



*Erick Vargas*

# Interview notes

## Contents

<b>1</b>	<b>A little bit of theory</b>	<b>2</b>
1.1	Data structures . . . . .	2
1.1.1	¿What is a data structure? . . . . .	2
1.1.2	¿Why do we need data strucures? . . . . .	2
1.1.3	Basic data structures . . . . .	2
1.2	Arrays . . . . .	7
1.2.1	Some problems . . . . .	8
1.3	Stacks . . . . .	12
1.4	Queues . . . . .	13
1.5	Singly linked lists . . . . .	13
1.6	Doubly linked lists . . . . .	15
1.7	Graphs . . . . .	15
1.8	Trees . . . . .	15
1.9	Tries . . . . .	15

## 1.1 Data structures

### 1.1.1 ¿What is a data structure?

A simple form to describe this is: a data structure is a container to store data in a specific layout. This layout allow us to make efficient our data structure in some operations but inefficient in others.

### 1.1.2 ¿Why do we need data strucures?

Depending of different scenarios, data needs to be stored in a specific format, to solve this we need to know the different types of data structures.

### 1.1.3 Basic data structures

Nowadays there are a lot of types of data structures but here we have a little list of the commonly used data structures.

- Arrays
- Stacks
- Queues
- Linked list
- Trees
- Graphs
- Tries
- Hash tables

## Arrays

An array is the simplest used data structure and other data structures can be made using arrays, like stacks and queues. Most of the programming languages index the first element in zero.

We have two types of arrays:

- One-dimensional arrays
- Multi-dimensional arrays

## Basic operations on arrays

1. Insert: Inserts an element at given index
2. Get: returns the element at given index
3. Delete: Deletes an element at given index
4. Size: Get the total number of elements in an array

## Commonly asked array interview questions

1. Find the second minimum element of an array
2. First non-repeating integers in the array
3. Merge two sorted arrays
4. Rearrange positive and negative values in an array

## Stacks

This data structure has a lot of applications, for example when we use the undo command. A stack works like a pile of something for example books. In this example each book are placed in a vertical order. If we have a stack of books we can not get a book placed in the middle, first you need to remove all the books on the top to get it. This behaviour is known as LIFO (Last In First Out)

## Basic operations

1. Push: inserts an element of the top
2. Pop: returns the top element, after removing from the stack
3. isEmpty: returns true if the stack is empty
4. Top: returns the top element without removing from the stack

**Commonly asked stack interview questions**

1. Evaluate postfix expression using a stack
2. Sort values in a stack
3. Check balanced parentheses in an expression

**Queues**

Similar to stack, queue is another data structure that stores the element in a sequential manner. The difference is that instead of use LIFO method, queue implements FIFO method (First In First Out)

An example of a queue in real life is a line of people waiting to buy a ticket. If a new person comes, he or she will join the line from the end, not from the start and the person first person (start) will be the first to buy the ticket and then he or she leaves the line.

**Basic operations**

1. Enqueue: inserts an element at the end of the queue
2. Dequeue: removes an element from the start of the queue
3. isEmpty: returns true if the queue is empty
4. top: returns the first element of the queue

**Commonly asked queue interview questions**

1. Implement a stack using a queue
2. Reverse first k elements of a queue
3. Generate binary numbers from 1 to n using a queue

**Linked list**

A linked list is another important data structure that is very similar to an array but differs in memory allocation, internal structure and how basic operations of insertion and deletion are carried out.

A linked list is like a chain of nodes, where each node contains information like data and a pointer to the succeeding node in the chain. There's a head pointer, which points to the first element of the linked list, and if the list is empty then it simply points to null or nothing.

Linked lists are used to implement file systems, hash tables, and adjacency lists.

