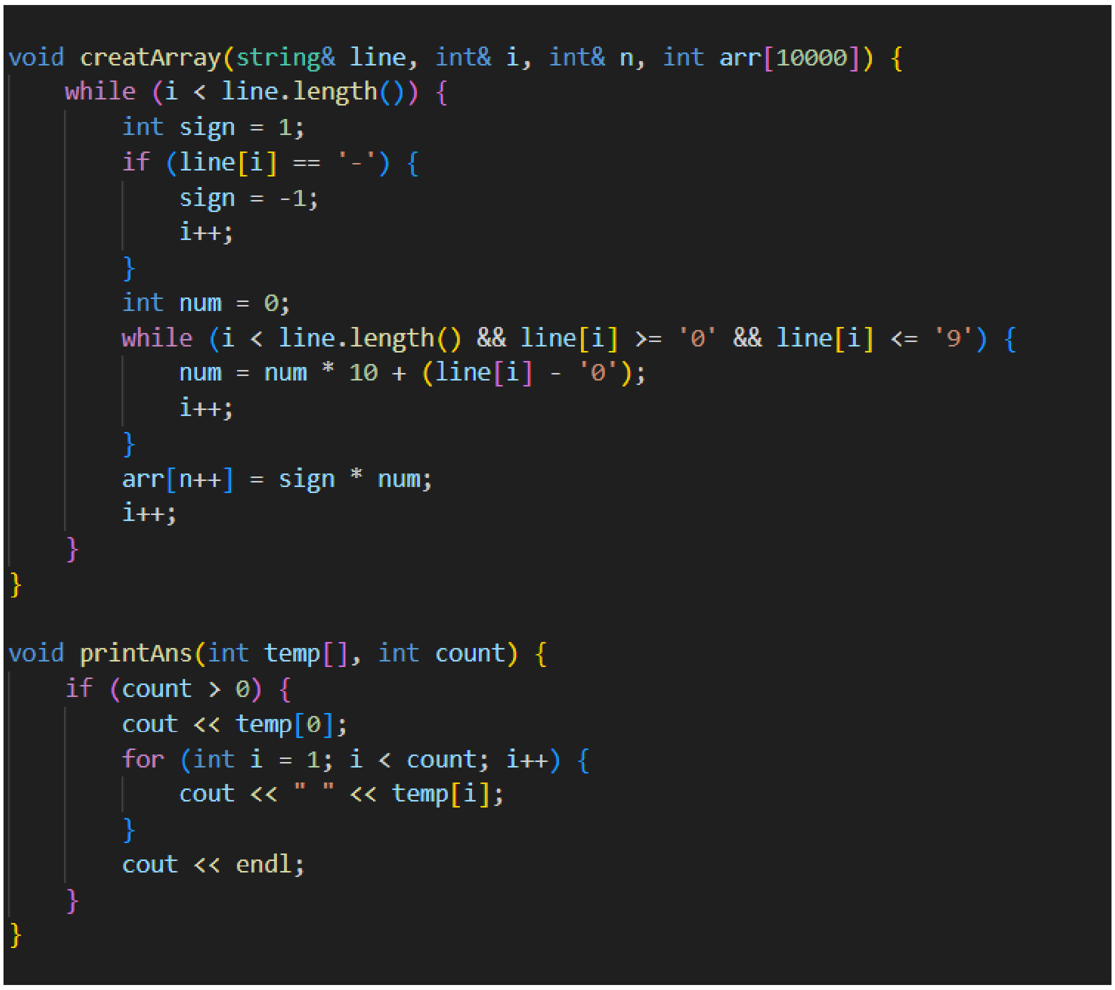
Name: TANG Pok Man

SID: 59285726

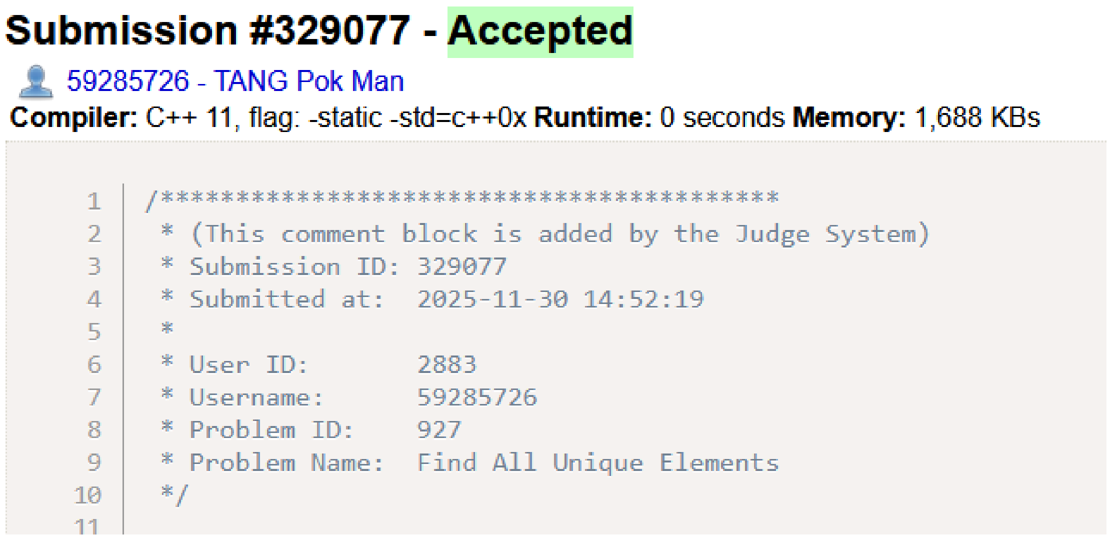
The compiler is C++ 11, flag: -static -std=c++0x.

