i][j] = 1, 0000$
- Answer: $f[m - 1][n - 1]$

C++

```
class Solution {
public:
    /**
     * @param n, m: positive integer (1 <= n ,m <= 100)
     * @return an integer
     */
    int uniquePaths(int m, int n) {
        if (m < 1 || n < 1) {
            return 0;
        }

        vector<vector<int>> ret(m, vector<int>(n, 1));

        for (int i = 1; i != m; ++i) {
            for (int j = 1; j != n; ++j) {
                ret[i][j] = ret[i - 1][j] + ret[i][j - 1];
            }
        }
    }
}
```

```
        return ret[m - 1][n - 1];
    }
};
```

- 1.
2. 1
- 3.
4. `ret[m - 1][n - 1]`

Unique Paths II

- tags: [DP_Matrix]

Source

- lintcode: ([115](#)) Unique Paths II

Follow up for "Unique Paths":

Now consider if some obstacles are added to the grids.
How many unique paths would there be?

An obstacle and empty space is marked as 1 and 0 respectively in the grid.
Note
 m and n will be at most 100.

Example

For example,

There is one obstacle in the middle of a 3x3 grid as illustrated below.

```
[  
    [0,0,0],  
    [0,1,0],  
    [0,0,0]  
]
```

The total number of unique paths is 2.

obstacal0

C++ initialization error

```
class Solution {  
public:  
    /**  
     * @param obstacleGrid: A list of lists of integers  
     * @return: An integer  
     */  
    int uniquePathsWithObstacles(vector<vector<int>> &obstacleGrid) {  
        if(obstacleGrid.empty() || obstacleGrid[0].empty()) {  
            return 0;  
        }  
  
        const int M = obstacleGrid.size();  
        const int N = obstacleGrid[0].size();  
  
        vector<vector<int>> ret(M, vector<int>(N, 0));
```

```

        for (int i = 0; i != M; ++i) {
            if (0 == obstacleGrid[i][0]) {
                ret[i][0] = 1;
            }
        }
        for (int i = 0; i != N; ++i) {
            if (0 == obstacleGrid[0][i]) {
                ret[0][i] = 1;
            }
        }
    }

    for (int i = 1; i != M; ++i) {
        for (int j = 1; j != N; ++j) {
            if (obstacleGrid[i][j]) {
                ret[i][j] = 0;
            } else {
                ret[i][j] = ret[i - 1][j] + ret[i][j - 1];
            }
        }
    }

    return ret[M - 1][N - 1];
}
};

```

000//0

C++

```

class Solution {
public:
    /**
     * @param obstacleGrid: A list of lists of integers
     * @return: An integer
     */
    int uniquePathsWithObstacles(vector<vector<int>> &obstacleGrid) {
        if(obstacleGrid.empty() || obstacleGrid[0].empty()) {
            return 0;
        }

        const int M = obstacleGrid.size();
        const int N = obstacleGrid[0].size();

        vector<vector<int>> ret(M, vector<int>(N, 0));

        for (int i = 0; i != M; ++i) {
            if (obstacleGrid[i][0]) {
                break;
            } else {
                ret[i][0] = 1;
            }
        }
        for (int i = 0; i != N; ++i) {
            if (obstacleGrid[0][i]) {

```

```
        break;
    } else {
        ret[0][i] = 1;
    }
}

for (int i = 1; i != M; ++i) {
    for (int j = 1; j != N; ++j) {
        if (obstacleGrid[i][j]) {
            ret[i][j] = 0;
        } else {
            ret[i][j] = ret[i - 1][j] + ret[i][j - 1];
        }
    }
}

return ret[M - 1][N - 1];
};

};
```

1.

2. (0) break

3.

4. ret[M - 1][N - 1]

Climbing Stairs

Source

- lintcode: [\(111\) Climbing Stairs](#)

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps.
In how many distinct ways can you climb to the top?

Example

Given an example n=3 , 1+1+1=2+1=1+2=3

return 3

- State: $f[i]$ i
- Function: $f[i] = f[i-1] + f[i-2]$
- Initialization: $f[0]=1, f[1]=1$
- Answer: $f[n]$

$f[i]f[i-1]f[i-1]f[i]f[i-2]f[i-2]f[i]1+12 f[i]=f[i-1]+f[i-2]+1f[i-2]f[i]f[i-1]f[i]=f[i-1]+f[i-2].$

C++

```
class Solution {
public:
    /**
     * @param n: An integer
     * @return: An integer
     */
    int climbStairs(int n) {
        if (n < 1) {
            return 0;
        }

        vector<int> ret(n + 1, 1);

        for (int i = 2; i != n + 1; ++i) {
            ret[i] = ret[i - 1] + ret[i - 2];
        }

        return ret[n];
    }
}
```

```

    }
};
```

- 1.
2. $n+1$
- 3.
4. $\text{ret}[n]$

$\text{ret}[0] = 10$

$O(n)$ $n+1$ climbing-stairs |

C++

```

class Solution {
public:
    /**
     * @param n: An integer
     * @return: An integer
     */
    int climbStairs(int n) {
        if (n < 1) {
            return 0;
        }

        int ret0 = 1, ret1 = 1, ret2 = 1;

        for (int i = 2; i != n + 1; ++i) {
            ret0 = ret1 + ret2;
            ret2 = ret1;
            ret1 = ret0;
        }

        return ret0;
    }
};
```

Jump Game

Source

- lintcode:

(116) Jump Game

Given an array of non-negative integers, you are initially positioned at the first index. Each element in the array represents your maximum jump length at that position. Determine if you are able to reach the last index.

Example

A = [2,3,1,1,4], return true.

A = [3,2,1,0,4], return false.



(-)

1. State: $f[i]$ i
2. Function: $f[i] = \text{OR } (f[j], j < i \wedge j + A[j] \geq i)$, $j \leq i$
3. Initialization: $f[0] = \text{true}$;
4. Answer: $N - 1$ $f[N-1]$

$$O(n) \quad 1 + 2 + \dots + n = O(n^2), \quad \text{TLE AC}$$

C++ from top to bottom

```
class Solution {
public:
    /**
     * @param A: A list of integers
     * @return: The boolean answer
     */
    bool canJump(vector<int> A) {
        if (A.empty()) {
            return true;
        }

        vector<bool> jumpto(A.size(), false);
        jumpto[0] = true;

        for (int i = 1; i != A.size(); ++i) {
            for (int j = i - 1; j >= 0; --j) {
                if (jumpto[j] && (j + A[j] >= i)) {

```

```

                jumpto[i] = true;
                break;
            }
        }

        return jumpto[A.size() - 1];
    }
};

```

(-)

i $f[i] = i + A[i]$ $i \leq f[i] \geq N - 1$, i, j $f[j] \geq i - 1$. 0.

1. State: $f[i] \quad i$
2. Function: $f[j] = j + A[j] \geq i, \quad j \quad i, \text{ true}$
3. Initialization: `true` `A.size() - 1`
4. Answer: 0 `true` `true`

C++ greedy, from bottom to top

```

class Solution {
public:
    /**
     * @param A: A list of integers
     * @return: The boolean answer
     */
    bool canJump(vector<int> A) {
        if (A.empty()) {
            return true;
        }

        int index_true = A.size() - 1;
        for (int i = A.size() - 1; i >= 0; --i) {
            if (i + A[i] >= index_true) {
                index_true = i;
            }
        }

        return 0 == index_true ? true : false;
    }
};

```

(-)

iA `A.size() - 1`

C++ greedy, from top to bottom

```
class Solution {
public:
    /**
     * @param A: A list of integers
     * @return: The boolean answer
     */
    bool canJump(vector<int> A) {
        if (A.empty()) {
            return true;
        }

        int farthest = A[0];

        for (int i = 1; i != A.size(); ++i) {
            if ((i <= farthest) && (i + A[i] > farthest)) {
                farthest = i + A[i];
            }
        }

        return farthest >= A.size() - 1;
    }
};
```

Word Break

- tags: [DP_Sequence]

Source

- leetcode: Word Break | LeetCode OJ
- lintcode: (107) Word Break

Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

For example, given

```
s = "leetcode",
dict = ["leet", "code"].
```

Return true because "leetcode" can be segmented as "leet code".

(DP_Sequence) DP

- State: $f[i]$ i
- Function: $f[i] = \text{or}\{f[j], j < i, \text{ letter in } [j+1, i] \text{ can be found in dict}\}$, $i \in f[j]$
 $j+1 \leq i \leq f[i]$
- Initialization: $f[0] = \text{true}$, $+1$
- Answer: $f[s.length]$

`s`

Python

```
class Solution:
    # @param s, a string
    # @param wordDict, a set<string>
    # @return a boolean
    def wordBreak(self, s, wordDict):
        if not s:
            return True
        if not wordDict:
            return False

        max_word_len = max([len(w) for w in wordDict])
        can_break = [True]
        for i in xrange(len(s)):
            can_break.append(False)
            for j in xrange(i, -1, -1):
                # optimize for too long interval
                if j - i + 1 > max_word_len:
                    break
                if s[j:i+1] in wordDict:
                    can_break[i+1] = True
                    break
```

```

        if i - j + 1 > max_word_len:
            break
        if can_break[j] and s[j:i + 1] in wordDict:
            can_break[i + 1] = True
            break
    return can_break[-1]

```

C++

```

class Solution {
public:
    bool wordBreak(string s, unordered_set<string>& wordDict) {
        if (s.empty()) return true;
        if (wordDict.empty()) return false;

        // get the max word length of wordDict
        int max_word_len = 0;
        for (unordered_set<string>::iterator it = wordDict.begin();
             it != wordDict.end(); ++it) {

            max_word_len = max(max_word_len, (*it).size());
        }

        vector<bool> can_break(s.size() + 1, false);
        can_break[0] = true;
        for (int i = 1; i <= s.size(); ++i) {
            for (int j = i - 1; j >= 0; --j) {
                // optimize for too long interval
                if (i - j > max_word_len) break;

                if (can_break[j] &&
                    wordDict.find(s.substr(j, i - j)) != wordDict.end()) {

                    can_break[i] = true;
                    break;
                }
            }
        }

        return can_break[s.size()];
    }
};

```

Java

```

public class Solution {
    public boolean wordBreak(String s, Set<String> wordDict) {
        if (s == null || s.length() == 0) return true;
        if (wordDict == null || wordDict.isEmpty()) return false;

        // get the max word length of wordDict
        int max_word_len = 0;
        for (String word : wordDict) {
            max_word_len = Math.max(max_word_len, word.length());
        }
    }
}

```

```

    }

    boolean[] can_break = new boolean[s.length() + 1];
    can_break[0] = true;
    for (int i = 1; i <= s.length(); i++) {
        for (int j = i - 1; j >= 0; j--) {
            // optimize for too long interval
            if (i - j > max_word_len) break;

            String word = s.substring(j, i);
            if (can_break[j] && wordDict.contains(word)) {
                can_break[i] = true;
                break;
            }
        }
    }

    return can_break[s.length()];
}
}

```

Python i C++ Java 1

1. $O(L_D \cdot L_w)$
2. $O(1)0$
3. for for $O(nL_w)$.
4. $O(nL_w)$.
5. word $O(n)$.

Longest Increasing Subsequence

- tags: [DP_Sequence]

Source

- lintcode: (76) Longest Increasing Subsequence
- Dynamic Programming | Set 3 (Longest Increasing Subsequence) - GeeksforGeeks

Given a sequence of integers, find the longest increasing subsequence (LIS). You code should return the length of the LIS.

Example

For [5, 4, 1, 2, 3], the LIS is [1, 2, 3], return 3

For [4, 2, 4, 5, 3, 7], the LIS is [4, 4, 5, 7], return 4

Challenge

Time complexity $O(n^2)$ or $O(n \log n)$

Clarification

What's the definition of longest increasing subsequence?

* The longest increasing subsequence problem is to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous, or unique.

