15th December 2012

Combination Sum

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.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a_1, a_2, … , a_k) must be in non-descending order. (ie, $a_1 \le a_2 \le \dots \le a_k$).
- The solution set must not contain duplicate combinations.

For example, given candidate set `2,3,6,7` and target `7`,
A solution set is:
`[7]`
`[2, 2, 3]`

[http://www.leetcode.com/onlinejudge#]

Solution

One popular solution online to this problem is to find the combinations for every number from 1 up to target. The following solution is a little faster than the aforementioned one. It is also a DP solution. The general idea is:

1. Sort the input and find the unique elements (because each element can be used unlimited times).

2. For each candidate C_i in the input, the solution is [C_i, resultOf(target - C_i with elements no greater than C_i)]. For example, for target 10 and input [1, 2, 3, 4, 5, 6, 7], the result is the collection of [7, elements <= 7 and sum to 3], [6, elements <= 6 and sum to 4], [5, elements <= 5 and sum to 5], [4, elements <= 4 and sum to 6] …. Note that elements <= 6 and sum to 4 indicates that elements are also <= 4.

```
vector<vector<int> > findComb(vector<int> &candidates, int
 target, int high, map<int, map<int, vector<vector<int> >
> > &mymap){
    vector<vector<int> > result;
    for (int i = candidates.size() - 1; i + 1 > 0; i--){
        int x = candidates[i];
        if (x <= high){ //high stores the upper-bound of t
he elements in the result
            if (target == x){
                result.push_back(vector<int>(1, x));
            }
            else if (target > x){
                int upper = (high < target-x) ? high : (ta
rget-x);
                upper = (upper < x) ? upper : x; //upper i
s the upper-bound of the elements for the intermediate sol
ution
                if (mymap.find(target-x) == mymap.end() ||
 mymap[target-x].find(upper) == mymap[target-x].end())
                    mymap[target-x][upper] = findComb(cand
idates, target-x, upper, mymap);
                vector<vector<int> > temp = mymap[target-x
][upper];
                for (int j = 0; j < temp.size(); j++){
                    temp[j].push_back(x);
                    result.push_back(temp[j]);
                }
            }
        }
    }
    return result;
}

class Solution {
public:
    vector<vector<int> > combinationSum(vector<int> &candi
dates, int target) {
        sort(candidates.begin(), candidates.end());
        vector<int>::iterator it = unique(candidates.begin
(), candidates.end());
        candidates.erase(it, candidates.end());
        map<int, map<int, vector<vector<int> > > > mymap;
        return findComb(candidates, target, target, mymap)
;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 15th December 2012 by Go Ka

0 Add a comment

8th December 2012

Climbing Stairs

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?

[http://www.leetcode.com/onlinejudge#]

Solution:

```
int climb(int n, vector<int> &count){
    if (n < 0)
        return 0;
    if (n == 0)
        return 1;
    if (n < count.size() && count[n] > -1)
        return count[n];
    count[n] = climb(n-1, count) + climb(n-2,count);
    return count[n];
}

class Solution {
public:
    int climbStairs(int n) {
        vector<int> count(n+1, -1);
        return climb(n, count);
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Binary Tree Zigzag Level Order Traversal

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).

For 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]
]
```

Solution:

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */

class Solution {
public:
    vector<vector<int> > zigzagLevelOrder(TreeNode *root) {
        vector<vector<int> > result;
        if (root){
            vector<TreeNode *> temp;
            temp.push_back(root);
            bool backward = 0;
            while (!temp.empty()){
                vector<TreeNode *> nextLevel;
                vector<int> tempV;
                for (int i = 0; i < temp.size(); i++){
                    if (temp[i]->left)
```

```
                            nextLevel.push_back(temp[i]->left);
                        if (temp[i]->right)
                            nextLevel.push_back(temp[i]->right);
                        tempV.push_back(temp[i]->val);
                    }
                    if (backward){//reverse the result
                        for (int i = 0, j = tempV.size() - 1; i < j; i++
, j--){
                            int t = tempV[i];
                            tempV[i] = tempV[j];
                            tempV[j] = t;
                        }
                    }
                    result.push_back(tempV);
                    backward = backward ? 0 : 1;
                    temp = nextLevel;
                }
            }
            return result;
        }
};
```

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012 Binary Tree Maximum Path Sum

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.