Also exists different types of linked lists:

- Singly linked lists
- Doubly linked lists

**Basic operations**

1. InsertAtEnd: inserts given element at the end of the linked list
2. InsertAtHead: inserts given element at the start/head of the linked list
3. Delete: deletes given element from the linked list
4. DeleteAtHead: deletes first element of the linked list
5. Search: returns the given element from a linked list
6. isEmpty: returns true if the linked list is empty

**Commonly asked linked list interview questions**

1. Reverse a linked list
2. Detect loop in a linked list
3. Return N-th node from the end in a linked list
4. Remove duplicates from a linked list

**Graphs**

A graph is a set of nodes that are connected to each other in the form of a network. Nodes are also called vertices. A pair(x, y) is called an edge, which indicates that vertex x is connected to vertex y. An edge may contain weight/cost, showing how much cost is required to traverse from vertex x to y.

Also we have different types of graphs:

- Undirected graphs  $\longrightarrow$
- Directed graphs  $\longleftrightarrow$

We can represent graphs in two forms:

- Adjacency matrix
- Adjacency list

Finally we have two common graph traversing algorithms:

- Breadth First Search (BFS)
- Depth First Search (DFS)

**Commonly asked graph interview questions**

1. Implement Breadth and Depth First Search
2. Check if a graph is tree or not
3. Count number of edges in a graph
4. Find the shortest path between two vertices

## Trees

A tree is a special type of graph the difference is that in a tree a cycle cannot exist. This data structure also use vertices (nodes) and edges that connect them.

We have different types of trees:

- N-ary tree
- Balanced tree
- Binary tree
- Binary search tree
- AVL tree
- Red-Black tree
- 2-3 tree

But the most used trees are binary tree and binary search tree.

## Commonly asked tree interview questions

1. Find the height of a binary tree
2. Find the k-th maximum value in a binary search tree
3. Find nodes at "k" distance from the root
4. Find ancestor of a given node in a binary tree

## Trie

Trie is also known as "prefix trees", is a data structure very similar to a tree which proves to be quite efficient for solving problems related to strings. It is commonly used to search words in a dictionary, providing auto suggestions in a search engine, and even for IP routing

## Commonly asked trie interview questions

1. Count total number of words in Trie
2. Print all words stored in Trie
3. Sort elements of an array using trie
4. Form words from a dictionary using trie
5. Build a T9 dictionary

## Hash table

Hashing is a process used to uniquely identify objects and store each object at some pre-calculated unique index called its "key". So, the object is stored in the form of a "key-value" pair, and the collection of such items called a "dictionary". Each object can be searched using that key. There are a lot of data structures based on hashing, but the commonly used data structure is hash table.

Hash tables are commonly implemented using arrays and the performance of hashing data structure depends of three factors:

- Hash function
- Size of the hash table
- Collision handling method

## Commonly asked hash table interview questions

1. Find symetric pairs in an array
2. Trace complete path of a journey
3. Find if an array is a subset of another array
4. Check if given arrays are disjoint

## 1.2 Arrays

We already known what is an array, and now we need to know how to solve some problems using this data structure. First of all we need to know how to use arrays in a programming language, in my case I used to use C++ language because is very easy to understand and is faster than a lot of programming languages.

In C++ we can declare an array using the next piece of code.

```
|| data_type name_of_our_array[ size_of_our_array ]
```

Imagine that you need an array of integer numbers with length ten, we need to write the next in out C++ programm.

```
|| #include <bits/stdc++.h>
||
|| using namespace std;
||
|| int main(){
||     //As you can see this is the form of create an array of
||         integers with length ten
||     int array[10];
||     return 0;
|| }
```



If we need to set a value in a element of our array we just need to put the index of our element, but consider that arrays indexes starts with 0

```
#include <bits/stdc++.h>

using namespace std;

int main(){

    int array[10];
    array[0] = 1;
    array[1] = 2;
    array[3] = 3;
    .
    .
    .
    array[9] = 10;
    return 0;
}
```

If we want to get the value of an element of our array we need just to indicate the index of our element, something like this:

```
#include <bits/stdc++.h>

using namespace std;

int main(){
    int array[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    cout << array[2] << endl;
    return 0;
}

Output: 3
```

