Section - A

Que : Find sequence of each element in a given Array.

Int arr[] = {1,1,1,4,4,3,3,3,5,5,1,1,4,,3};

Output:

```
1 occ's at 3 times
4 occ's at 2 times
3 occ's at 3 times
5 occ's at 2 times
1 occ's at 2 times
4 occ's at 1 time
3 occ's at 1 time
```

Que : Find Frequency of each element in a given Array?

Int arr[] = {1,1,1,4,4,3,3,3,5,5,1,1,4,3};

Output:

```
1 occ's at 5 times
4 occ's at 3 times
3 occ's at 4 times
5 occ's at 2 times
```

Que : Print the following pattern using one loop.

```
*
* *
* * *
* * * *
* * * * *
```

Que : Consider an array that contain some elements(+ve,-ve) and print the second highest element in the array

Int arr[] = {65,23,4,-45,55,8,-5,-7,0,67}

Que : Print the following pattern using a given Array.

```
Int arr[] = {3, 2, -3, 5, 4, -2};
```

```
            *
            *   *
*           *   *
*   *       *   *
*   *       *   *
        *           *
        *           *
        *
```

Que : Consider an array that contains some elements and shift all the elements by two steps in the right direction.

```
Int arr[] = {3, 2, 5, 6, 1, 9, 8, 4};
```

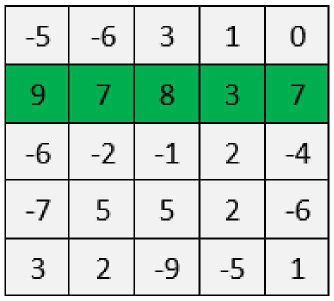
Output:

```
8, 4, 3, 2, 5, 6, 1, 9
```

Section - B

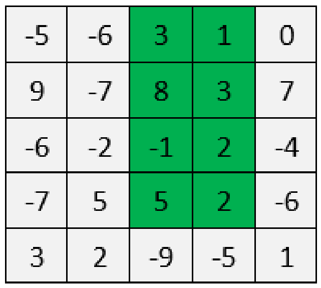
Que: Find maximum sum submatrix present in a matrix

Given an N × N matrix of integers, find the maximum sum submatrix present in it.
For example, the maximum sum submatrix is highlighted in green in the following matrices:



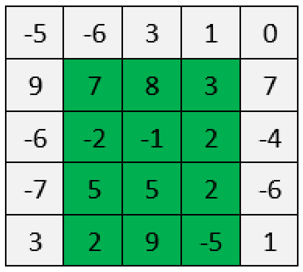
-5	-6	3	1	0
9	**7**	**8**	**3**	**7**
-6	-2	-1	2	-4
-7	5	5	2	-6
3	2	-9	-5	1

Sum = 34



-5	-6	**3**	**1**	0
9	-7	**8**	**3**	7
-6	-2	**-1**	**2**	-4
-7	5	**5**	**2**	-6
3	2	-9	-5	1

Sum = 23

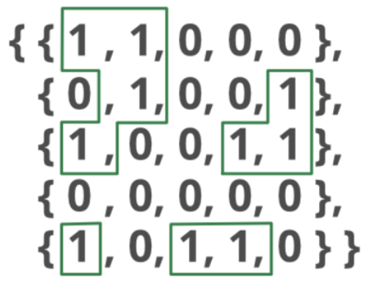


-5	-6	3	1	0
9	**7**	**8**	**3**	7
-6	**-2**	**-1**	**2**	-4
-7	**5**	**5**	**2**	-6
3	**2**	**9**	**-5**	1

Sum = 35

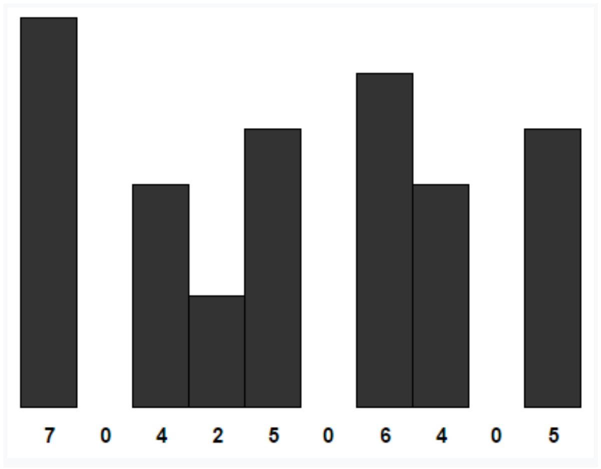
Que: Given a binary 2D matrix, find the number of islands. A group of connected 1s forms an island.
For example, the below matrix contains 4 islands.
Example**:**

> ***Input:*** *mat[][] = {{1, 1, 0, 0, 0},*
> *{0, 1, 0, 0, 1},*
> *{1, 0, 0, 1, 1},*
> *{0, 0, 0, 0, 0},*
> *{1, 0, 1, 0, 0}}*
>
> ***Output:*** *4*

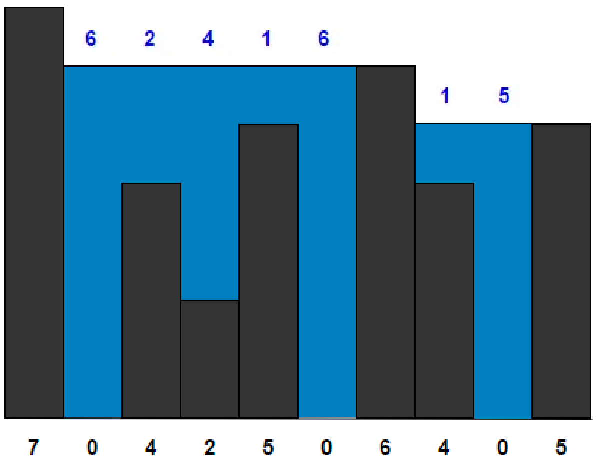


Que: Trapping rainwater problem:
Find the maximum amount of water that can be trapped within a given set of bars where each bar's width is 1 unit.
For example,

Input: An array containing height of bars {7, 0, 4, 2, 5, 0, 6, 4, 0, 5}



The maximum amount of water that can be trapped is 25, as shown below (blue).