* https://en.wikipedia.org/wiki/Longest_common_subsequence_problem

```
f[i] i LIC ... LIS ()           f[i] i i LIS      f[i] = {1 +
max{f[j]} where j < i, nums[j] < nums[i]} , j  f[j]  f[i] ,    f[i] = 1 . f[i] = 1 +
max(f[j]) , if f[i] < 1 + f[j], f[i] = 1 + f[j] .  max(f[]) .
```

Python

```
class Solution:
    """
    @param nums: The integer array
    @return: The length of LIS (longest increasing subsequence)
    """
    def longestIncreasingSubsequence(self, nums):
        if not nums:
            return 0

        lis = [1] * len(nums)
        for i in xrange(1, len(nums)):
            for j in xrange(i):
```

```

        if nums[j] <= nums[i] and lis[i] < 1 + lis[j]:
            lis[i] = 1 + lis[j]
    return max(lis)

```

C++

```

class Solution {
public:
    /**
     * @param nums: The integer array
     * @return: The length of LIS (longest increasing subsequence)
     */
    int longestIncreasingSubsequence(vector<int> nums) {
        if (nums.empty()) return 0;

        int len = nums.size();
        vector<int> lis(len, 1);

        for (int i = 1; i < len; ++i) {
            for (int j = 0; j < i; ++j) {
                if (nums[j] <= nums[i] && (lis[i] < lis[j] + 1)) {
                    lis[i] = 1 + lis[j];
                }
            }
        }

        return *max_element(lis.begin(), lis.end());
    }
};

```

Java

```

public class Solution {
    /**
     * @param nums: The integer array
     * @return: The length of LIS (longest increasing subsequence)
     */
    public int longestIncreasingSubsequence(int[] nums) {
        if (nums == null || nums.length == 0) return 0;

        int[] lis = new int[nums.length];
        Arrays.fill(lis, 1);

        for (int i = 1; i < nums.length; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[j] <= nums[i] && (lis[i] < lis[j] + 1)) {
                    lis[i] = lis[j] + 1;
                }
            }
        }

        // get the max lis
        int max_lis = 0;
        for (int i = 0; i < lis.length; i++) {

```

```
    if (lis[i] > max_lis) {
        max_lis = lis[i];
    }
}

return max_lis;
}
```

1. 1
2. lis[i]
3. lis

nums $O(n)$. for $O(n^2)$, $O(n)$, $O(n^2)$.

Palindrome Partitioning II

- tags: [DP_Sequence]

Source

- leetcode: Palindrome Partitioning II | LeetCode OJ
- lintcode: (108) Palindrome Partitioning II

Given a string s, cut s into some substrings such that every substring is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

Example

For example, given s = "aab",

Return 1 since the palindrome partitioning ["aa", "b"] could be produced using 1 cut.

1 -

bug-free : (f[i] i (i) f[i] = min{1 + f[j]}, where j < i and substring [j, i] is palindrome , [j, i]

Python

```
class Solution:
    # @param s, a string
    # @return an integer
    def minCut(self, s):
        if not s:
            print 0

        cut = [i - 1 for i in xrange(1 + len(s))]

        for i in xrange(1 + len(s)):
            for j in xrange(i):
                # s[j:i] is palindrome
                if s[j:i] == s[j:i][::-1]:
                    cut[i] = min(cut[i], 1 + cut[j])
        return cut[-1]
```

1. s None 0
- 2.

```

3. [0, len(s) - 1], j i i           1 + len(s)
4. [::-1]
5.

```

$1/2 \cdots n^2)$, $O(\text{len}(s))$, $O(n^3)$. s **TLE**. s $O(n)$.

2 -

PaMat[i][j] [i, j] , PaMat[i][j] = s[i] == s[j] && PaMat[i+1][j-1] , i
bug-free

Python

```

class Solution:
    # @param s, a string
    # @return an integer
    def minCut(self, s):
        if not s:
            print 0

        cut = [i - 1 for i in xrange(1 + len(s))]
        PaMatrix = self.getMat(s)

        for i in xrange(1 + len(s)):
            for j in xrange(i):
                if PaMatrix[j][i - 1]:
                    cut[i] = min(cut[i], cut[j] + 1)
        return cut[-1]

    def getMat(self, s):
        PaMat = [[True for i in xrange(len(s))] for j in xrange(len(s))]
        for i in xrange(len(s), -1, -1):
            for j in xrange(i, len(s)):
                if j == i:
                    PaMat[i][j] = True
                # not necessary if init with True
                # elif j == i + 1:
                #     PaMat[i][j] = s[i] == s[j]
                else:
                    PaMat[i][j] = s[i] == s[j] and PaMat[i + 1][j - 1]
        return PaMat

```

C++

```

class Solution {
public:

```

```

int minCut(string s) {
    if (s.empty()) return 0;

    int len = s.size();
    vector<int> cut;
    for (int i = 0; i < 1 + len; ++i) {
        cut.push_back(i - 1);
    }
    vector<vector<bool>> mat = getMat(s);

    for (int i = 1; i < 1 + len; ++i) {
        for (int j = 0; j < i; ++j) {
            if (mat[j][i - 1]) {
                cut[i] = min(cut[i], 1 + cut[j]);
            }
        }
    }

    return cut[len];
}

vector<vector<bool>> getMat(string s) {
    int len = s.size();
    vector<vector<bool>> mat = vector<vector<bool>>(len, vector<bool>(len, true));
    for (int i = len; i >= 0; --i) {
        for (int j = i; j < len; ++j) {
            if (j == i) {
                mat[i][j] = true;
            } else if (j == i + 1) {
                mat[i][j] = (s[i] == s[j]);
            } else {
                mat[i][j] = ((s[i] == s[j]) && mat[i + 1][j - 1]);
            }
        }
    }

    return mat;
}
};

```

Java

```

public class Solution {
    public int minCut(String s) {
        if (s == null || s.length() == 0) return 0;

        int len = s.length();
        int[] cut = new int[1 + len];
        for (int i = 0; i < 1 + len; ++i) {
            cut[i] = i - 1;
        }
        boolean[][] mat = paMat(s);

        for (int i = 1; i < 1 + len; i++) {
            for (int j = 0; j < i; j++) {
                if (mat[j][i - 1]) {

```

```

        cut[i] = Math.min(cut[i], 1 + cut[j]);
    }
}
}

return cut[len];
}

private boolean[][] paMat(String s) {
    int len = s.length();
    boolean[][] mat = new boolean[len][len];

    for (int i = len - 1; i >= 0; i--) {
        for (int j = i; j < len; j++) {
            if (j == i) {
                mat[i][j] = true;
            } else if (j == i + 1) {
                mat[i][j] = (s.charAt(i) == s.charAt(j));
            } else {
                mat[i][j] = (s.charAt(i) == s.charAt(j)) && mat[i + 1][j - 1];
            }
        }
    }

    return mat;
}
}

```

cut 1 + len(s) , cut[0] = -1 mat[i][j] == ... mat[i + 1][j - 1] j - 1 > i + 1 ,
getMat cut 1+len(s), for

for n $O(n^2)$, $O(n^2)$.

Reference

- [Palindrome Partitioning II Java/C++/Python](#)
- [soulmachine leetcode](#)

Longest Common Subsequence

- tags: [DP_Two_Sequence]

Source

- lintcode: (77) Longest Common Subsequence

Given two strings, find the longest common subsequence (LCS).

Your code should return the length of LCS.

Have you met this question in a real interview? Yes

Example

For "ABCD" and "EDCA", the LCS is "A" (or "D", "C"), return 1.

For "ABCD" and "EACB", the LCS is "AC", return 2.

Clarification

What's the definition of Longest Common Subsequence?

https://en.wikipedia.org/wiki/Longest_common_subsequence_problem

<http://baike.baidu.com/view/2020307.htm>

```
f[i][j] A    i B    j      ABCD EDCA    f 1  f[0][0] = 0 , f[0][*] = 0 , f[*]
[0] = 0 , f[lenA][lenB] . f(1)A B      f[1][1] = 1 + f[0][0] , f[1][1] =
max(f[0][1], f[1][0])

A[i] == B[j] , LCS 102          A[i] == B[j]  LCS 1.  A[i] != B[j]    A[i]
B[j]  LCS      f[i][j] = max(f[i - 1][j], f[i][j - 1]) . LCS
```

Python

```
class Solution:
    """
    @param A, B: Two strings.
    @return: The length of longest common subsequence of A and B.
    """
    def longestCommonSubsequence(self, A, B):
        if not A or not B:
            return 0

        lenA, lenB = len(A), len(B)
        lcs = [[0 for i in xrange(1 + lenA)] for j in xrange(1 + lenB)]
```

```

        for i in xrange(1, 1 + lenA):
            for j in xrange(1, 1 + lenB):
                if A[i - 1] == B[j - 1]:
                    lcs[i][j] = 1 + lcs[i - 1][j - 1]
                else:
                    lcs[i][j] = max(lcs[i - 1][j], lcs[i][j - 1])
        return lcs[lenA][lenB]
    
```

C++

```

class Solution {
public:
    /**
     * @param A, B: Two strings.
     * @return: The length of longest common subsequence of A and B.
     */
    int longestCommonSubsequence(string A, string B) {
        if (A.empty()) return 0;
        if (B.empty()) return 0;

        int lenA = A.size();
        int lenB = B.size();
        vector<vector<int>> lcs = \
            vector<vector<int>>(1 + lenA, vector<int>(1 + lenB));

        for (int i = 1; i < 1 + lenA; i++) {
            for (int j = 1; j < 1 + lenB; j++) {
                if (A[i - 1] == B[j - 1]) {
                    lcs[i][j] = 1 + lcs[i - 1][j - 1];
                } else {
                    lcs[i][j] = max(lcs[i - 1][j], lcs[i][j - 1]);
                }
            }
        }

        return lcs[lenA][lenB];
    }
};
    
```

Java

```

public class Solution {
    /**
     * @param A, B: Two strings.
     * @return: The length of longest common subsequence of A and B.
     */
    public int longestCommonSubsequence(String A, String B) {
        if (A == null || A.length() == 0) return 0;
        if (B == null || B.length() == 0) return 0;

        int lenA = A.length();
        int lenB = B.length();
        int[][] lcs = new int[1 + lenA][1 + lenB];
    }
}
    
```

```

        for (int i = 1; i < 1 + lenA; i++) {
            for (int j = 1; j < 1 + lenB; j++) {
                if (A.charAt(i - 1) == B.charAt(j - 1)) {
                    lcs[i][j] = 1 + lcs[i - 1][j - 1];
                } else {
                    lcs[i][j] = Math.max(lcs[i - 1][j], lcs[i][j - 1]);
                }
            }
        }

        return lcs[lenA][lenB];
    }
}

```

Python `[[0] * len(A)] * len(B)]`, Python Stackoverflow

for $O(lenA \times lenB)$, $O(lenA \times lenB)$.

Reference

- Stackoverflow. [Python multi-dimensional array initialization without a loop - Stack Overflow](#) ↵

Edit Distance

- tags: [DP_Two_Sequence]

Source

- leetcode: Edit Distance | LeetCode OJ
- lintcode: (119) Edit Distance

Given two words word1 and word2, find the minimum number of steps required to convert word1 to word2. (each operation is counted as 1 step.)

You have the following 3 operations permitted on a word:

Insert a character

Delete a character

Replace a character

Example

Given word1 = "mart" and word2 = "karma", return 3.