For example:
Given the below binary tree,

```
       1
      / \
     2   3
```

Return `6`.

[http://www.leetcode.com/onlinejudge#]

Solution:

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int maxPathSum(TreeNode *root){
        if (root == NULL)
            return INT_MIN;
        int leftMax = maxPathSum(root->left);
        int rightMax = maxPathSum(root->right);
        int currentMax = root->val;
        int leftVal = 0, rightVal = 0;
        if (root->left != NULL && root->left->val > 0){
            currentMax += root->left->val;
            leftVal = root->left->val;
        }
        if (root->right != NULL && root->right->val > 0){
            currentMax += root->right->val;
            rightVal = root->right->val;
        }
        if (rightVal > 0 || leftVal > 0)
            root->val += leftVal > rightVal ? leftVal : rightVal;
        if (leftMax > rightMax)
            return (leftMax > currentMax) ? leftMax : currentMax;
        else
            return (rightMax > currentMax) ? rightMax : currentMax;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Binary Tree Level Order Traversal II

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).

For 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],
]
```

Solution:

We can reverse the result from **Binary Tree Level Order Traversal**. The following solution tries to avoid the reverse process, by inserting new element at the front of the returning vector. But the overhead for inserting at the front of vector is huge.

```
void getOrder(TreeNode *root, int level, vector<vector<int> > &result){
    if (root){
        if (level < result.size()){
            result[result.size() - level - 1].push_back(root->val);
        }
        else {
            result.insert(result.begin(), vector<int>());
            result[result.size() - level - 1].push_back(root->val);
        }
        getOrder(root->left, level+1, result);
        getOrder(root->right, level+1, result);
    }
}

class Solution {
public:
    vector<vector<int> > levelOrderBottom(TreeNode *root) {
        vector<vector<int> > result;
        getOrder(root, 0, result);
        return result;
    }
};
```

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012 Binary Tree Level Order Traversal

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).

For 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]
]
```

Solution:

BFS:

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
```

```
class Solution {
public:
    vector<vector<int> > levelOrder(TreeNode *root) {
        vector<vector<int> > result;
        if (root == NULL)
            return result;
        vector<TreeNode *> temp(1, root);
        while(!temp.empty()){
            vector<TreeNode *> nextLevel;
            result.push_back(vector<int>());
            for (int i = 0; i < temp.size(); i++){
                result.back().push_back(temp[i]->val);
                if (temp[i]->left)
                    nextLevel.push_back(temp[i]->left);
                if (temp[i]->right)
                    nextLevel.push_back(temp[i]->right);
            }
            temp = nextLevel;
        }
        return result;
    }
};
```

DFS:

```
void getOrder(TreeNode *root, int level, vector<vector<int> > &resul
t){
    if (root){
        if (level < result.size()){
            result[level].push_back(root->val);
        }
        else {
            result.push_back(vector<int>());
            result.back().push_back(root->val);
        }
        getOrder(root->left, level+1, result);
        getOrder(root->right, level+1, result);
    }
}

class Solution {
public:
    vector<vector<int> > levelOrder(TreeNode *root) {
        vector<vector<int> > result;
        getOrder(root, 0, result);
        return result;
    }
};
```

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Binary Tree Inorder Traversal

Binary Tree Inorder Traversal

Given a binary tree, return the *inorder* traversal of its nodes' values.

For example:
Given binary tree `{1,#,2,3}` ,