Introduction to Node.js:
- What is Node.js?
- How does it differ from the browser's JavaScript environment?
- The event-driven, non-blocking I/O model.

Setting Up Node.js:
- Installing Node.js and npm (Node Package Manager).
- Running your first Node.js script.

Core Modules:
- Understanding the built-in modules like `fs` (File System), `http`, `https`, `util`, and `events`.
- How to use core modules to perform common tasks.

Asynchronous Programming:
- Callbacks and the callback pattern.
- Promises and async/await for managing asynchronous operations.
- Dealing with callback hell (Callback Pyramid of Doom).

File System Operations:
- Reading and writing files.
- Working with directories.
- File I/O in an asynchronous environment.

HTTP and Web Servers:
- Creating a basic HTTP server.
- Handling HTTP requests and responses.
- Building RESTful APIs with Node.js.

Intermediate Node.js:

Modules and Package Management:
- Creating and organizing your own Node.js modules.
- Using npm to manage third-party packages.
- Understanding package.json and npm scripts.

Event Emitters:
- EventEmitter class for custom event handling.
- Implementing custom event-driven patterns.

Streams and Buffers:
- Understanding streams for efficient data processing.
- Working with readable and writable streams.
- Buffer objects for handling binary data.

Error Handling:
- Dealing with errors and exceptions in Node.js.
- Using try-catch and error-first callbacks.
- Creating custom error classes.

Express.js:
- Building web applications and APIs with Express.js.
- Middleware functions and routing.
- Templating engines like EJS and Pug.

Advanced Node.js:

Authentication and Authorization:
- Implementing user authentication and authorization.
- Using libraries like Passport.js.

Database Connectivity:
- Connecting to databases (e.g., MongoDB, MySQL, PostgreSQL) using libraries like Mongoose or Sequelize.
- Performing CRUD operations.

WebSocket Communication:
- Real-time communication with WebSockets using libraries like Socket.io.

Scaling and Load Balancing:
- Strategies for scaling Node.js applications.
- Load balancing with tools like PM2.

Security Considerations:
- Best practices for securing your Node.js applications.
- Common security vulnerabilities and how to prevent them.

Testing and Debugging:
- Writing unit tests with frameworks like Mocha and Chai.
- Debugging Node.js applications using Node.js Inspector.

Deployment and DevOps:
- Deploying Node.js applications to production servers.
- Continuous integration and deployment (CI/CD) pipelines.

Performance Optimization:
- Profiling and optimizing Node.js code.
- Using performance monitoring tools.

Microservices and Containers:
- Building microservices with Node.js.
- Containerization with Docker.

Basics of Dynamic Programming:

Understanding the Concept:

- What is dynamic programming?
- When and why is it used?
- How does it differ from other problem-solving techniques?

Optimal Substructure:

- Recognizing problems with optimal substructure.
- Identifying subproblems and their relationships.

Overlapping Subproblems:

- Identifying problems with overlapping subproblems.
- How solving subproblems once can optimize the overall solution.

Top-Down vs. Bottom-Up Approach:

- The recursive (top-down) approach.
- The iterative (bottom-up) approach.

Dynamic Programming Paradigms:

Memoization:

- Storing solutions to subproblems in a cache.
- Implementing memoization using data structures like arrays or dictionaries.

Tabulation:

- Building a table to store solutions to subproblems.
- Using arrays or matrices for tabulation.

Common Dynamic Programming Problems:

Fibonacci Sequence:

- Solving the Fibonacci sequence using both recursive and dynamic programming approaches.

Coin Change Problem:

- Finding the minimum number of coins needed to make change.
- Dynamic programming solution with memoization or tabulation.

Longest Common Subsequence (LCS):

- Finding the longest common subsequence between two sequences.
- Dynamic programming approach.

Longest Increasing Subsequence (LIS):

- Finding the longest increasing subsequence in an array.
- Dynamic programming solution.

Knapsack Problem:

- Solving the 0/1 Knapsack problem for optimization.
- Dynamic programming approach.

Advanced Dynamic Programming Topics:

Matrix Chain Multiplication:
- Finding the optimal way to multiply a chain of matrices.
- Dynamic programming approach.

Edit Distance (Levenshtein Distance):
- Calculating the minimum number of edits (insertions, deletions, substitutions) to transform one string into another.
- Dynamic programming solution.

Dynamic Programming on Trees:
- Solving problems involving trees using dynamic programming.
- Examples like finding the maximum independent set in a tree.

Bitmask Dynamic Programming:
- Using bitmasks to represent states in dynamic programming problems.
- Applications in problems like the Traveling Salesman Problem.

Advanced Techniques:
- Techniques like divide and conquer optimization, convex hull trick, and others for specific problem types.

Practice and Implementation:

Coding Exercises:
- Solve a variety of dynamic programming problems on platforms like LeetCode, HackerRank, or Codeforces.

Project Work:
- Apply dynamic programming to real-world problems or personal projects.

Additional Resources:

Books and Courses:
- Consider studying from textbooks like "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein or online courses on platforms like Coursera, edX, or Udemy.

Online Communities:
- Join online programming communities and forums to discuss and learn about dynamic programming techniques and solutions.