1 -

```
f[i][j] 1 i 2 j           f[0][j] = j, f[i][0] = i, f[i][j] f[i - 1][j - 1]
1] LCS          word1[i] == word2[j]

1. i == j , word1[i] == word2[j] , f[i - 1][j - 1] -> f[i][j]      f[i][j] = f[i - 1][j - 1].
2. i != j , / word1 word2          f[i][j] = 1 + min{f[i - 1][j], f[i][j - 1]}.

1. i == j , f[i][j] = 1 + f[i - 1][j - 1] .
2. i != j , / word1 word2          f[i][j] = 1 + min{f[i - 1][j], f[i][j - 1]} .

f[len(word1)][len(word2)]
```

Python

```
class Solution:
    # @param word1 & word2: Two string.
    # @return: The minimum number of steps.
    def minDistance(self, word1, word2):
        len1, len2 = 0, 0
        if word1:
            len1 = len(word1)
        if word2:
            len2 = len(word2)
```

```

if not word1 or not word2:
    return max(len1, len2)

f = [[i + j for i in xrange(1 + len2)] for j in xrange(1 + len1)]

for i in xrange(1, 1 + len1):
    for j in xrange(1, 1 + len2):
        if word1[i - 1] == word2[j - 1]:
            f[i][j] = min(f[i - 1][j - 1], 1 + f[i - 1][j], 1 + f[i][j - 1])
        else:
            f[i][j] = 1 + min(f[i - 1][j - 1], f[i - 1][j], f[i][j - 1])
return f[len1][len2]

```

C++

```

class Solution {
public:
    /**
     * @param word1 & word2: Two string.
     * @return: The minimum number of steps.
     */
    int fistance(string word1, string word2) {
        if (word1.empty() || word2.empty()) {
            return max(word1.size(), word2.size());
        }

        int len1 = word1.size();
        int len2 = word2.size();
        vector<vector<int>> f = \
            vector<vector<int>>(1 + len1, vector<int>(1 + len2, 0));
        for (int i = 0; i <= len1; ++i) {
            f[i][0] = i;
        }
        for (int i = 0; i <= len2; ++i) {
            f[0][i] = i;
        }

        for (int i = 1; i <= len1; ++i) {
            for (int j = 1; j <= len2; ++j) {
                if (word1[i - 1] == word2[j - 1]) {
                    f[i][j] = min(f[i - 1][j - 1], 1 + f[i - 1][j]);
                    f[i][j] = min(f[i][j], 1 + f[i][j - 1]);
                } else {
                    f[i][j] = min(f[i - 1][j - 1], f[i - 1][j]);
                    f[i][j] = 1 + min(f[i][j], f[i][j - 1]);
                }
            }
        }

        return f[len1][len2];
    }
};

```

Java

```

public class Solution {
    public int minDistance(String word1, String word2) {
        int len1 = 0, len2 = 0;
        if (word1 != null && word2 != null) {
            len1 = word1.length();
            len2 = word2.length();
        }
        if (word1 == null || word2 == null) {
            return Math.max(len1, len2);
        }

        int[][] f = new int[1 + len1][1 + len2];
        for (int i = 0; i <= len1; i++) {
            f[i][0] = i;
        }
        for (int i = 0; i <= len2; i++) {
            f[0][i] = i;
        }

        for (int i = 1; i <= len1; i++) {
            for (int j = 1; j <= len2; j++) {
                if (word1.charAt(i - 1) == word2.charAt(j - 1)) {
                    f[i][j] = Math.min(f[i - 1][j - 1], 1 + f[i - 1][j]);
                    f[i][j] = Math.min(f[i][j], 1 + f[i][j - 1]);
                } else {
                    f[i][j] = Math.min(f[i - 1][j - 1], f[i - 1][j]);
                    f[i][j] = 1 + Math.min(f[i][j], f[i][j - 1]);
                }
            }
        }
        return f[len1][len2];
    }
}

```

- 1.
2. (Python list len2 len1)
3. i, j 1 word1 word2
4. `f[len1][len2]`

for $O(\text{len1} \cdot \text{len2})$. $O(\text{len1} \cdot \text{len2})$.

Jump Game II

Source

- lintcode: [\(117\) Jump Game II](#)

Given an array of non-negative integers,
you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

Example

Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2.
(Jump 1 step from index 0 to 1, then 3 steps to the last index.)

(-)

A TLE $O(n^3)$.

- State: $f[i]$
- Function: $f[i] = \min(f[j]+1, j < i \&& j + A[j] \geq i)$
- Initialization: $f[0] = 0$
- Answer: $f[n-1]$

C++ Dynamic Programming

```
class Solution {
public:
    /**
     * @param A: A list of lists of integers
     * @return: An integer
     */
    int jump(vector<int> A) {
        if (A.empty()) {
            return -1;
        }

        const int N = A.size() - 1;
        vector<int> steps(N, INT_MAX);
        steps[0] = 0;

        for (int i = 1; i != N + 1; ++i) {
            for (int j = 0; j != i; ++j) {
                if ((steps[j] != INT_MAX) && (j + A[j] >= i)) {
                    steps[i] = steps[j] + 1;
                }
            }
        }
    }
}
```

```

                break;
            }
        }

        return steps[N];
    }
};

```

```

if ((steps[j] != INT_MAX) && (j + A[j] >= i)) {
    steps[i] = steps[j] + 1;
    break;
}

```

breakMINj

(-)

Jump Game

A min_index min_index end , min_index 0 jumps bug

C++ greedy from bottom to top, bug version

```

class Solution {
public:
    /**
     * @param A: A list of lists of integers
     * @return: An integer
     */
    int jump(vector<int> A) {
        if (A.empty()) {
            return -1;
        }

        const int N = A.size() - 1;
        int jumps = 0;
        int last_index = N;
        int min_index = N;

        for (int i = N - 1; i >= 0; --i) {
            if (i + A[i] >= last_index) {
                min_index = i;
            }

            if (0 == min_index) {
                return ++jumps;
            }
        }
    }
};

```

```

        if ((0 == i) && (min_index < last_index)) {
            ++jumps;
            last_index = min_index;
            i = last_index - 1;
        }
    }

    return jumps;
}
};

```

jumps last_index min_index

bug min_index 1 i = 0, for--i if (0 == min_index) 1

C++ greedy, from bottom to top

```

class Solution {
public:
    /**
     * @param A: A list of lists of integers
     * @return: An integer
     */
    int jump(vector<int> A) {
        if (A.empty()) {
            return 0;
        }

        const int N = A.size() - 1;
        int jumps = 0, end = N, min_index = N;

        while (end > 0) {
            for (int i = end - 1; i >= 0; --i) {
                if (i + A[i] >= end) {
                    min_index = i;
                }
            }

            if (min_index < end) {
                ++jumps;
                end = min_index;
            } else {
                // cannot jump to the end
                return -1;
            }
        }

        return jumps;
    }
};

```

bug version

min_index

```

for (int i = 0; i != end; ++i) {
    if (i + A[i] >= end) {
        min_index = i;
        break;
    }
}

```

(-)

```

farthest      start end      i+A[i] > farthest  farthest ()      end = farthest  end >=
A.size() - 1

```

C++

```


/*
 * http://www.jiuzhang.com/solutions/jump-game-ii/
 */
class Solution {
public:
    /**
     * @param A: A list of lists of integers
     * @return: An integer
     */
    int jump(vector<int> A) {
        if (A.empty()) {
            return 0;
        }

        const int N = A.size() - 1;
        int start = 0, end = 0, jumps = 0;

        while (end < N) {
            int farthest = end;
            for (int i = start; i <= end; ++i) {
                if (i + A[i] >= farthest) {
                    farthest = i + A[i];
                }
            }

            if (end < farthest) {
                ++jumps;
                start = end + 1;
                end = farthest;
            } else {
                // cannot jump to the end
                return -1;
            }
        }
    }
}

```

```
        return jumps;
    }
};
```

Best Time to Buy and Sell Stock

Source

- leetcode: Best Time to Buy and Sell Stock | LeetCode OJ
- lintcode: (149) Best Time to Buy and Sell Stock

Say you have an array for which the i th 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.

Example

Given an example [3,2,3,1,2], return 1

Python

```
class Solution:
    """
    @param prices: Given an integer array
    @return: Maximum profit
    """
    def maxProfit(self, prices):
        if prices is None or len(prices) <= 1:
            return 0

        profit = 0
        cur_price_min = 2**31 - 1
        for price in prices:
            profit = max(profit, price - cur_price_min)
            cur_price_min = min(cur_price_min, price)

        return profit
```

C++

```
class Solution {
public:
    /**
     * @param prices: Given an integer array
     * @return: Maximum profit
```

```

/*
int maxProfit(vector<int> &prices) {
    if (prices.size() <= 1) return 0;

    int profit = 0;
    int cur_price_min = INT_MAX;
    for (int i = 0; i < prices.size(); ++i) {
        profit = max(profit, prices[i] - cur_price_min);
        cur_price_min = min(cur_price_min, prices[i]);
    }

    return profit;
}
};

```

Java

```

public class Solution {
    /**
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    public int maxProfit(int[] prices) {
        if (prices == null || prices.length <= 1) return 0;

        int profit = 0;
        int curPriceMin = Integer.MAX_VALUE;
        for (int price : prices) {
            profit = Math.max(profit, price - curPriceMin);
            curPriceMin = Math.min(curPriceMin, price);
        }

        return profit;
    }
}

```

max min if

prices $O(n)$, $O(1)$.

Reference

- soulmachine

Best Time to Buy and Sell Stock II

Source

- leetcode: Best Time to Buy and Sell Stock II | LeetCode OJ
- lintcode: (150) Best Time to Buy and Sell Stock II

Say you have an array for which the i th element is the price of a given stock on day i .

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times). However, you may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Example

Given an example [2,1,2,0,1], return 2

~

—— soulmachine 0

Python

```
class Solution:
    """
    @param prices: Given an integer array
    @return: Maximum profit
    """
    def maxProfit(self, prices):
        if prices is None or len(prices) <= 1:
            return 0

        profit = 0
        for i in xrange(1, len(prices)):
            diff = prices[i] - prices[i - 1]
            if diff > 0:
                profit += diff

        return profit
```

C++

```
class Solution {
```

```

public:
    /**
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    int maxProfit(vector<int> &prices) {
        if (prices.size() <= 1) return 0;

        int profit = 0;
        for (int i = 1; i < prices.size(); ++i) {
            int diff = prices[i] - prices[i - 1];
            if (diff > 0) profit += diff;
        }

        return profit;
    }
};

```

Java

```

class Solution {
    /**
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    public int maxProfit(int[] prices) {
        if (prices == null || prices.length <= 1) return 0;

        int profit = 0;
        for (int i = 1; i < prices.length; i++) {
            int diff = prices[i] - prices[i - 1];
            if (diff > 0) profit += diff;
        }

        return profit;
    }
};

```

i 1

$O(n)$, $O(1)$.

Reference

- soulmachine

Best Time to Buy and Sell Stock III

Source

- leetcode: Best Time to Buy and Sell Stock III | LeetCode OJ
- lintcode: (151) Best Time to Buy and Sell Stock III

Say you have an array for which the i th element is the price of a given stock on day i .

Design an algorithm to find the maximum profit.
You may complete at most two transactions.

Example

Given an example [4,4,6,1,1,4,2,5], return 6.

Note

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

2

$O(n^2)$

[1,n] [1,i] [i+1,n], [1,i] [i+1,n] n n [1, i] Talk is
cheap, show me the code!

