TODO:

* Interface
* Static
* Abstract class

**Tasks**

TwoSum

Θέλουμε να γυρνάει τα indexes του πίνακα που το άθροισμά τους δίνει το target

Input = [1, 2, 3, 4, 5, 1], Target = 2

Expected output = 0, 5

Create an empty dictionary (numToIndex)

For i from 0 to length of array:

complement = target - array[i]

If complement is in numToIndex:

Return [numToIndex[complement], i]

Store array[i] in dictionary with value i

Return empty array (no solution found)

Contains Duplicate

Create empty set

For each number in array:

If number is already in set:

Return true

Add number to set

Return false

Group Anagrams

Given an array of strings, group the anagrams together. Example: Input: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]

Create empty map: key = sorted string, value = list of original strings

For each string:

Sort the string → key

Add string to list in map[key]

Return all values in the map as a list of lists

Valid Anagram

Determine if two strings are anagrams of each other.

If lengths of s and t are different:

Return false

Create array of size 26 for letter counts

For i from 0 to length of s:

Increment count at s[i] position

Decrement count at t[i] position

If all counts are zero:

Return true

Else:

Return false

Longest Substring Without Repeating Characters

Given a string s, find the length of the **longest** **substring** without duplicate characters.

Create empty set

Initialize left = 0, maxLength = 0

For right from 0 to end of string:

While character at right is in set:

Remove character at left from set

Increment left

Add character at right to set

Update maxLength with max(maxLength, right - left + 1)

Return maxLength

Encode and Decode Strings

Encode Pseudocode:

For each string in list:

Append string length + "#" + string to result

Decode Pseudocode:

Initialize empty result list

While not at end of encoded string:

Find next '#' character

Parse number before '#' as length

Extract string of that length after '#'

Add string to result list

Return result list

Minimum Size Subarray Sum

**Problem**: Find smallest length subarray with sum ≥ target.

Initialize left = 0, sum = 0, minLen = infinity

For right from 0 to end of array:

Add array[right] to sum

While sum >= target:

Update minLen with (right - left + 1)

Subtract array[left] from sum

Increment left

If minLen was updated:

Return minLen

Else:

Return 0

Min Stack

**Problem**: Stack with constant-time min().

Create two stacks: stack and minStack

Push(val):

Push val to stack

If minStack is empty or val <= minStack.top:

Push val to minStack

Pop():

Pop value from stack

If value == minStack.top:

Pop from minStack

Top():

Return stack.top

GetMin():

Return minStack.top

Climbing Stairs

Dynamic programming / Fibonacci style.

You are climbing a staircase. It takes n steps to reach the top.  
Each time you can either climb **1** or **2** steps.  
How many **distinct ways** can you climb to the top?

**SQL QUERIES**

**Second Highest Salary**

SELECT MAX(Salary) AS SecondHighestSalary

FROM Employee

WHERE Salary < (SELECT MAX(Salary) FROM Employee);

**Department Highest Salary**

Join + subquery to get max salary per department.

SELECT Department.Name AS Department,

Employee.Name AS Employee,

Salary

FROM Employee

JOIN Department ON Employee.DepartmentId = Department.Id

WHERE (DepartmentId, Salary) IN (

SELECT DepartmentId, MAX(Salary)

FROM Employee

GROUP BY DepartmentId

);

**Problem:**

Δίνεται ένας πίνακας ARR που αναπαριστά ένα elevation map. Πρέπει να υπολογίσουμε πόσο **νερό μπορεί να "παγιδευτεί"** ανάμεσα στις μπάρες.

Η ιδέα είναι:

1. Για κάθε στοιχείο i, βρίσκουμε το μέγιστο ύψος στα αριστερά και το μέγιστο ύψος στα δεξιά.
2. Το νερό που μπορεί να παγιδευτεί στη θέση i είναι:  
   min(maxLeft[i], maxRight[i]) - ARR[i]  
   (αν αυτό το αποτέλεσμα είναι θετικό)

**Data Structures**

**Είδος Συλλογής**

* **Στατικές** (π.χ. Array) → μέγεθος ορίζεται αρχικά και δεν αλλάζει.
* **Δυναμικές** (π.χ. List<T>, Dictionary<K,V>) → το μέγεθος αυξομειώνεται δυναμικά.

**Τύπος Πρόσβασης**

* **Γραμμική πρόσβαση**: List<T>, LinkedList<T>
* **Άμεση πρόσβαση μέσω δείκτη/κλειδιού**: Dictionary<K,V>, Array, HashSet<T>

**Τρόπος Οργάνωσης Δεδομένων**

| **Δομή Δεδομένων** | **Περιγραφή** |
| --- | --- |
| Array | Στατικός πίνακας. Πρόσβαση με δείκτη. |
| List<T> | Δυναμικός πίνακας με index. |
| LinkedList<T> | Συνδεδεμένη λίστα (μονή/διπλή). |
| Stack<T> | LIFO (Last-In-First-Out). |
| Queue<T> | FIFO (First-In-First-Out). |
| Dictionary<K,V> | Ζευγάρια κλειδί-τιμή (hash-based). |
| SortedDictionary<K,V> | Ταξινομημένο λεξικό (by key). |
| SortedList<K,V> | Λίστα με ταξινομημένα κλειδιά. |
| HashSet<T> | Μοναδικές τιμές, χωρίς συγκεκριμένη σειρά. |
| SortedSet<T> | Μοναδικές και ταξινομημένες τιμές. |
| ObservableCollection<T> | Συλλογή που ειδοποιεί για αλλαγές. |

**Χρήση σε Multi-threading**

* Οι περισσότερες δομές **δεν είναι thread-safe** από μόνες τους.
* Υπάρχουν "Concurrent" εκδόσεις, π.χ. ConcurrentDictionary<K,V>, ConcurrentQueue<T> για ασφάλεια σε πολλά νήματα.

**Ειδοποίηση για Αλλαγές (Events)**

* Π.χ. ObservableCollection<T> είναι χρήσιμη στο WPF, επειδή υποστηρίζει INotifyCollectionChanged.

**Πλαίσιο χρήσης**

| **Ανάγκη** | **Κατάλληλη Δομή** |
| --- | --- |
| Γρήγορη αναζήτηση με κλειδί | Dictionary<K,V> |
| Αποθήκευση μοναδικών τιμών | HashSet<T> |
| Στοίβα / ουρά | Stack<T>, Queue<T> |
| Παρακολούθηση αλλαγών | ObservableCollection<T> |
| Ταξινομημένα δεδομένα με γρήγορη πρόσβαση | SortedList<K,V> / SortedSet<T> |

Static

The static keyword is used to declare members that belong to the **type itself rather than to a specific object**.

A static class cannot be instantiated and can contain only static members.

A method that can be called on the class itself, not on an instance.

A variable that is shared across all instances of a class.

Used to initialize static data or perform actions that need to be done only once.

Interface/Abstract

Interfaces and abstract classes are used to define contracts and provide polymorphic behavior, but they serve slightly different purposes and have different rules.

An interface defines a contract (set of method/property signatures) that a class must implement. It cannot contain implementation

An abstract class is a class that cannot be instantiated, and may contain:

* Abstract members (without implementation)
* Concrete members (with implementation)
* Fields, constructors, etc.

It serves as a partial blueprint for other classes.