### 1.2.1 Some problems

#### Problem 01

Find the second minimum element of an array For this problem maybe we think that sort the elements of the array can solve this problem efficiently. The complexity of do this is  $n\log(n)$  if we sort the elements using merge sort, and the code to do this (if we do not implement the merge sort) is too short, something like this:

```
#include <bits/stdc++.h>

using namespace std;

int main(){

    int n, min_01, min_02 = INT_MAX;
    vector<int> numbers;
    cin >> n;
```

```

        numbers.resize(n, 0);
        for(int i = 0; i < n; i++)
            cin >> numbers[i];

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

        min_01 = numbers[n - 1];

        for(int i = n - 2; i >= 0; i--){
            //Searching an element bigger than the minimum (it
            will be different that the minimum)
            if(min_01 != numbers[i]){
                min_02 = numbers[i];
                break;
            }
        }

        cout << ( min_02 == INT_MAX ? "none" : to_string(min_02)
            ) << endl;
        return 0;
    }

```

But, can we made something more efficient? And the answer is yes, and is very simple. We only need two variables one to know the minimum element and other to know the second minimum element (if exists). The solution is move inside the array searching by the minimum element, if we find an element smaller than the actual minimum we find a new candidate to minimum and the last value of the minimum now is the second minimum element of the array. It works but what happen if all the elements are equal? If we do this we never find a solution for this problem so we need to compare if the *i*-th element is different of the minimum current element and if this element is bigger than the second minimum element.

```

#include <bits/stdc++.h>

using namespace std;

int main(){

    int n;
    int min_01 = INT_MAX, min_02 = INT_MAX;
    vector<int> numbers;
    cin >> n;
    numbers.resize(n, 0);
    for(int i = 0; i < n; i++)
        cin >> numbers[i];

    for(int i = 0; i < n; i++){
        if(numbers[i] < min_01){
            min_02 = min_01;
            min_01 = numbers[i];
        }
        //If the elements are in ascending order and the i-
        th number is different of the minimum
        else if (numbers[i] < min_02 && numbers[i] != min_01
            )
    }

```

```

        min_02 = numbers[i];
    }

    cout << (min_02 == INT_MAX ? "none" : to_string(min_02))
    << endl;
    return 0;
}

```

## Problem 02

First non-repeating integers in the array To solve this problem we can compare each element with the rest of the array looking for the same number, if we do not find another number we find the solution, but made this is  $O(n^2)$  something like this:

```

#include <bits/stdc++.h>

using namespace std;

int main(){
    int n;

    vector<int> numbers;
    cin >> n;
    numbers.resize(n, 0);
    for(int i = 0; i < n; i++)
        cin >> numbers[i];

    for(int i = 0; i < n; i++) {
        for(int j = i + 1; j < n; j++) {
            if(numbers[i] == numbers[j])
                break;
            if(j == n - 1){
                cout << numbers[j];
                exit(0);
            }
        }
    }
    return 0;
}

```

We can made the same more efficiently if we use something else for example hasing the elements inside our array. To make this easier we use a map< key, value > which is a data structure of C++, if you want to see more about this I recommend show the reference of this data structure. For this problem our key is an integer, this integer is the i-th value of our array called numbers, and the value is the number of appearances of this number. Finally if this value is equal to one it means that we find our answer. You can see the solution in the next code:

```

#include <bits/stdc++.h>

```

```
using namespace std;

int main(){
    int n;
    vector<int> numbers;
    map<int, int> repeated;
    cin >> n;
    numbers.resize(n, 0);
    for(int i = 0; i < n; i++){
        cin >> numbers[i];

        for(int i = 0; i < n; i++){
            repeated[ numbers[i] ]++;

            for(auto e : repeated ){
                if(e.second == 1){
                    cout << e.first << endl;
                    break;
                }
            }
        }

        return 0;
    }
}
```