```
   1
    \
     2
    /
   3
```

return `[1,3,2]` .

Note: Recursive solution is trivial, could you do it iteratively?

Solution:

Solution 1: with stack

```
/**
 * Definition for binary tree
 * 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) {
        if (root == NULL)
            return vector<int>();
        stack<TreeNode*> nodeStack;
        vector<int> result;
        while(!nodeStack.empty()||root){
            if (!root){
                root = nodeStack.top();
                nodeStack.pop();
                result.push_back(root->val);
                root = root->right;
            }
            else{
                while(root){
                    nodeStack.push(root);
                    root = root->left;
                }
            }
        }
        return result;
    }
};
```

Solution 2: stackless, modify the tree, without revert it

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode *root) {
        vector<int> result;
        if (root == NULL)
            return result;
        do{
            while (root->left){
                TreeNode *left = root->left;
                if (left){
                    root->left = NULL;
                    TreeNode *tempRoot = left;
                    while (left->right)
                        left = left->right;
                    left->right = root;
                    root = tempRoot;
                }
            }
            result.push_back(root->val);
            root = root->right;
        }while (root);
        return result;
    }
};
```

Solution 3 (morris algorithm): stackless, modify the tree and revert it at the end

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode *root) {
        vector<int> result;
        if (root == NULL)
            return result;
        TreeNode *pre;
        while (root != NULL){
            if (root->left == NULL){
                result.push_back(root->val);
                root = root->right;
            }
            else{
                pre = root->left;
                while (pre->right != NULL && pre->right != root)
                    pre = pre->right;
                if (pre->right == NULL){
                    pre->right = root;
                    root = root->left;
                }
```

```
                    else{
                        pre->right = NULL;
                        result.push_back(root->val);
                        root = root->right;
                    }
                }
            }
            return result;
        }
    };
```

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012 Best Time to Buy and Sell Stock III

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.

Note:
You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

[http://www.leetcode.com/onlinejudge#]

Solution:

```
vector<int> getMaxSum(vector<int> diff){
    vector<int> result;
    int sum = 0; int max = 0;
    for (int i = 0; i < diff.size(); i++){
        sum += diff[i];
        if (sum > max)
            max = sum;
        if (sum < 0)
            sum = 0;
        result.push_back(max);
    }
    return result;
}
```

```
class Solution {
public:
    int maxProfit(vector<int> &prices) {
        if (prices.size() < 2) // note
            return 0;
        vector<int> diff1, diff2;
        for (int i = 1; i < prices.size(); i++)
            diff1.push_back(prices[i] - prices[i-1]);
        for (int i = prices.size() - 1; i > 0; i--)
            diff2.push_back(prices[i] - prices[i-1]);
        vector<int> maxSumForward = getMaxSum(diff1);
        vector<int> maxSumBackward = getMaxSum(diff2);
        int maxProfit = maxSumForward.back(); // note
        for (int i = 0, j = maxSumForward.size() - 2; i < maxSumForw
ard.size() - 1 && j >= 0; i++, j--){
            int sum = maxSumForward[i] + maxSumBackward[j];
            if (sum > maxProfit)
                maxProfit = sum;
        }
        return maxProfit;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012 Best Time to Buy and Sell Stock II

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).

[http://www.leetcode.com/onlinejudge#]

Solution:

```
class Solution {
public:
    int maxProfit(vector<int> &prices) {
        int sum = 0;
        for (int i = 1; i < prices.size(); i++){
            int diff = prices[i] - prices[i-1];
            if (diff > 0)
                sum += diff;
        }
        return sum;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Best Time to Buy and Sell Stock

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.

[http://www.leetcode.com/onlinejudge#]

Solution:

```
class Solution {
public:
    int maxProfit(vector<int> &prices) {
        int max = 0, sum = 0;
        for (int i = 1; i < prices.size(); i++){
            int diff = prices[i] - prices[i-1];
            sum += diff;
            if (sum > max)
                max = sum;
            if (sum < 0)
                sum = 0;
        }
        return max;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Balanced Binary Tree

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 depth of the two subtrees of *every*node never differ by more than 1.