Python

```
class Solution:
    """
    @param prices: Given an integer array
    @return: Maximum profit
    """
    def maxProfit(self, prices):
        if prices is None or len(prices) <= 1:
            return 0

        n = len(prices)
        # get profit in the front of prices
        profit_front = [0] * n
        valley = prices[0]
        for i in xrange(1, n):
            profit_front[i] = max(profit_front[i - 1], prices[i] - valley)
            valley = min(valley, prices[i])
        # get profit in the back of prices, (i, n)
        profit_back = [0] * n
```

```

peak = prices[-1]
for i in xrange(n - 2, -1, -1):
    profit_back[i] = max(profit_back[i + 1], peak - prices[i])
    peak = max(peak, prices[i])
# add the profit front and back
profit = 0
for i in xrange(n):
    profit = max(profit, profit_front[i] + profit_back[i])

return profit

```

C++

```

class Solution {
public:
    /**
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    int maxProfit(vector<int> &prices) {
        if (prices.size() <= 1) return 0;

        int n = prices.size();
        // get profit in the front of prices
        vector<int> profit_front = vector<int>(n, 0);
        for (int i = 1, valley = prices[0]; i < n; ++i) {
            profit_front[i] = max(profit_front[i - 1], prices[i] - valley);
            valley = min(valley, prices[i]);
        }
        // get profit in the back of prices, (i, n)
        vector<int> profit_back = vector<int>(n, 0);
        for (int i = n - 2, peak = prices[n - 1]; i >= 0; --i) {
            profit_back[i] = max(profit_back[i + 1], peak - prices[i]);
            peak = max(peak, prices[i]);
        }
        // add the profit front and back
        int profit = 0;
        for (int i = 0; i < n; ++i) {
            profit = max(profit, profit_front[i] + profit_back[i]);
        }

        return profit;
    }
};

```

Java

```

class Solution {
    /**
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    public int maxProfit(int[] prices) {
        if (prices == null || prices.length <= 1) return 0;

```

```

// get profit in the front of prices
int[] profitFront = new int[prices.length];
profitFront[0] = 0;
for (int i = 1, valley = prices[0]; i < prices.length; i++) {
    profitFront[i] = Math.max(profitFront[i - 1], prices[i] - valley);
    valley = Math.min(valley, prices[i]);
}
// get profit in the back of prices, (i, n)
int[] profitBack = new int[prices.length];
profitBack[prices.length - 1] = 0;
for (int i = prices.length - 2, peak = prices[prices.length - 1]; i >= 0; i--) {
    profitBack[i] = Math.max(profitBack[i + 1], peak - prices[i]);
    peak = Math.max(peak, prices[i]);
}
// add the profit front and back
int profit = 0;
for (int i = 0; i < prices.length; i++) {
    profit = Math.max(profit, profitFront[i] + profitBack[i]);
}

return profit;
};


```

$O(n)$, $O(n)$.

Reference

- soulmachine

Best Time to Buy and Sell Stock IV

Source

- leetcode: Best Time to Buy and Sell Stock IV | LeetCode OJ
- lintcode: (393) Best Time to Buy and Sell Stock IV

Say you have an array for which the i th element is the price of a given stock on day i .

Design an algorithm to find the maximum profit. You may complete at most k transactions.

Example

Given $\text{prices} = [4, 4, 6, 1, 1, 4, 2, 5]$, and $k = 2$, return 6.

Note

You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Challenge

$O(nk)$ time.

1

3

TLE.

$k \leq n/2$ $k < n/2$

$f[i][j] \leftarrow k \times k-1 \times k+1 \times i$

```
f[i][j] = max(f[x][j - 1] + profit(x + 1, i))
```

01

Python

```
class Solution:
    """
    @param k: an integer
    @param prices: a list of integer
    @return: an integer which is maximum profit
    """
    def maxProfit(self, k, prices):
        if prices is None or len(prices) <= 1 or k <= 0:
            return 0
```

```

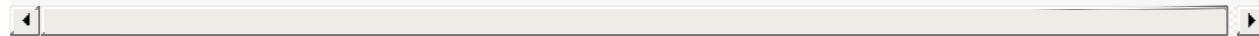
n = len(prices)
# k >= prices.length / 2 ==> multiple transactions Stock II
if k >= n / 2:
    profit_max = 0
    for i in xrange(1, n):
        diff = prices[i] - prices[i - 1]
        if diff > 0:
            profit_max += diff
    return profit_max

f = [[0 for i in xrange(k + 1)] for j in xrange(n + 1)]
for j in xrange(1, k + 1):
    for i in xrange(1, n + 1):
        for x in xrange(0, i + 1):
            f[i][j] = max(f[i][j], f[x][j - 1] + self.profit(prices, x + 1, i))

return f[n][k]

# calculate the profit of prices(l, u)
def profit(self, prices, l, u):
    if l >= u:
        return 0
    valley = 2**31 - 1
    profit_max = 0
    for price in prices[l - 1:u]:
        profit_max = max(profit_max, price - valley)
        valley = min(valley, price)
    return profit_max

```



C++

```

class Solution {
public:
    /**
     * @param k: An integer
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    int maxProfit(int k, vector<int> &prices) {
        if (prices.size() <= 1 || k <= 0) return 0;

        int n = prices.size();
        // k >= prices.length / 2 ==> multiple transactions Stock II
        if (k >= n / 2) {
            int profit_max = 0;
            for (int i = 1; i < n; ++i) {
                int diff = prices[i] - prices[i - 1];
                if (diff > 0) {
                    profit_max += diff;
                }
            }
            return profit_max;
        }

        vector<vector<int>> f = vector<vector<int>>(n + 1, vector<int>(k + 1, 0));
        for (int j = 1; j <= k; ++j) {

```

```

        for (int i = 1; i <= n; ++i) {
            for (int x = 0; x <= i; ++x) {
                f[i][j] = max(f[i][j], f[x][j - 1] + profit(prices, x + 1, i));
            }
        }
    }

    return f[n][k];
}

private:
    int profit(vector<int> &prices, int l, int u) {
        if (l >= u) return 0;

        int valley = INT_MAX;
        int profit_max = 0;
        for (int i = l - 1; i < u; ++i) {
            profit_max = max(profit_max, prices[i] - valley);
            valley = min(valley, prices[i]);
        }

        return profit_max;
    }
};

```

Java

```

class Solution {
    /**
     * @param k: An integer
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    public int maxProfit(int k, int[] prices) {
        if (prices == null || prices.length <= 1 || k <= 0) return 0;

        int n = prices.length;
        if (k >= n / 2) {
            int profit_max = 0;
            for (int i = 1; i < n; i++) {
                if (prices[i] - prices[i - 1] > 0) {
                    profit_max += prices[i] - prices[i - 1];
                }
            }
            return profit_max;
        }

        int[][] f = new int[n + 1][k + 1];
        for (int j = 1; j <= k; j++) {
            for (int i = 1; i <= n; i++) {
                for (int x = 0; x <= i; x++) {
                    f[i][j] = Math.max(f[i][j], f[x][j - 1] + profit(prices, x + 1, i));
                }
            }
        }

        return f[n][k];
    }
}

```

```

    }

    private int profit(int[] prices, int l, int u) {
        if (l >= u) return 0;

        int valley = Integer.MAX_VALUE;
        int profit_max = 0;
        for (int i = l + 1; i < u; i++) {
            profit_max = Math.max(profit_max, prices[i] - valley);
            valley = Math.min(valley, prices[i]);
        }
        return profit_max;
    }
}

```

Python `[[0] * k] * n]` , Python profit

$$O(n^2 \cdot k), f \quad O(n \cdot k).$$

Reference

- [LeetCode] Best Time to Buy and Sell Stock I II III IV |
- Best Time to Buy and Sell Stock IV Java/C++/Python
- leetcode-Best Time to Buy and Sell Stock //
- [LeetCode]Best Time to Buy and Sell Stock IV |

Distinct Subsequences

Source

- leetcode: Distinct Subsequences | LeetCode OJ
- lintcode: ([118](#)) Distinct Subsequences

Given a string S and a string T, count the number of distinct subsequences of T in S.
A subsequence of a string is a new string
which is formed from the original string by deleting some (can be none) of the characters
without disturbing the relative positions of the remaining characters.
(ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

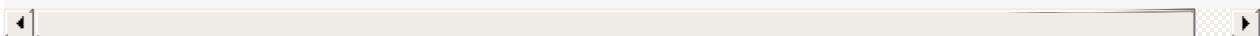
Example

Given S = "rabbbit", T = "rabbit", return 3.

Challenge

Do it in $O(n^2)$ time and $O(n)$ memory.

$O(n^2)$ memory is also acceptable if you do not know how to optimize memory.



1

subsequence substring subsequence S T S T S T

S

S T S S S T

Python

```
class Solution:
    # @param S, T: Two string.
    # @return: Count the number of distinct subsequences
    def numDistinct(self, S, T):
        if S is None or T is None:
            return 0
        if len(S) < len(T):
            return 0
        if len(T) == 0:
            return 1

        num = 0
        for i, Si in enumerate(S):
            if Si == T[0]:
                num += self.numDistinct(S[i + 1:], T[1:])

        return num
```

C++

```

class Solution {
public:
    /**
     * @param S, T: Two string.
     * @return: Count the number of distinct subsequences
     */
    int numDistinct(string &S, string &T) {
        if (S.size() < T.size()) return 0;
        if (T.empty()) return 1;

        int num = 0;
        for (int i = 0; i < S.size(); ++i) {
            if (S[i] == T[0]) {
                string Si = S.substr(i + 1);
                string t = T.substr(1);
                num += numDistinct(Si, t);
            }
        }
        return num;
    }
};

```

Java

```

public class Solution {
    /**
     * @param S, T: Two string.
     * @return: Count the number of distinct subsequences
     */
    public int numDistinct(String S, String T) {
        if (S == null || T == null) return 0;
        if (S.length() < T.length()) return 0;
        if (T.length() == 0) return 1;

        int num = 0;
        for (int i = 0; i < S.length(); i++) {
            if (S.charAt(i) == T.charAt(0)) {
                // T.length() >= 1, T.substring(1) will not throw index error
                num += numDistinct(S.substring(i + 1), T.substring(1));
            }
        }
        return num;
    }
}

```

1. null (C++ string NULL handle)
2. S T T S O

3. T 0 T S 1

```
for      T.length() >= 1 , T 1Java T      T.substring(1) ""
```

S T

 $O(n)$; S T

$$f(n) = \sum_{i=1}^{n-1} f(i), \text{ Fibonacci}$$

2 - Dynamic Programming

1 () , ([DP_Two_Sequence](#))

```
f[i][j] S[0:i] T[0:j] f[i-1][j], f[i-1][j-1], f[i][j-1] —— S[i] T[j]
```

1. $S[i] == T[j]$: $S[i] T[j] f[i][j] = f[i-1][j-1]; S[i] T[j] S[0:i-1] T[j]$
 $f[i][j] = f[i-1][j].$ $S[i] == T[j] f[i][j] = f[i-1][j-1] + f[i-1][j]$
2. $S[i] != T[j]$: $S[i] T[j] f[i][j] = f[i-1][j]$

($f[i][0]$ 1, 0. S T 01

Python

```
class Solution:
    # @param S, T: Two string.
    # @return: Count the number of distinct subsequences
    def numDistinct(self, S, T):
        if S is None or T is None:
            return 0
        if len(S) < len(T):
            return 0
        if len(T) == 0:
            return 1

        f = [[0 for i in xrange(len(T) + 1)] for j in xrange(len(S) + 1)]
        for i, Si in enumerate(S):
            f[i][0] = 1
            for j, Tj in enumerate(T):
                if Si == Tj:
                    f[i + 1][j + 1] = f[i][j + 1] + f[i][j]
                else:
                    f[i + 1][j + 1] = f[i][j + 1]

        return f[len(S)][len(T)]
```

C++

```
class Solution {
public:
    /**
     * @param S, T: Two string.
```

```

    * @return: Count the number of distinct subsequences
    */
int numDistinct(string &S, string &T) {
    if (S.size() < T.size()) return 0;
    if (T.empty()) return 1;

    vector<vector<int>> f(S.size() + 1, vector<int>(T.size() + 1, 0));
    for (int i = 0; i < S.size(); ++i) {
        f[i][0] = 1;
        for (int j = 0; j < T.size(); ++j) {
            if (S[i] == T[j]) {
                f[i + 1][j + 1] = f[i][j + 1] + f[i][j];
            } else {
                f[i + 1][j + 1] = f[i][j + 1];
            }
        }
    }

    return f[S.size()][T.size()];
}
};

```

Java

```

public class Solution {
    /**
     * @param S, T: Two string.
     * @return: Count the number of distinct subsequences
     */
    public int numDistinct(String S, String T) {
        if (S == null || T == null) return 0;
        if (S.length() < T.length()) return 0;
        if (T.length() == 0) return 1;

        int[][] f = new int[S.length() + 1][T.length() + 1];
        for (int i = 0; i < S.length(); i++) {
            f[i][0] = 1;
            for (int j = 0; j < T.length(); j++) {
                if (S.charAt(i) == T.charAt(j)) {
                    f[i + 1][j + 1] = f[i][j + 1] + f[i][j];
                } else {
                    f[i + 1][j + 1] = f[i][j + 1];
                }
            }
        }

        return f[S.length()][T.length()];
    }
}

```

for $O(n^2)$, $O(n^2)$. Dynamic Programming - .