### Problem 03

Merge two sorted arrays Solve this problem is not difficult just you need to check which element between our two arrays is smaller than the other, and we will do this until one of our indexes is in the limit. Finally we need to check if one of our arrays was not checked completely.

```
#include <bits/stdc++.h>

using namespace std;

int main(){

    int m, n;
    vector<int> M, N, ans;

    cin >> m >> n;
    M.resize(m, 0);
    N.resize(n, 0);

    for(int i = 0; i < m; i++){
        cin >> M[i];
    }

    for(int i = 0; i < n; i++){
        cin >> N[i];
    }

    int i = 0, j = 0;
    while( i < m && j < n ){
        if(M[i] < N[j]){
            ans.push_back( M[i++] );
        } else {

```

```
        ans.push_back( N[j++] );
    }
}

for ( ; i < m; i++)
    ans.push_back( M[i] );

for( ; j < n; j++)
    ans.push_back( N[j] );

for( auto e : ans )
    cout << e << " ";

cout << endl;
return 0;
}
```

#### Problem 04

Rearrange positive and negative values in an array

## 1.3 Stacks

We already know what is a stack, but how can we made a stack using code? First of all remember that we already defined the basic operations of a Stack, so we have some clues about what we need to code.

```
struct Node{
    Node *previous;
    int value;

    Node(int _value){
        value = _value;
        previous = NULL;
    }
};

struct Stack {

    Node *top;
    int size;

    Stack(){
        top = NULL;
        size = 0;
    }

    void Push(int value){
        Node *node = new Node( value );
        if( size == 0 )
            top = node;
        else {
            node -> previous = top;
        }
    }
};
```

```
        top = node;
    }
    size++;
}

int Pop(){
    int v_return = INT_MIN;
    if( size > 0){

        v_return = top -> value;
        top = top -> previous;
        size--;

    } else {
        cout << "\nStack is empty \n";
    }

    return v_return;
}

bool IsEmpty(){
    return (size == 0);
}

int Top(){
    return top -> value;
}

int GetSize(){
    return size;
}

};
```

## 1.4 Queues

## 1.5 Singly linked lists

```
struct Node{

    Node *next;
    int value;

    Node( int _value ){
        value = _value;
        next = NULL;
    }

};

typedef struct LinkedList{

    Node *tail;
    Node *head;
```

```
int size;

LinkedList(){
    tail = head = NULL;
    size = 0;
}

void insertAtTail(int value){
    Node *node = new Node( value );
    if( size > 0 ){
        tail -> next = node;
        tail = node;
    } else {
        tail = head = node;
    }
    size++;
}

void insertAtHead(int value){
    Node *node = new Node( value );
    if( size > 0 ){
        node -> next = head;
        head = node;
    } else {
        head = tail = node;
    }
    size++;
}

void Delete(int i){
    if( i < size && i >= 0 ){
        Node *aux = head;
        //If we want to delete the first element
        if( i == 0 )
            head = head -> next;
        //If we want to delete the last element
        else if ( i == size ){
            for( int j = 0; j <= size - 1; j++ )
                aux = aux -> next;

            aux -> next = NULL;
            tail = aux;
        } else {
            for( int j = 0; j < i - 1; j++)
                aux = aux -> next;
            aux -> next = aux -> next -> next;
        }
        size--;
    }
}

void DeleteAtHead(){
    if( size == 1 )
        head = tail = head -> next;
    else
```

```
        head = head -> next;
        size--;
    }

    int Search(int i){
        if( i < size ){
            Node *aux = head;
            for(int j = 0; j < i; j++){
                aux = aux -> next;
            }
            return aux -> value;
        }

        return INT_MIN;
    }

    bool isEmpty(){
        return ( size == 0 );
    }

    int getSize(){
        return size;
    }

    void print(){
        Node *aux = head;
        while( aux != NULL ){
            cout << aux -> value << "\t";
            aux = aux -> next;
        }
        cout << endl;
    }
} LinkedList;
```

## 1.6 Doubly linked lists

## 1.7 Graphs

## 1.8 Trees

## 1.9 Tries