[http://www.leetcode.com/onlinejudge#]

Solution:

```
/**
 * Definition for binary tree
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
int abs(int i){
    return i > 0 ? i : -i;
}

bool checkHeight(TreeNode *root, int &height){
    if (root == NULL){
        height = 0;
        return 1;
    }
    int lh, rh;
    bool left = checkHeight(root->left, lh);
    bool right = checkHeight(root->right, rh);
    height = lh > rh ? (lh + 1) : (rh + 1);
    return left && right && abs(lh - rh) < 2;
}

class Solution {
public:
    bool isBalanced(TreeNode *root) {
        int height;
        return checkHeight(root, height);
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Anagrams

Anagrams

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

Note: All inputs will be in lower-case.

[http://www.leetcode.com/onlinejudge#]

Solution:

```
class Solution {
public:
    vector<string> anagrams(vector<string> &strs) {
        map<string, vector<string> > groups;
        vector<string> result;
        for (vector<string>::iterator iter = strs.begin(); iter != s
trs.end(); iter++){
            string s = *iter;
            sort(s.begin(), s.end());
            groups[s].push_back(*iter);
        }
        for (map<string, vector<string> >::iterator iter = groups.be
gin(); iter != groups.end(); iter++){
            if (iter->second.size() > 1)
                result.insert(result.end(), iter->second.begin(), it
er->second.end());
        }
        return result;
    }

};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Add Two Numbers

Add Two Numbers

You are given two linked lists representing two non-negative numbers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

```
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8
```

[http://www.leetcode.com/onlinejudge#]

Solution:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *addTwoNumbers(ListNode *l1, ListNode *l2) {
        if (l1 == NULL)
            return l2;
        if (l2 == NULL)
            return l1;
        ListNode *p = new ListNode(0), *q = p;
        int carry = 0;
        while (l1 || l2){
            int sum = carry;
            if (l1) {sum += l1->val; l1 = l1->next;}
            if (l2) {sum += l2->val; l2 = l2->next;}
            q->val = sum%10;
            carry = sum/10;
            if (l1 || l2 || carry){
                q->next = new ListNode(carry);
                q = q->next;
            }
        }
        return p;
    }
};
```

My original solution:

```
class Solution {
```

```
public:
    ListNode *addTwoNumbers(ListNode *l1, ListNode *l2) {
        if (l1 == NULL)
            return l2;
        if (l2 == NULL)
            return l1;
        ListNode *p = new ListNode(0), *q = p, *r = p;
        int carry = 0;
        while (l1 || l2){
            int sum = carry;
            if (l1) {sum += l1->val; l1 = l1->next;}
            if (l2) {sum += l2->val; l2 = l2->next;}
            q->next = new ListNode(sum%10);
            q = q->next;
            carry = sum/10;
        }
        if (carry)
            q->next = new ListNode(carry);
        p = p->next;
        r->next = 0;
        delete r;
        return p;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

Add Binary

Add Binary

Given two binary strings, return their sum (also a binary string).

For example,

a = "11"

b = "1"

Return "100" .

[http://www.leetcode.com/onlinejudge#]

Solution:

```
class Solution {
public:
    string addBinary(string a, string b){
        int la = a.size() - 1;
        int lb = b.size() - 1;
        string result = la > lb ? a : b;
        int carryon = 0;
        for (int i = result.size() - 1; i >= 0; i--){
            int c = carryon;
            if (la >= 0) c += a[la--] - '0';
            if (lb >= 0) c += b[lb--] - '0';
            result[i] = c%2  + '0';
            carryon = c/2;
        }
        if (carryon)
            result.insert(result.begin(), '1');
        return result;
    }
};
```