For all implementations, there is a phasing function (creatArray) and printing function (printAns) which both time complexity is O(n).



```cpp
void creatArray(string& line, int& i, int& n, int arr[10000]) {
    while (i < line.length()) {
        int sign = 1;
        if (line[i] == '-') {
            sign = -1;
            i++;
        }
        int num = 0;
        while (i < line.length() && line[i] >= '0' && line[i] <= '9') {
            num = num * 10 + (line[i] - '0');
            i++;
        }
        arr[n++] = sign * num;
        i++;
    }
}

void printAns(int temp[], int count) {
    if (count > 0) {
        cout << temp[0];
        for (int i = 1; i < count; i++) {
            cout << " " << temp[i];
        }
        cout << endl;
    }
}
```

The first implementation is array based brute force approach. For this implementation, there is a nested for-loops to search whether there is duplicate of every number, so the time complexity is O(n^2). This is O(n^2) in worst case, even in average case, it's quadratic. If all elements are unique, it checks n(n-1)/2 pairs, and if all elements are same, it still O(n^2).



Submission #329077 - Accepted

59285726 - TANG Pok Man

Compiler: C++ 11, flag: -static -std=c++0x **Runtime:** 0 seconds **Memory:** 1,688 KBs

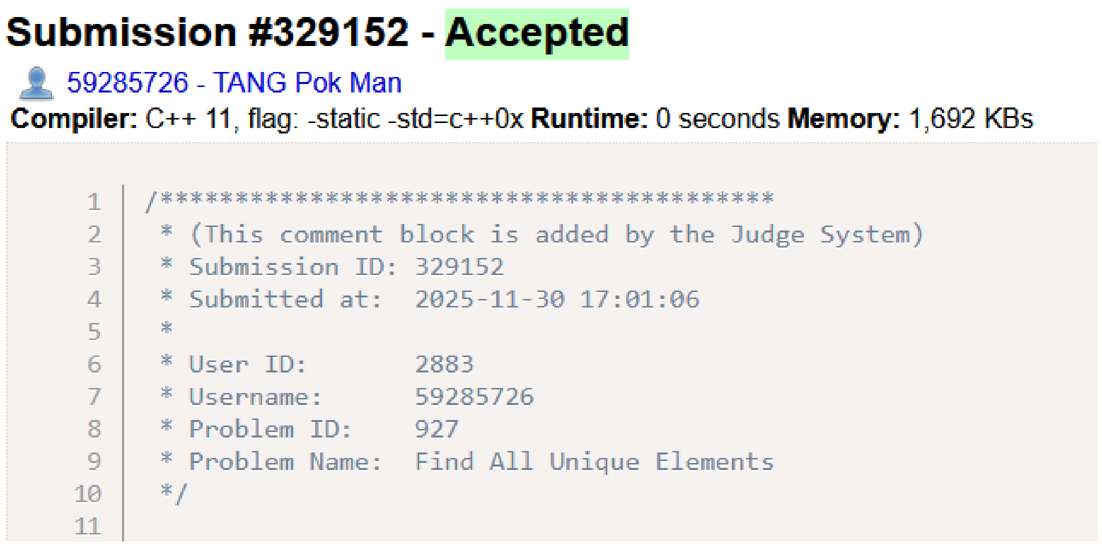
```
/*******************************************
 * (This comment block is added by the Judge System)
 * Submission ID: 329077
 * Submitted at:  2025-11-30 14:52:19
 *
 * User ID:       2883
 * Username:      59285726
 * Problem ID:    927
 * Problem Name:  Find All Unique Elements
 */