Java

```
public class Solution {
    /**
     * @param S, T: Two string.
     * @return: Count the number of distinct subsequences
     */
    public int numDistinct(String S, String T) {
        if (S == null || T == null) return 0;
        if (S.length() < T.length()) return 0;
        if (T.length() == 0) return 1;

        int[] f = new int[T.length() + 1];
        f[0] = 1;
        for (int i = 0; i < S.length(); i++) {
            for (int j = T.length() - 1; j >= 0; j--) {
                if (S.charAt(i) == T.charAt(j)) {
                    f[j + 1] += f[j];
                }
            }
        }
        return f[T.length()];
    }
}
```

Reference

- [LeetCode: Distinct Subsequences - _](#)
- [soulmachine leetcode-cpp Distinct Subsequences](#)
- [Distinct Subsequences | Training dragons the hard way](#)

Interleaving String

Source

- leetcode: [Interleaving String | LeetCode OJ](#)
- lintcode: [\(29\) Interleaving String](#)

Given three strings: s1, s2, s3, determine whether s3 is formed by the interleaving of s1 and s2.

Example

For s1 = "aabcc", s2 = "dbbca"

When s3 = "aadbcbcac", return true.

When s3 = "aadbbaaccc", return false.

Challenge

O(n²) time or better

1 - bug

s3 s1 s2 s1 s2 s3 s1 s2 false. bug

Java

```
public class Solution {
    /**
     * Determine whether s3 is formed by interleaving of s1 and s2.
     * @param s1, s2, s3: As description.
     * @return: true or false.
     */
    public boolean isInterleave(String s1, String s2, String s3) {
        int len1 = (s1 == null) ? 0 : s1.length();
        int len2 = (s2 == null) ? 0 : s2.length();
        int len3 = (s3 == null) ? 0 : s3.length();

        if (len3 != len1 + len2) return false;

        int i1 = 0, i2 = 0;
        for (int i3 = 0; i3 < len3; i3++) {
            boolean result = false;
            if (i1 < len1 && s1.charAt(i1) == s3.charAt(i3)) {
                i1++;
                result = true;
                continue;
            }
            if (i2 < len2 && s2.charAt(i2) == s3.charAt(i3)) {
                i2++;
                result = true;
                continue;
            }
        }
    }
}
```

```

        // return instantly if both s1 and s2 can not pair with s3
        if (!result) return false;
    }

    return true;
}
}

```

s1, s2, s3 i1, i2, i3 null lintcode 15 test debug

s1[i1] s2[i2] s3[i3] s1 s2 s1 s1 s2 s3 true
false. OJ

bug (s1[i1] == s3[i3]) && (s2[i2] == s3[i3]) isInterleave, s1, s2, s3

s3, $O(n)$, $O(1)$.

2

```
(s1[i1] == s3[i3]) && (s2[i2] == s3[i3]) s1[i1] s3[i3] s2[i2] s3[i3]
s1[1+i1:], s2[i2], s3[1+i3:] s1[i1:], s2[1+i2], s3[1+i3:] . i1 i2.
```

Python

```

class Solution:
    """
    @params s1, s2, s3: Three strings as description.
    @return: return True if s3 is formed by the interleaving of
             s1 and s2 or False if not.
    @hint: you can use [[True] * m for i in range (n)] to allocate a n*m matrix.
    """
    def isInterleave(self, s1, s2, s3):
        len1 = 0 if s1 is None else len(s1)
        len2 = 0 if s2 is None else len(s2)
        len3 = 0 if s3 is None else len(s3)

        if len3 != len1 + len2:
            return False

        i1, i2 = 0, 0
        for i3 in xrange(len(s3)):
            result = False
            if (i1 < len1 and s1[i1] == s3[i3]) and \
               (i1 < len1 and s1[i1] == s3[i3]):
                # s1[1+i1:], s2[i2:], s3[1+i3:]
                case1 = self.isInterleave(s1[1+i1:], s2[i2:], s3[1+i3:])
                # s1[i1:], s2[1+i2:], s3[1+i3:]

```

```

        case2 = self.isInterleave(s1[i1:], s2[1 + i2:], s3[1 + i3:])
        return case1 or case2

    if i1 < len1 and s1[i1] == s3[i3]:
        i1 += 1
        result = True
        continue

    if i2 < len2 and s2[i2] == s3[i3]:
        i2 += 1
        result = True
        continue

    # return instantly if both s1 and s2 can not pair with s3
    if not result:
        return False

    return True

```

C++

```

class Solution {
public:
    /**
     * Determine whether s3 is formed by interleaving of s1 and s2.
     * @param s1, s2, s3: As description.
     * @return: true or false.
     */
    bool isInterleave(string s1, string s2, string s3) {
        int len1 = s1.size();
        int len2 = s2.size();
        int len3 = s3.size();

        if (len3 != len1 + len2) return false;

        int i1 = 0, i2 = 0;
        for (int i3 = 0; i3 < len3; ++i3) {
            bool result = false;
            if (i1 < len1 && s1[i1] == s3[i3] &&
                i2 < len2 && s2[i2] == s3[i3]) {
                // s1[1+i1:], s2[i2:], s3[1+i3:]
                bool case1 = isInterleave(s1.substr(1 + i1), s2.substr(i2), s3.substr(1 +
                // s1[i1:], s2[1+i2:], s3[1+i3:])
                bool case2 = isInterleave(s1.substr(i1), s2.substr(1 + i2), s3.substr(1 +
                // return instantly
                return case1 || case2;
            }

            if (i1 < len1 && s1[i1] == s3[i3]) {
                i1++;
                result = true;
                continue;
            }

            if (i2 < len2 && s2[i2] == s3[i3]) {
                i2++;
                result = true;
            }
        }
    }
}

```

```

        continue;
    }

    // return instantly if both s1 and s2 can not pair with s3
    if (!result) return false;
}

return true;
};


```

Java

```

public class Solution {
    /**
     * Determine whether s3 is formed by interleaving of s1 and s2.
     * @param s1, s2, s3: As description.
     * @return: true or false.
     */
    public boolean isInterleave(String s1, String s2, String s3) {
        int len1 = (s1 == null) ? 0 : s1.length();
        int len2 = (s2 == null) ? 0 : s2.length();
        int len3 = (s3 == null) ? 0 : s3.length();

        if (len3 != len1 + len2) return false;

        int i1 = 0, i2 = 0;
        for (int i3 = 0; i3 < len3; i3++) {
            boolean result = false;
            if (i1 < len1 && s1.charAt(i1) == s3.charAt(i3) &&
                i2 < len2 && s2.charAt(i2) == s3.charAt(i3)) {
                // s1[1+i1:], s2[i2:], s3[1+i3:]
                boolean case1 = isInterleave(s1.substring(1 + i1), s2.substring(i2), s3.s
                // s1[i1:], s2[1+i2:], s3[1+i3:]
                boolean case2 = isInterleave(s1.substring(i1), s2.substring(1 + i2), s3.s
                // return instantly
                return case1 || case2;
            }

            if (i1 < len1 && s1.charAt(i1) == s3.charAt(i3)) {
                i1++;
                result = true;
                continue;
            }

            if (i2 < len2 && s2.charAt(i2) == s3.charAt(i3)) {
                i2++;
                result = true;
                continue;
            }

            // return instantly if both s1 and s2 can not pair with s3
            if (!result) return false;
        }

        return true;
    }
}

```

```

    }
}
```

3 -

```

1 2 f[i1][i2][i3] s1i1 s2 i2 s3 i3 s1[i1], s2[i2], s3[i3]8
len3 == len1 +
len2 ,
f[i1][i2], s1  i1 s2      i2 s3      i1 + i2      s1[i1] == s3[i3]      s2[i2] == s3[i3]

f[i1][i2] = (s1[i1 - 1] == s3[i1 + i2 - 1] && f[i1 - 1][i2]) ||
            (s2[i2 - 1] == s3[i1 + i2 - 1] && f[i1][i2 - 1])

```

trick, `f[*][0]` `f[0][*]`.

Python

```

class Solution:
    """
    @params s1, s2, s3: Three strings as description.
    @return: return True if s3 is formed by the interleaving of
             s1 and s2 or False if not.
    @hint: you can use [[True] * m for i in range (n)] to allocate a n*m matrix.
    """
    def isInterleave(self, s1, s2, s3):
        len1 = 0 if s1 is None else len(s1)
        len2 = 0 if s2 is None else len(s2)
        len3 = 0 if s3 is None else len(s3)

        if len3 != len1 + len2:
            return False

        f = [[True] * (1 + len2) for i in xrange(1 + len1)]
        # s1[i1 - 1] == s3[i1 + i2 - 1] && f[i1 - 1][i2]
        for i in xrange(1, 1 + len1):
            f[i][0] = s1[i - 1] == s3[i - 1] and f[i - 1][0]
        # s2[i2 - 1] == s3[i1 + i2 - 1] && f[i1][i2 - 1]
        for i in xrange(1, 1 + len2):
            f[0][i] = s2[i - 1] == s3[i - 1] and f[0][i - 1]
        # i1 >= 1, i2 >= 1
        for i1 in xrange(1, 1 + len1):
            for i2 in xrange(1, 1 + len2):
                case1 = s1[i1 - 1] == s3[i1 + i2 - 1] and f[i1 - 1][i2]
                case2 = s2[i2 - 1] == s3[i1 + i2 - 1] and f[i1][i2 - 1]
                f[i1][i2] = case1 or case2

        return f[len1][len2]

```

C++

```

class Solution {
public:
    /**
     * Determine whether s3 is formed by interleaving of s1 and s2.
     * @param s1, s2, s3: As description.
     * @return: true or false.
     */
    bool isInterleave(string s1, string s2, string s3) {
        int len1 = s1.size();
        int len2 = s2.size();
        int len3 = s3.size();

        if (len3 != len1 + len2) return false;

        vector<vector<bool>> f(1 + len1, vector<bool>(1 + len2, true));
        // s1[i1 - 1] == s3[i1 + i2 - 1] && f[i1 - 1][i2]
        for (int i = 1; i <= len1; ++i) {
            f[i][0] = s1[i - 1] == s3[i - 1] && f[i - 1][0];
        }
        // s2[i2 - 1] == s3[i1 + i2 - 1] && f[i1][i2 - 1]
        for (int i = 1; i <= len2; ++i) {
            f[0][i] = s2[i - 1] == s3[i - 1] && f[0][i - 1];
        }
        // i1 >= 1, i2 >= 1
        for (int i1 = 1; i1 <= len1; ++i1) {
            for (int i2 = 1; i2 <= len2; ++i2) {
                bool case1 = s1[i1 - 1] == s3[i1 + i2 - 1] && f[i1 - 1][i2];
                bool case2 = s2[i2 - 1] == s3[i1 + i2 - 1] && f[i1][i2 - 1];
                f[i1][i2] = case1 || case2;
            }
        }
        return f[len1][len2];
    }
};

```

Java

```

public class Solution {
    /**
     * Determine whether s3 is formed by interleaving of s1 and s2.
     * @param s1, s2, s3: As description.
     * @return: true or false.
     */
    public boolean isInterleave(String s1, String s2, String s3) {
        int len1 = (s1 == null) ? 0 : s1.length();
        int len2 = (s2 == null) ? 0 : s2.length();
        int len3 = (s3 == null) ? 0 : s3.length();

        if (len3 != len1 + len2) return false;

        boolean [][] f = new boolean[1 + len1][1 + len2];
        f[0][0] = true;
        // s1[i1 - 1] == s3[i1 + i2 - 1] && f[i1 - 1][i2]
        for (int i = 1; i <= len1; i++) {
            f[i][0] = s1.charAt(i - 1) == s3.charAt(i - 1) && f[i - 1][0];
        }

```

```

    }
    // s2[i2 - 1] == s3[i1 + i2 - 1] && f[i1][i2 - 1]
    for (int i = 1; i <= len2; i++) {
        f[0][i] = s2.charAt(i - 1) == s3.charAt(i - 1) && f[0][i - 1];
    }
    // i1 >= 1, i2 >= 1
    for (int i1 = 1; i1 <= len1; i1++) {
        for (int i2 = 1; i2 <= len2; i2++) {
            boolean case1 = s1.charAt(i1 - 1) == s3.charAt(i1 + i2 - 1) && f[i1 - 1][
                boolean case2 = s2.charAt(i2 - 1) == s3.charAt(i1 + i2 - 1) && f[i1][i2 -
                f[i1][i2] = case1 || case2;
        }
    }
    return f[len1][len2];
}
}

```

1for ()

for $O(n^2)$, $O(n^2)$.