My original:

```
class Solution {
public:
    string addBinary(string a, string b){
        string longer = (a.size() >= b.size()) ? a : b;
        string shorter = (a.size() >= b.size()) ? b : a;
        string result = longer;
        char carryon = '0';
        int diff = longer.size() - shorter.size();
        for (int i = longer.size() - 1; i >= diff; i--){
            if ((longer[i] == '1' && shorter[i - diff] == '1') || (l
onger[i] == '0' && shorter[i - diff] == '0')){
                result[i] = carryon;
                carryon = (longer[i] == '1') ? '1' : '0';
            }
            else{
                result[i] = (carryon == '1') ? '0' : '1';
                carryon = (carryon == '1') ? '1' : '0';
            }
        }
        for (int i = diff - 1; i >= 0; i--){
            if (carryon == '1'){
                result[i] = longer[i] == '1' ? '0' : '1';
                carryon = longer[i] == '1' ? '1' : '0';
            }
        }
        if (carryon == '1')
            result.insert(result.begin(), carryon);
        return result;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 8th December 2012 by Go Ka

0 Add a comment

8th December 2012

4sum

4Sum

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

Note:

- Elements in a quadruplet (*a*,*b*,*c*,*d*) must be in non-descending order. (ie, *a* ≤ *b* ≤ *c* ≤ *d*)
- The solution set must not contain duplicate quadruplets.

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

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

[http://www.leetcode.com/onlinejudge#]

Solution:

This is a straightforward solution based on 3sum.

```
class Solution {
public:
     vector<vector<int> > fourSum(vector<int> &num, int target) {
          vector<vector<int> > result;
          if (num.size() < 4)
               return result;
          vector<int> sorted = num;
          sort(sorted.begin(), sorted.end());
          int n = sorted.size();
          for (int i = 0; i + 3 < n; i++){
               if (i == 0 || sorted[i] != sorted[i-1]){ //skip the d
uplicated first element
                    for (int j = i + 1; j + 2 < n; j++){
                         if (j == i+1 || sorted[j] != sorted[j-1]){
//skip duplicated second element
                              int l = j + 1, r = n - 1;
                              while (l < r){
                                   int sum = sorted[i] + sorted[j] +
 sorted[l] + sorted[r];
                                   if (sum == target){
```

```
                                                vector<int> temp;
                                                temp.push_back(sorted[i]);
                                                temp.push_back(sorted[j]);
                                                temp.push_back(sorted[l]);
                                                temp.push_back(sorted[r]);
                                                result.push_back(temp);
                                                while (l < r && sorted[l] ==
 sorted[l+1]) //skip duplicated third element
                                                     l++;
                                                l++; //NO need to skip the f
ourth duplicated
                                           }
                                           else if (sum < target)
                                                l++;
                                           else
                                                r--;
                                      }
                                 }
                            }
                       }
                  }
                  return result;
             }
        };
```

Posted 8th December 2012 by Go Ka

0 Add a comment

7th December 2012

4sum

4Sum

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

Note:

- Elements in a quadruplet (*a*,*b*,*c*,*d*) must be in non-descending order. (ie, *a* ≤ *b* ≤ *c* ≤ *d*)
- The solution set must not contain duplicate quadruplets.

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

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

[http://www.leetcode.com/onlinejudge#]

Solution:

There is a straightforward solution to this problem based on 3sum, with an average/worst-case complexity of O(n^3). The following solution tries to provide an O(n^2 lg n) solution, but fails. My analysis shows that in the worst case, the output size is O(n^3), which indicates there cannot be a better solution than O(n^3). Please corret me if I am wrong.

Analysis:
For input (-n, -n+1, -n+2, ..., 0, 1, ..., n-1, n), with target 0.
The following are possible combinations.

Negative group:	Positive group:
(-n, -n+1)	(n-1, n)
(-n, -n+2)	(n-2, n)
(-n, -n+3) (-n+1, -n+2)	(n-3, n) (n-2, n-1)
(-n, -n+4) (-n+1, -n+3)	(n-4, n) (n-3, n-1)
(-n, -n+5) (-n+1, -n+4) (-n+2, -n+3)	(n-5, n) (n-4, n-1) (n-3, n-2)
(-n, -n+6) (-n+1, -n+5) (-n+2, -n+4)	(n-6, n) (n-5, n-1) (n-4, n-2)
....	....
(-n, -1) (-n+1, -2) .... (-n/2-1, -n/2)	(1, n) (2, n-1) ..... (n/2, n/2+1)

Note that the last line in each group has n/2 pairs, which can make n^2/4 combinations that sum to 0. So the total is no less than (1 + 4 + 9 + ... + n^2/4) = O(n^3).

The following solution is straightforward, but is more complex when comes to details. The high level idea is: group every pair of elements (O(n^2)), sort the pairs according to sum (O(n^2 lg n)). Find the quadruplet just as 2sum (O(n^3)). It becomes tricky when there are pairs of the same sum. E.g.,
for input (-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5), with target 0
we get .... (-5, 0) (-4, -1) (-3, -2) .... (0, 5) (1, 4) (2, 3) ....
possible output (-5, 0, 2, 3) **(-5, 0, 1, 4) (-4, -1, 2, 3) (-3, -2, 2, 3)** (-4, -1, 1, 4) **(-4, -1, 0, 5) (-3, -2, 1, 4)** ...
Those in bold are different from a simple run of 2sum, since duplicates are typically ignored in 2sum.
Therefore, when come to case like this, lines 20 to 35 test all possible combinations. This makes the worst case overhead O(n^3) in certain situations.