```

```
#include <iostream>

#include <string>

using namespace std;


void creatArray(string& line, int& i, int& n, int arr[10000]) {

    while (i < line.length()) {

        int sign = 1;

        if (line[i] == '-') {

            sign = -1;

            i++;

        }

        int num = 0;

        while (i < line.length() && line[i] >= '0' && line[i] <= '9') {

            num = num * 10 + (line[i] - '0');

            i++;

        }
        arr[n++] = sign * num;

        i++;

    }

}


void printAns(int temp[], int count) {

    if (count > 0) {

        cout << temp[0];

        for (int i = 1; i < count; i++) {

            cout << " " << temp[i];

        }

        cout << endl;

    }

}


void findUnique(string line) {

    int i = 0;

    int n = 0;

    int arr[10000];

    int temp[10000];

    int count = 0;

    creatArray(line, i, n, arr);


    bool visited[10000] = {false};
```

```
        for (int j = 0; j < n; j++) {
                bool repeated = false;
                if (visited[j]) {
                        continue;
                }
                for (int k = j+1; k < n; k++) {
                        if (arr[j] == arr[k]) {
                                visited[k] = true;
                                repeated = true;
                        }
                }
                if (!repeated) {
                        temp[count++] = arr[j];
                }
        }
        printAns(temp, count);
}

int main() {
        string line;
        getline(cin, line);
        while (line != "") {
                findUnique(line);
                getline(cin, line);
```

```
    }
    return 0;
}
```

The second implementation is binary search tree. For this implementation, the time complexity of insert function and search function are O(n*log(n)) in average and the worst-case scenario is O(n^2). It would happen when the numbers are sorted and the tree becomes unbalanced and performs like a linked list.



Submission #329152 - Accepted

59285726 - TANG Pok Man

Compiler: C++ 11, flag: -static -std=c++0x **Runtime:** 0 seconds **Memory:** 1,692 KBs

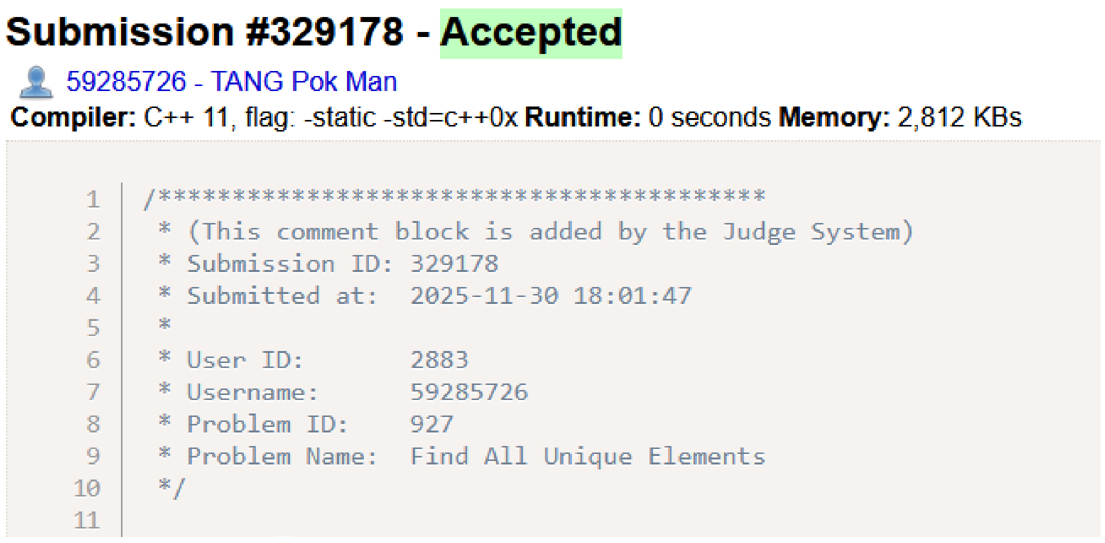
```
/******************************************
 * (This comment block is added by the Judge System)
 * Submission ID: 329152
 * Submitted at:  2025-11-30 17:01:06
 *
 * User ID:       2883
 * Username:      59285726
 * Problem ID:    927
 * Problem Name:  Find All Unique Elements
 */