Reference

- soulmachine Interleaving String
- [Interleaving String Java/C++/Python](#)

Maximum Subarray

Source

- leetcode: Maximum Subarray | LeetCode OJ
- lintcode: (41) Maximum Subarray

Given an array of integers,
find a contiguous subarray which has the largest sum.

Example

Given the array [-2,2,-3,4,-1,2,1,-5,3],
the contiguous subarray [4,-1,2,1] has the largest sum = 6.

Note

The subarray should contain at least one number.

Challenge

Can you do it in time complexity $O(n)$?

1 -

n^2

sum	maxSum	sum ≤ 0	maxSum	sum , 0 sum
-----	--------	--------------	--------	-------------

Java

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @return: A integer indicate the sum of max subarray
     */
    public int maxSubArray(ArrayList<Integer> nums) {
        // -1 is not proper for illegal input
        if (nums == null || nums.isEmpty()) return -1;

        int sum = 0, maxSub = Integer.MIN_VALUE;
        for (int num : nums) {
            // drop negative sum
            sum = Math.max(sum, 0);
            sum += num;
            // update maxSub
            maxSub = Math.max(maxSub, sum);
        }

        return maxSub;
    }
}
```

```
sum maxSub
```

$O(n)$, $O(1)$.

2 - 1()

/ i

Java

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @return: A integer indicate the sum of max subarray
     */
    public int maxSubArray(ArrayList<Integer> nums) {
        // -1 is not proper for illegal input
        if (nums == null || nums.isEmpty()) return -1;

        int sum = 0, minSum = 0, maxSub = Integer.MIN_VALUE;
        for (int num : nums) {
            minSum = Math.min(minSum, sum);
            sum += num;
            maxSub = Math.max(maxSub, sum - minSum);
        }

        return maxSub;
    }
}
```

0

$O(n)$, $O(1)$.

3 - 2()

1

Java

```

public class Solution {
    /**
     * @param nums: A list of integers
     * @return: A integer indicate the sum of max subarray
     */
    public int maxSubArray(ArrayList<Integer> nums) {
        // -1 is not proper for illegal input
        if (nums == null || nums.isEmpty()) return -1;

        int size = nums.size();
        int[] local = new int[size];
        int[] global = new int[size];
        local[0] = nums.get(0);
        global[0] = nums.get(0);
        for (int i = 1; i < size; i++) {
            // drop local[i - 1] < 0
            local[i] = Math.max(nums.get(i), local[i - 1] + nums.get(i));
            // update global with local
            global[i] = Math.max(global[i - 1], local[i]);
        }

        return global[size - 1];
    }
}

```

0 local global

$O(n)$, $O(n)$.

Reference

- Offer
- [Maximum Subarray Java/C++/Python](#)

Maximum Subarray II

Source

- lintcode: [\(42\) Maximum Subarray II](#)

Given an array of integers,
find two non-overlapping subarrays which have the largest sum.

The number in each subarray should be contiguous.

Return the largest sum.

Example

For given [1, 3, -1, 2, -1, 2],
the two subarrays are [1, 3] and [2, -1, 2] or [1, 3, -1, 2] and [2],
they both have the largest sum 7.

Note

The subarray should contain at least one number

Challenge

Can you do it in time complexity $O(n)$?

[Maximum Subarray](#) ONO!!!

Java

```
public class Solution {
    /**
     * @param nums: A list of integers
     * @return: An integer denotes the sum of max two non-overlapping subarrays
     */
    public int maxTwoSubArrays(ArrayList<Integer> nums) {
        // -1 is not proper for illegal input
        if (nums == null || nums.isEmpty()) return -1;

        int size = nums.size();
        // get max sub array forward
        int[] maxSubArrayF = new int[size];
        forwardTraversal(nums, maxSubArrayF);
        // get max sub array backward
        int[] maxSubArrayB = new int[size];
        backwardTraversal(nums, maxSubArrayB);
        // get maximum subarray by iteration
        int maxTwoSub = Integer.MIN_VALUE;
        for (int i = 0; i < size - 1; i++) {
            // non-overlapping
            maxTwoSub = Math.max(maxTwoSub,
                maxSubArrayF[i] + maxSubArrayB[i + 1]);
        }
        return maxTwoSub;
    }

    private void forwardTraversal(List<Integer> nums, int[] maxSubArray) {
        int maxSum = Integer.MIN_VALUE;
        for (int num : nums) {
            maxSum = Math.max(maxSum, num);
            maxSubArray[nums.indexOf(num)] = maxSum;
        }
    }

    private void backwardTraversal(List<Integer> nums, int[] maxSubArray) {
        int maxSum = Integer.MIN_VALUE;
        for (int i = nums.size() - 1; i >= 0; i--) {
            maxSum = Math.max(maxSum, nums.get(i));
            maxSubArray[i] = maxSum;
        }
    }
}
```

```

        maxTwoSub = Math.max(maxTwoSub, maxSubArrayF[i] + maxSubArrayB[i + 1]);
    }

    return maxTwoSub;
}

private void forwardTraversal(List<Integer> nums, int[] maxSubArray) {
    int sum = 0, minSum = 0, maxSub = Integer.MIN_VALUE;
    int size = nums.size();
    for (int i = 0; i < size; i++) {
        minSum = Math.min(minSum, sum);
        sum += nums.get(i);
        maxSub = Math.max(maxSub, sum - minSum);
        maxSubArray[i] = maxSub;
    }
}

private void backwardTraversal(List<Integer> nums, int[] maxSubArray) {
    int sum = 0, minSum = 0, maxSub = Integer.MIN_VALUE;
    int size = nums.size();
    for (int i = size - 1; i >= 0; i--) {
        minSum = Math.min(minSum, sum);
        sum += nums.get(i);
        maxSub = Math.max(maxSub, sum - minSum);
        maxSubArray[i] = maxSub;
    }
}
}

```

maxTwoSub i 0, size - 2, i, i + 1.

$$O(2n) = O(n), \quad O(n). \quad O(n). \quad O(n), \quad O(n).$$

Graph

Topological Sorting

Source

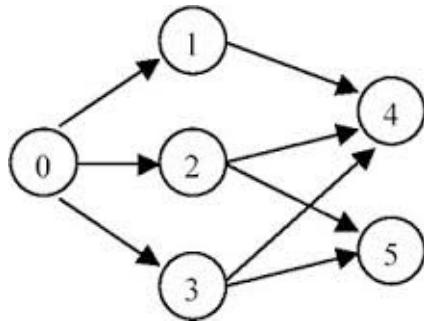
- lintcode: [\(127\) Topological Sorting](#)
- [Topological Sorting - GeeksforGeeks](#)

Given an directed graph, a topological order of the graph nodes is defined as follow:

For each directed edge A → B in graph, A must before B in the order list.

The first node in the order can be any node in the graph with no nodes direct to it.
Find any topological order for the given graph.

Example For graph as follow:



The topological order can be:

[0, 1, 2, 3, 4, 5]
[0, 2, 3, 1, 5, 4]
...

Note

You can assume that there is at least one topological order in the graph.

Challenge

Can you do it in both BFS and DFS?

1 - DFS and BFS

DFS BFS

1. —
2. 0
3. 0 **DFS**

C++

```

/**
 * Definition for Directed graph.
 * struct DirectedGraphNode {
 *     int label;
 *     vector<DirectedGraphNode *> neighbors;
 *     DirectedGraphNode(int x) : label(x) {};
 * };
 */
class Solution {
public:
    /**
     * @param graph: A list of Directed graph node
     * @return: Any topological order for the given graph.
     */
    vector<DirectedGraphNode*> topSort(vector<DirectedGraphNode*> graph) {
        vector<DirectedGraphNode*> result;
        if (graph.size() == 0) return result;

        map<DirectedGraphNode*, int> indegree;
        // get indegree of all DirectedGraphNode
        indeg(graph, indegree);
        // dfs recursively
        for (int i = 0; i < graph.size(); ++i) {
            if (indegree[graph[i]] == 0) {
                dfs(indegree, graph[i], result);
            }
        }
        return result;
    }

private:
    /** get indegree of all DirectedGraphNode
     */
    void indeg(vector<DirectedGraphNode*> &graph,
               map<DirectedGraphNode*, int> &indegree) {

        for (int i = 0; i < graph.size(); ++i) {
            for (int j = 0; j < graph[i]->neighbors.size(); j++) {
                if (indegree.find(graph[i]->neighbors[j]) == indegree.end()) {
                    indegree[graph[i]->neighbors[j]] = 1;
                } else {
                    indegree[graph[i]->neighbors[j]] += 1;
                }
            }
        }
    }
    void dfs(map<DirectedGraphNode*, int> &indegree, DirectedGraphNode *i,
            vector<DirectedGraphNode*> &ret) {

        ret.push_back(i);
        indegree[i]--;
        for (int j = 0; j < i->neighbors.size(); ++j) {
            indegree[i->neighbors[j]]--;
            if (indegree[i->neighbors[j]] == 0) {
                dfs(indegree, i->neighbors[j], ret);
            }
        }
    }
}

```

```

        }
    }
};

}
}
};
```

C++ unordered_map

V E

$O(V + E)$, $O(V)$. 0 $O(V)$. 0 DFS $O(V + E)$.

DFS DFS BFS

$O(V + E)$, $O(V)$.

2 - BFS

DFS BFS,

- 1.
2. BFS 0 BFS
3. BFS0

C++

```

/*
 * Definition for Directed graph.
 * struct DirectedGraphNode {
 *     int label;
 *     vector<DirectedGraphNode *> neighbors;
 *     DirectedGraphNode(int x) : label(x) {};
 * };
 */
class Solution {
public:
    /**
     * @param graph: A list of Directed graph node
     * @return: Any topological order for the given graph.
     */
    vector<DirectedGraphNode*> topSort(vector<DirectedGraphNode*> graph) {
        vector<DirectedGraphNode*> result;
        if (graph.size() == 0) return result;

        map<DirectedGraphNode*, int> indegree;
        // get indegree of all DirectedGraphNode
        indeg(graph, indegree);
        queue<DirectedGraphNode*> q;
```

```

    // bfs
    bfs(graph, indegree, q, result);

    return result;
}

private:
    /** get indegree of all DirectedGraphNode
     */
    void indeg(vector<DirectedGraphNode*> &graph,
               map<DirectedGraphNode*, int> &indegree) {

        for (int i = 0; i < graph.size(); ++i) {
            for (int j = 0; j < graph[i]->neighbors.size(); j++) {
                if (indegree.find(graph[i]->neighbors[j]) == indegree.end()) {
                    indegree[graph[i]->neighbors[j]] = 1;
                } else {
                    indegree[graph[i]->neighbors[j]] += 1;
                }
            }
        }
    }

    void bfs(vector<DirectedGraphNode*> &graph, map<DirectedGraphNode*, int> &indegree,
             queue<DirectedGraphNode *> &q, vector<DirectedGraphNode*> &ret) {

        for (int i = 0; i < graph.size(); ++i) {
            if (indegree[graph[i]] == 0) {
                ret.push_back(graph[i]);
                q.push(graph[i]);
            }
        }

        while (!q.empty()) {
            DirectedGraphNode *cur = q.front();
            q.pop();
            for(int j = 0; j < cur->neighbors.size(); ++j) {
                indegree[cur->neighbors[j]]--;
                if (indegree[cur->neighbors[j]] == 0) {
                    ret.push_back(cur->neighbors[j]);
                    q.push(cur->neighbors[j]);
                }
            }
        }
    };
}

```

C++0 map 1 map map indegree 0

1 $O(V + E)$, $O(V)$.