```
1:  struct sumTwo{ //struct to store two elements
2:       int ind1, ind2, sum; //ind1 and ind2 are the indices of the
 element in the input array
```

```
    sumTwo(int x1, int x2, int v1, int v2): ind1(x1), ind2(x2), sum(v1 + v2){}
    bool operator==(sumTwo &s){ // check whether two sumTwos overlap
        return (ind1 == s.ind1 || ind1 == s.ind2 || ind2 == s.ind1 || ind2 == s.ind2);
    }
};
```

```
bool comp(sumTwo s1, sumTwo s2){ //compare two sumTwos, return true if s1 < s2
    if (s1.sum != s2.sum)
        return s1.sum < s2.sum;
    else if (s1.ind1 == s2.ind1) //tie firstly breaks by first element
        return s1.ind2 < s2.ind2;//tie then breaks by second element
    else
        return s1.ind1 < s2.ind1;
}
```

```
vector<int> sortTwos(sumTwo s1, sumTwo s2, const vector<int> &num){ //used when insert into the final result
    int iarr[4];
    iarr[0] = num[s1.ind1];
    iarr[1] = num[s1.ind2];
    iarr[2] = num[s2.ind1];
    iarr[3] = num[s2.ind2];
    sort(iarr, iarr + 4);
    return vector<int>(iarr, iarr + 4);
}
```

```
class Solution {
public:
    vector<vector<int> > fourSum(vector<int> &num, int target) {
        set<vector<int> > result;
        vector<vector<int> > ret;
        sort(num.begin(), num.end());
        vector<sumTwo> sumTwos;
        for (int i = 0; i < num.size(); i++){ // n^2
            for (int j = i + 1; j < num.size(); j++)
                sumTwos.push_back(sumTwo(i, j, num[i], num[j]));
        }
        sort(sumTwos.begin(), sumTwos.end(), comp); //n^2\lg(n^2)
        int i = 0, j = sumTwos.size() - 1;
        while (i < j){ // n^2
            if (sumTwos[i].sum + sumTwos[j].sum == target){
                if (!(sumTwos[i] == sumTwos[j]))
```