```

```
#include <iostream>
#include <string>
using namespace std;

class Node {
public:
    int data;
    Node* left;
    Node* right;
    int appear;

    Node(int d) {

        data = d;

        left = nullptr;

        right = nullptr;

        appear = 1;

    }

};


class Tree {

public:

    Node* root;

    Tree() {root = nullptr;}

    void insert(int d);

    void insert(Node* node, int d);

    Node* search(Node* node, int d);

    Node* search(int d);

};


void Tree::insert(int d) {

    if (root == nullptr) {

        root = new Node(d);

    } else {

        insert(root, d);
    }
}

void Tree::insert(Node* node, int d) {
    if (d < node->data) {
        if (node->left == nullptr) {
            node->left = new Node(d);
        } else {
            insert(node->left, d);
        }
    } else if (d > node->data) {
        if (node->right == nullptr) {
            node->right = new Node(d);
        } else {
            insert(node->right, d);
        }
    } else {
        node->appear++;
    }
}

Node* Tree::search(Node* node, int d) {
    if (node == nullptr) {
        return nullptr;
    }
    if (d < node->data) {
        return search(node->left, d);
    } else if (d > node->data) {
        return search(node->right, d);
    } else {
        return node;
    }
}

Node* Tree::search(int d) {
    return search(root, d);
}

void creatArray(string& line, int& i, int& n, int arr[10000]) {
    while (i < line.length()) {
        int sign = 1;
        if (line[i] == '-') {
            sign = -1;
            i++;
        }
        int num = 0;
        while (i < line.length() && line[i] >= '0' && line[i] <= '9') {
            num = num * 10 + (line[i] - '0');
```

```
                i++;
            }
            arr[n++] = sign * num;
            i++;
        }
}

void printAns(int temp[], int count) {
        if (count > 0) {
            cout << temp[0];
            for (int i = 1; i < count; i++) {
                cout << " " << temp[i];
            }
            cout << endl;
        }
}

void findUnique(string line) {
        int i = 0;
        int n = 0;
        int arr[10000];
        int temp[10000];
        int count = 0;
        creatArray(line, i, n, arr);
```

```
    Tree myTree;
    for (int i = 0; i < n; i++) {
        myTree.insert(arr[i]);
    }
    for (int i = 0; i < n; i++) {
        Node* node = myTree.search(arr[i]);
        if (node->appear == 1) {
            temp[count++] = arr[i];
        }
    }

    printAns(temp, count);
}

int main() {
    string line;
    getline(cin, line);
    while (line != "") {
        findUnique(line);
        getline(cin, line);
    }
    return 0;
}
```

The third implementation is linked list. For this implementation, the time complexity is O(n^2) since linked list can only search linearly. The worst-case scenario is when all numbers are unique, the function needs to travel the whole linked list for all numbers.



Submission #329178 - Accepted

59285726 - TANG Pok Man

Compiler: C++ 11, flag: -static -std=c++0x **Runtime:** 0 seconds **Memory:** 2,812 KBs

```
/******************************************
 * (This comment block is added by the Judge System)
 * Submission ID: 329178
 * Submitted at:  2025-11-30 18:01:47
 *
 * User ID:       2883
 * Username:      59285726
 * Problem ID:    927
 * Problem Name:  Find All Unique Elements
 */

```

```
#include <iostream>

#include <string>

using namespace std;


class Node {

public:

    int data;

    int appear;

    Node* next;

    Node(int d, int a) {

        data = d;

        appear = a;
        next = nullptr;
    }
    Node() {
        data = 0;
        appear = 0;
        next = nullptr;
    }
};

class List {
public:
    Node* first;
    Node allNode[10000];
    int size;

    List() {
        first = nullptr;
        size = 0;
    }

    Node* search(int d);
    void insert(int d);
    void temp(int temp[], int& count);
};
```

```
Node* List::search(int d) {
    Node* curr = first;
    while (curr != nullptr) {
        if (curr->data == d) {
            return curr;
        }
        curr = curr->next;
    }
    return nullptr;
}

void List::insert(int d) {
    Node* newNode = search(d);
    if (newNode == nullptr) {
        if (first == nullptr) {
            allNode[0] = Node(d, 1);
            size++;
            first = &allNode[0];
        } else {
            allNode[size] = Node(d, 1);
            allNode[size-1].next = &allNode[size];
            size++;
        }
    } else {
        newNode->appear++;
    }
}

void List::temp(int temp[], int& count) {
    Node* curr = first;
    while (curr != nullptr) {
        if (curr->appear == 1) {
            temp[count++] = curr->data;
        }
        curr = curr->next;
    }
}

void creatArray(string& line, int& i, int& n, int arr[10000]) {
    while (i < line.length()) {
        int sign = 1;
        if (line[i] == '-') {
            sign = -1;
            i++;
        }
        int num = 0;
        while (i < line.length() && line[i] >= '0' && line[i] <= '9') {
            num = num * 10 + (line[i] - '0');
            i++;
        }
        arr[n++] = sign * num;
        i++;
    }
}

void printAns(int temp[], int count) {
    if (count > 0) {
        cout << temp[0];
        for (int i = 1; i < count; i++) {
            cout << " " << temp[i];
        }
        cout << endl;
    }
}

void findUnique(string line) {
    int i = 0;
    int n = 0;
    int arr[10000];
    int temp[10000];
    int count = 0;
    creatArray(line, i, n, arr);


    List* list = new List();
    for (int i = 0; i < n; i++) {
        list->insert(arr[i]);
    }
    list->temp(temp, count);


    printAns(temp, count);
}


int main() {
    string line;
    getline(cin, line);
    while (line != "") {
        findUnique(line);
        getline(cin, line);
    }
    return 0;
}
```