Reference

- [Topological Sorting Java/C++/Python](#)

Data Structure

Queue, Stack

Implement Queue by Two Stacks

Source

- lintcode: [\(40\) Implement Queue by Two Stacks](#)

As the title described, you should only use two stacks to implement a queue's actions.

The queue should support `push(element)`, `pop()` and `top()` where `pop` is pop the first(a.k.a front) element in the queue.

Both `pop` and `top` methods should return the value of first element.

Example

For `push(1)`, `pop()`, `push(2)`, `push(3)`, `top()`, `pop()`, you should return 1, 2 and 2

Challenge

implement it by two stacks, do not use any other data structure and push, pop and top should be $O(1)$ by AVERAGE.

LIFO, FIFO, 12, LIFO + LIFO ==> FIFO, push Queue.

Java

```
public class Solution {
    private Stack<Integer> stack1;
    private Stack<Integer> stack2;

    public Solution() {
        // source stack
        stack1 = new Stack<Integer>();
        // target stack
        stack2 = new Stack<Integer>();
    }

    public void push(int element) {
        stack1.push(element);
    }

    public int pop() {
        if (stack2.empty()) {
            stack1ToStack2(stack1, stack2);
        }
        return stack2.pop();
    }

    public int top() {
        if (stack2.empty()) {

```

```
        stack1ToStack2(stack1, stack2);
    }
    return stack2.peek();
}

private void stack1ToStack2(Stack<Integer> stack1, Stack<Integer> stack2) {
    while (!stack1.empty()) {
        stack2.push(stack1.pop());
    }
}
```

112

push $O(1)$.

Reference

- [Implement Queue by Two Stacks Java/C++/Python](#)

Min Stack

Source

- lintcode: [\(12\) Min Stack](#)

Implement a stack with `min()` function, which will return the smallest number in the stack.

It should support `push`, `pop` and `min` operation all in $O(1)$ cost.

Example

Operations: `push(1), pop(), push(2), push(3), min(), push(1), min()` Return: `1, 2, 1`

Note

`min` operation will never be called if there is no number in the stack

$O(1)$ $O(1)$

Java

```
public class Solution {
    public Solution() {
        stack1 = new Stack<Integer>();
        stack2 = new Stack<Integer>();
    }

    public void push(int number) {
        stack1.push(number);
        if (stack2.empty()) {
            stack2.push(number);
        } else {
            stack2.push(Math.min(number, stack2.peek()));
        }
    }

    public int pop() {
        stack2.pop();
        return stack1.pop();
    }

    public int min() {
        return stack2.peek();
    }

    private Stack<Integer> stack1; // original stack
    private Stack<Integer> stack2; // min stack
}
```

(null)

$O(1)$.

Sliding Window Maximum

Source

- leetcode: [Sliding Window Maximum | LeetCode OJ](#)
- lintcode: [\(362\) Sliding Window Maximum](#)

Given an array of n integer with duplicate number, and a moving window(size k), move the window at each iteration from the start of the array, find the maximum number inside the window at each moving.

Example

For array [1, 2, 7, 7, 8], moving window size k = 3. return [7, 7, 8]

At first the window is at the start of the array like this

[|1, 2, 7| ,7, 8] , return the maximum 7;

then the window move one step forward.

[1, |2, 7 ,7|, 8], return the maximum 7;

then the window move one step forward again.

[1, 2, |7, 7, 8|], return the maximum 8;

Challenge

$O(n)$ time and $O(k)$ memory

$O(nk)$

$O(1)$

$O(1)$

Java

```
public class Solution {
    /**
     * @param nums: A list of integers.
     * @return: The maximum number inside the window at each moving.
     */
    public ArrayList<Integer> maxSlidingWindow(int[] nums, int k) {
        ArrayList<Integer> winMax = new ArrayList<Integer>();
        if (nums == null || nums.length == 0 || k <= 0) return winMax;

        int len = nums.length;
        Deque<Integer> deque = new ArrayDeque<Integer>();
        for (int i = 0; i < len; i++) {
            // remove the smaller in the rear of queue
            if (deque.size() > 0 && deque.peekFirst() <= i - k)
                deque.pollFirst();
            while (deque.size() > 0 && deque.peekLast() <= nums[i])
                deque.pollLast();
            deque.offerLast(nums[i]);
            if (i + 1 >= k)
                winMax.add(deque.peekFirst());
        }
        return winMax;
    }
}
```

```

        while ((!deque.isEmpty()) && (nums[i] > deque.peekLast())) {
            deque.pollLast();
        }
        // push element in the rear of queue
        deque.offer(nums[i]);
        // remove invalid max
        if (i + 1 > k && deque.peekFirst() == nums[i - k]) {
            deque.pollFirst();
        }
        // add max in current window
        if (i + 1 >= k) {
            winMax.add(deque.peekFirst());
        }
    }

    return winMax;
}
}

```

1.

2. `nums[i] > deque.peekLast()`3. `i k` $O(n)$, $O(k)$. k $O(k)$.

Reference

- Offer
- [sliding-window-maximum Java/C++/Python](#)
- [Maximum of all subarrays of size k \(Added a \$O\(n\)\$ method\) - GeeksforGeeks](#)

Longest Words

Source

- lintcode: [\(133\) Longest Words](#)

Given a dictionary, find all of the longest words in the dictionary.

Example

Given

```
{
    "dog",
    "google",
    "facebook",
    "internationalization",
    "blabla"
}
the longest words are(is) ["internationalization"].
```

Given

```
{
    "like",
    "love",
    "hate",
    "yes"
}
the longest words are ["like", "love", "hate"].
```

Challenge

It's easy to solve it in two passes, can you do it in one pass?

Java

```
class Solution {
    /**
     * @param dictionary: an array of strings
     * @return: an arraylist of strings
     */
    ArrayList<String> longestWords(String[] dictionary) {
        ArrayList<String> result = new ArrayList<String>();
        if (dictionary == null || dictionary.length == 0) return result;

        for (String str : dictionary) {
            // combine empty and shorter length
```

```
    if (result.isEmpty() || str.length() > result.get(0).length()) {
        result.clear();
        result.add(str);
    } else if (str.length() == result.get(0).length()) {
        result.add(str);
    }
}

return result;
}
}
```

$O(n)$, $n - 1$ $O(n)$.

Reference

- [Lintcode: Longest Words | codesolutiony](#)

Problem Misc

Nuts and Bolts Problem

Source

- lintcode: [\(399\) Nuts & Bolts Problem](#)

Given a set of n nuts of different sizes and n bolts of different sizes.
 There is a one-one mapping between nuts and bolts.
 Comparison of a nut to another nut or a bolt to another bolt is not allowed.
 It means nut can only be compared with bolt and bolt can only
 be compared with nut to see which one is bigger/smaller.

We will give you a compare function to compare nut with bolt.

Example

Given nuts = ['ab', 'bc', 'dd', 'gg'], bolts = ['AB', 'GG', 'DD', 'BC'].

Your code should find the matching bolts and nuts.

one of the possible return:

nuts = ['ab', 'bc', 'dd', 'gg'], bolts = ['AB', 'BC', 'DD', 'GG'].

we will tell you the match compare function.

If we give you another compare function.

the possible return is the following:

nuts = ['ab', 'bc', 'dd', 'gg'], bolts = ['BC', 'AA', 'DD', 'GG'].

So you must use the compare function that we give to do the sorting.

The order of the nuts or bolts does not matter.

You just need to find the matching bolt for each nut.

nuts bolts nuts nuts compare nuts bolts nuts bolts

$$O(n^2), \quad O(n \log n), \quad O(n \log n)$$

$$O(n)$$

partition ()

nuts bolts partition bolts nuts partition

Python

```

# class Comparator:
#     def cmp(self, a, b)
# You can use Compare.cmp(a, b) to compare nuts "a" and bolts "b",
# if "a" is bigger than "b", it will return 1, else if they are equal,
# it will return 0, else if "a" is smaller than "b", it will return -1.
# When "a" is not a nut or "b" is not a bolt, it will return 2, which is not valid.
class Solution:
    # @param nuts: a list of integers
    # @param bolts: a list of integers
    # @param compare: a instance of Comparator
    # @return: nothing
    def sortNutsAndBolts(self, nuts, bolts, compare):
        if nuts is None or bolts is None:
            return
        if len(nuts) != len(bolts):
            return
        self.qsort(nuts, bolts, 0, len(nuts) - 1, compare)

    def qsort(self, nuts, bolts, l, u, compare):
        if l >= u:
            return
        # find the partition index for nuts with bolts[l]
        part_inx = self.partition(nuts, bolts[l], l, u, compare)
        # partition bolts with nuts[part_inx]
        self.partition(bolts, nuts[part_inx], l, u, compare)
        # qsort recursively
        self.qsort(nuts, bolts, l, part_inx - 1, compare)
        self.qsort(nuts, bolts, part_inx + 1, u, compare)

    def partition(self, alist, pivot, l, u, compare):
        m = l
        i = l + 1
        while i <= u:
            if compare.cmp(alist[i], pivot) == -1 or \
                compare.cmp(pivot, alist[i]) == 1:
                m += 1
                alist[i], alist[m] = alist[m], alist[i]
                i += 1
            elif compare.cmp(alist[i], pivot) == 0 or \
                compare.cmp(pivot, alist[i]) == 0:
                # swap nuts[l]/bolts[l] with pivot
                alist[i], alist[l] = alist[l], alist[i]
            else:
                i += 1
        # move pivot to proper index
        alist[l], alist[m] = alist[m], alist[l]

        return m

```

C++

```

/**
 * class Comparator {
 *     public:
 *         int cmp(string a, string b);
 * };
 * You can use compare.cmp(a, b) to compare nuts "a" and bolts "b",

```

```

* if "a" is bigger than "b", it will return 1, else if they are equal,
* it will return 0, else if "a" is smaller than "b", it will return -1.
* When "a" is not a nut or "b" is not a bolt, it will return 2, which is not valid.
*/
class Solution {
public:
    /**
     * @param nuts: a vector of integers
     * @param bolts: a vector of integers
     * @param compare: a instance of Comparator
     * @return: nothing
     */
    void sortNutsAndBolts(vector<string> &nuts, vector<string> &bolts, Comparator compare)
    {
        if (nuts.empty() || bolts.empty()) return;
        if (nuts.size() != bolts.size()) return;

        qsort(nuts, bolts, compare, 0, nuts.size() - 1);
    }

private:
    void qsort(vector<string>& nuts, vector<string>& bolts, Comparator compare,
               int l, int u) {

        if (l >= u) return;
        // find the partition index for nuts with bolts[l]
        int part_inx = partition(nuts, bolts[l], compare, l, u);
        // partition bolts with nuts[part_inx]
        partition(bolts, nuts[part_inx], compare, l, u);
        // qsort recursively
        qsort(nuts, bolts, compare, l, part_inx - 1);
        qsort(nuts, bolts, compare, part_inx + 1, u);
    }

    int partition(vector<string>& str, string& pivot, Comparator compare,
                  int l, int u) {

        int m = l;
        for (int i = l + 1; i <= u; ++i) {
            if (compare.getCmp(str[i], pivot) == -1 ||
                compare.getCmp(pivot, str[i]) == 1) {

                ++m;
                std::swap(str[m], str[i]);
            } else if (compare.getCmp(str[i], pivot) == 0 ||
                       compare.getCmp(pivot, str[i]) == 0) {
                // swap nuts[l]/bolts[l] with pivot
                std::swap(str[i], str[l]);
                --i;
            }
        }
        // move pivot to proper index
        std::swap(str[m], str[l]);

        return m;
    }
};