```cpp
                    result.insert(sortTwos(sumTwos[i], sumTwos[j], num));
                    //decide how to move the cursors. if the next element on both ends is equal to the current element respectively
                    //then the result should include all elements of the same value
                if (sumTwos[i].sum == sumTwos[i+1].sum && sumTwos[j].sum == sumTwos[j-1].sum){
                    int k = 1;
                    while (j - k > i && sumTwos[j].sum == sumTwos[j-k].sum){  //worst case: n
                        if (!(sumTwos[i] == sumTwos[j-k]))
                            result.insert(sortTwos(sumTwos[i], sumTwos[j-k], num));
                        k++;
                    }
                    k = 0;
                    while (i + k < j && sumTwos[i].sum == sumTwos[i+k].sum){  //worst case: n
                        if (!(sumTwos[i+k] == sumTwos[j]))
                            result.insert(sortTwos(sumTwos[i+k], sumTwos[j], num));
                        k++;
                    }
                    i++;
                    j--;
                }
                else if (sumTwos[i].sum == sumTwos[i+1].sum)
                    i++;
                else
                    j--;
            }
            else if (sumTwos[i].sum + sumTwos[j].sum < target)
                i++;
            else
                j--;
        for (set<vector<int> >::iterator iter = result.begin(); iter != result.end(); iter++)
            ret.push_back(*iter);
        return ret;
    }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 7th December 2012 by Go Ka

0 Add a comment

My LeetCode OJ Sol...

search

Classic Flipcard Magazine Mosaic Sidebar Snapshot Timeslide

3Sum Closest

Given an array *S* of *n* integers, find three integers in *S* such that the su sest to a given number, target. Return the sum of the three integers. You ume 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).
```

[http://www.leetcode.com/onlinejudge#]

Solutions:

```
class Solution {
public:
  int threeSumClosest(vector<int> &num, int target) {
    int closest = INT_MAX;
       int n = num.size();
       sort(num.begin(), num.end());
       for (int i = 0; i + 2 < n; i++){
            int j = i + 1, k = n - 1;
            while (j < k){
                  int dist = num[i] + num[j] + num[k] - target;
                  if (dist == 0)
                       return target;
                  if (dist > 0)
                       k--;
                  else{
                       j++;
                  }
                  if (abs(dist) < abs(closest)){
                       closest = dist;
                  }
```

```
                }
            }
            return closest + target;
        }
    };
```

Posted 7th December 2012 by Go Ka

0 Add a comment

7th December 2012

3sum

3Sum

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.

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.

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

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

[http://www.leetcode.com/onlinejudge#]

Solution:

```
class Solution {
public:
  vector<vector<int> > threeSum(vector<int> &num) {
    vector<int> sortedNum = num;
    sort(sortedNum.begin(), sortedNum.end());
    vector<vector<int> > result;
    for (int i = 0; i + 2 < sortedNum.size(); i++){
      if (i == 0 || sortedNum[i] != sortedNum[i-1]){
        int tempSum = -sortedNum[i];
         int leftInd = i + 1;
         int rightInd = sortedNum.size() - 1;
    while (leftInd < rightInd){
     if (sortedNum[leftInd] + sortedNum[rightInd] == tempSum){
      vector<int> triplet;
      triplet.push_back(sortedNum[i]);
      triplet.push_back(sortedNum[leftInd]);
      triplet.push_back(sortedNum[rightInd]);
      result.push_back(triplet);
      while(leftInd < rightInd && sortedNum[leftInd] == sortedNum[leftInd+1])
       leftInd++;
      leftInd++;
     }
     else if (sortedNum[leftInd] + sortedNum[rightInd] > tempSum)
      rightInd--;
     else
      leftInd++;
    }
   }
  }
  return result;
 }
};
```

[http://www.leetcode.com/onlinejudge#]

Posted 7th December 2012 by Go Ka

0 Add a comment

7th December 2012

Online tool for formatting source code in HTML

http://tohtml.com/cpp/

http://codeformatter.blogspot.com/

Posted 7th December 2012 by Go Ka

0 Add a comment