```

Java

```

/**
 * public class NBCompare {
 *     public int cmp(String a, String b);
 * }
 * You can use compare.cmp(a, b) to compare nuts "a" and bolts "b",
 * if "a" is bigger than "b", it will return 1, else if they are equal,
 * it will return 0, else if "a" is smaller than "b", it will return -1.
 * When "a" is not a nut or "b" is not a bolt, it will return 2, which is not valid.
 */
public class Solution {
    /**
     * @param nuts: an array of integers
     * @param bolts: an array of integers
     * @param compare: a instance of Comparator
     * @return: nothing
     */
    public void sortNutsAndBolts(String[] nuts, String[] bolts, NBComparator compare) {
        if (nuts == null || bolts == null) return;
        if (nuts.length != bolts.length) return;

        qsort(nuts, bolts, compare, 0, nuts.length - 1);
    }

    private void qsort(String[] nuts, String[] bolts, NBComparator compare,
                       int l, int u) {
        if (l >= u) return;
        // find the partition index for nuts with bolts[l]
        int part_inx = partition(nuts, bolts[l], compare, l, u);
        // partition bolts with nuts[part_inx]
        partition(bolts, nuts[part_inx], compare, l, u);
        // qsort recursively
        qsort(nuts, bolts, compare, l, part_inx - 1);
        qsort(nuts, bolts, compare, part_inx + 1, u);
    }

    private int partition(String[] str, String pivot, NBComparator compare,
                          int l, int u) {
        //
        int m = l;
        for (int i = l + 1; i <= u; i++) {
            if (compare.cmp(str[i], pivot) == -1 ||
                compare.cmp(pivot, str[i]) == 1) {
                //
                m++;
                swap(str, i, m);
            } else if (compare.cmp(str[i], pivot) == 0 ||
                       compare.cmp(pivot, str[i]) == 0) {
                // swap nuts[l]/bolts[l] with pivot
                swap(str, i, l);
                i--;
            }
        }
        // move pivot to proper index
        swap(str, m, l);

        return m;
    }
}

```

```
}

private void swap(String[] str, int l, int r) {
    String temp = str[l];
    str[l] = str[r];
    str[r] = temp;
}
```

◀ 1 ▶ 1

```
partition  compare.cmp(alist[i], pivot), compare.cmp(pivot, alist[i]),      alist[i]
== pivot    alist[l]    i l+1    alist[l] pivot while pivot alist[m] alist[l]
```

$O(2n \log n)$, $O(1)$.

Reference

- [LintCode/Nuts & Bolts Problem.py at master · algorhythms/LintCode](#)

Heapify

Source

- lintcode: [\(130\) Heapify](#)

Given an integer array, heapify it into a min-heap array.

For a heap array A, A[0] is the root of heap, and for each A[i], A[i * 2 + 1] is the left child of A[i] and A[i * 2 + 2] is the right child of A[i].

Example

Given [3,2,1,4,5], return [1,2,3,4,5] or any legal heap array.

Challenge

O(n) time complexity

Clarification

What is heap?

Heap is a data structure, which usually have three methods: push, pop and top.
where "push" add a new element the heap,
"pop" delete the minimum/maximum element in the heap,
"top" return the minimum/maximum element.

What is heapify?

Convert an unordered integer array into a heap array.

If it is min-heap, for each element A[i],

we will get A[i * 2 + 1] \geq A[i] and A[i * 2 + 2] \geq A[i].

What if there is a lot of solutions?

Return any of them.

Heap Sort

C++

```
class Solution {
public:
    /**
     * @param A: Given an integer array
     * @return: void
     */
    void heapify(vector<int> &A) {
        // build min heap
        for (int i = A.size() / 2; i >= 0; --i) {
            min_heap(A, i);
        }
    }
}
```

```

    }

private:
    void min_heap(vector<int> &nums, int k) {
        int len = nums.size();
        while (k < len) {
            int min_index = k;
            // left leaf node search
            if (k * 2 + 1 < len && nums[k * 2 + 1] < nums[min_index]) {
                min_index = k * 2 + 1;
            }
            // right leaf node search
            if (k * 2 + 2 < len && nums[k * 2 + 2] < nums[min_index]) {
                min_index = k * 2 + 2;
            }
            if (k == min_index) {
                break;
            }
            // swap with the minimal
            int temp = nums[k];
            nums[k] = nums[min_index];
            nums[min_index] = temp;
            // not only current index
            k = min_index;
        }
    }
};


```

k = min_index

$(N), \dots$

Reference

- [Heap Sort](#)
- [Heapify Java/C++/Python](#)

String to Integer

Source

- leetcode: String to Integer (atoi) | LeetCode OJ
- lintcode: (54) String to Integer(atoi)

Implement function atoi to convert a string to an integer.

If no valid conversion could be performed, a zero value is returned.

If the correct value is out of the range of representable values, INT_MAX (2147483647) or INT_MIN (-2147483648) is returned.

Example

"10" => 10

"-1" => -1

"123123123123123" => 2147483647

"1.0" => 1

0

Java

```
public class Solution {
    /**
     * @param str: A string
     * @return An integer
     */
    public int atoi(String str) {
        if (str == null || str.length() == 0) return 0;

        // trim left and right spaces
        String strTrim = str.trim();
        int len = strTrim.length();
        // sign symbol for positive and negative
        int sign = 1;
        // index for iteration
        int i = 0;
        if (strTrim.charAt(i) == '+') {
            i++;
        } else if (strTrim.charAt(i) == '-') {
            sign = -1;
            i++;
        }
    }
}
```

```
// store the result as long to avoid overflow
long result = 0;
while (i < len) {
    if (strTrim.charAt(i) < '0' || strTrim.charAt(i) > '9') {
        break;
    }
    result = 10 * result + sign * (strTrim.charAt(i) - '0');
    // overflow
    if (result > Integer.MAX_VALUE) {
        return Integer.MAX_VALUE;
    } else if (result < Integer.MIN_VALUE) {
        return Integer.MIN_VALUE;
    }
    i++;
}

return (int)result;
}
}
```

while while long

Reference

- [String to Integer \(atoi\) Java/C++/Python](#)

Appendix I Interview and Resume

Interview

Facebook workshop - Crush Your Coding Interview

Facebook 5Frank Facebook Q & A FT :) logo

CSer ~

Facebook [Facebook](#)

slides

Resume

What to include on your resume

1. University, degree, expected graduation date
 - Highly recommended including GPA with scale/ranking
2. Projects
 - Industry experience (internships, competition, full-time)
 - Interesting projects
 - Links where applicable (github, apps, websites)

//(HR)GPA GPA

Writing a great resume

1. Focus on what you did
2. Focus on Impact(metrics and numbers are a plus)
3. Be specific and concise (1 page if at all possible)
4. Pro tip: always start with an active verb
 - example: built, optimized, improved, doubled, etc
5. Don't include
 - Age, photo, ID number

ID

Coding interview

Goals of a coding interview

Protip: Think out loud!

1. How you think and tackle technical problems
2. How you consider engineering trade offs (speed vs. time)

3. How you communicate in English about codes
4. Limits of what you know
 - Don't feel bad if you don't get all answers right

What is covered?

Use your comfortable coding language (C++ Java would be better)

Google C++ Java

1. Data structures and algorithms
 - implement, not memorize
 - discuss complexity (space and time trade-offs)
 - Common library functions are fair game
2. Specific questions about concepts are rare
 - Unless you claim to be an expert or need the concept

During the interview

1. Clarify your understanding
 - ask questions until you fully understand problem space and constraints
 - validate or state any assumptions
 - draw pictures to help you better understand problems
2. Focus on getting a working solution first
 - handle corner cases
3. Iterate
 - 1.
 2. code
 - 3.

Done is better than perfect

be yourself,

//

1. context:
2. action:
3. result:

behavior question

1. motivation
2. passion:
3. team pair: ?
4. disagreement:

1. Think out loud, .
- 2.
3. shit code,
- 4.
- 5.

Reference

- www.geeksforgeeks.org -
- [Programming Interview Questions | CareerCup](#) - CC150
- [Glassdoor – Get Hired. Love Your Job.](#) - yelp
- [|](#) - mitbbs
- [-|||](#) -
- [\(JobHunting\) | \(mitbbs.com\)](#) jobhunting
- [Top 50](#) - - - Top 50
- - Evernote
- [on Vimeo](#) - Coursera
- [Facebook](#) . [Facebook - biaobiaoqi](#) - - Facebook Frank 

Resume

Facebook Gayle

The Google Resume

Resume Template

Markdown Latex sharelatex

CV or Resume modernCV

Professional CV

[LaTeX Templates](#) » Curricula Vitae/Résumés — [billryan/resume](#), [FontAwesome](#) ~

你的大名

✉ xxx@yuanbin.me ✆ (+86) 131-221-87xxx 🌐 http://www.yuanbin.me

🎓 教育背景

上海交通大学, 上海	2013 – 至今
在读硕士研究生 信息与通信工程, 预计 2016 年 3 月毕业	
西安电子科技大学, 西安, 陕西	2009 – 2013
学士 通信工程	

🐱 实习/项目经历

黑科技公司 上海	2015 年 3 月 – 2015 年 5 月
实习 经理: 高富帅	
xxx 后端开发	
<ul style="list-style-type: none"> 实现了 xxx 特性 后台资源占用率减少 8% xxx 	
分布式科学上网姿势	2014 年 6 月 – 至今
Golang, Linux 个人项目, 和富帅糕合作开发	
分布式负载均衡科学上网姿势, https://github.com/cyfdecyf/cow	
<ul style="list-style-type: none"> 修复了连接未正常关闭导致文件描述符耗尽的 bug 使用 Chord 哈希 URL, 实现稳定可靠地分流 xxx (尽量使用量化的客观结果) 	
LaTeX 简历模板	2015 年 5 月 – 至今
LaTeX, Python 个人项目	
优雅的 LaTeX 简历模板, https://github.com/billryan/resume	
<ul style="list-style-type: none"> 容易定制和扩展 完善的 Unicode 字体支持, 使用 XeLaTeX 编译 支持 FontAwesome 4.3.0 	

⚙️ IT 技能

- 编程语言: C == Python > C++ > Java
- 平台: Linux
- 开发: xxx

♡ 获奖情况

第一名, xxx 比赛	2013 年 6 月
其他奖项	2015

ℹ 其他

- 技术博客: <http://blog.yours.me>
- GitHub: <https://github.com/username>
- 语言: 英语 - 熟练 (TOEFL xxx)

Reference

- *The Google Resume* -
- [—— | @Get](#) -
- [——47 - Lucida](#) - Effective
- [- V2EX](#) -
- *Cracking the coding interview*

BFS

Breadth-First Search,

[11.5. Binary Tree Level Order Traversal II](#) [13. Exhaustive Search](#) [15.1. Topological Sorting](#)

DFS

Depth-First Search,

[14.3. Minimum Path Sum](#) [14.1. Triangle](#) [13. Exhaustive Search](#) [13.11. Palindrome Partitioning](#)
[13.1. Subsets](#) [13.7. Unique Binary Search Trees II](#) [15.1. Topological Sorting](#)
[9.13. Print Numbers by Recursion](#) [6.11. Wildcard Matching](#)

DP_Matrix

$f[x][y]$ (x, y)

[14.3. Minimum Path Sum](#) [14.4. Unique Paths](#) [14.5. Unique Paths II](#)

DP_Sequence

$f[i]$ $i // \dots f[n-1]$

[14. Dynamic Programming](#) [14.9. Longest Increasing Subsequence](#)
[14.10. Palindrome Partitioning II](#) [14.8. Word Break](#)

DP_Two_Sequence

$\cdot f[i][j] \cdot i \ j$

[14. Dynamic Programming](#) [14.18. Distinct Subsequences](#) [14.12. Edit Distance](#)
[14.11. Longest Common Subsequence](#)

TLE

Time Limit Exceeded OJ

14.13. Jump Game II 14.17. Best Time to Buy and Sell Stock IV 14.7. Jump Game
14.3. Minimum Path Sum 14.10. Palindrome Partitioning II 7.2. Zero Sum Subarray
10.15. Copy List with Random Pointer 10.17. Insertion Sort List 10.13. Merge k Sorted Lists
10.14. Reorder List 9.18. Ugly Number 6.9. Longest Palindromic Substring 6.4. Anagrams