A compiler and an interpreter are both tools used in the process of translating and executing computer programs, but they operate in different ways.

1. **Compiler:**
   * A compiler is a program that translates the entire source code of a high-level programming language into machine code or an intermediate code all at once.
   * The translation process involves multiple stages, including lexical analysis, syntax analysis, optimization, and code generation.
   * Once the compilation process is complete, the compiler produces an executable file or another form of code that can be executed independently of the original source code.
   * Examples of compiled languages include C, C++, and Java (though Java uses both compilation and interpretation through the use of the Java Virtual Machine).
2. **Interpreter:**
   * An interpreter, on the other hand, translates high-level programming code into machine code or an intermediate code line by line, executing the program as it goes along.
   * Interpretation occurs in real-time, and the source code is not translated all at once into a separate executable file.
   * Interpreters are often used in scripting languages, and they are generally slower than compiled languages because the translation happens on the fly during program execution.
   * Examples of interpreted languages include Python, JavaScript, and Ruby.

**Programming Paradigms:**

1. Procedural Programming
2. Functional Programming
3. Object-oriented Programming

**Java Programming Language:**

Java is a high-level, object-oriented programming language that was developed by Sun Microsystems (acquired by Oracle Corporation) in the mid-1990s. It is designed to be platform-independent, which means that Java programs can run on any device or operating system that has a Java Virtual Machine (JVM) installed. Here are some key features and characteristics of the Java programming language:

1. **Object-Oriented:**
   * Java follows the principles of object-oriented programming (OOP), including encapsulation, inheritance, and polymorphism. Everything in Java is treated as an object.
2. **Platform-Independent:**
   * Java programs are compiled into an intermediate form called bytecode, which can be executed on any device that has a Java Virtual Machine (JVM). This makes Java platform-independent and "write once, run anywhere" (WORA).
3. **Simple and Familiar:**
   * Java syntax is similar to other C-based languages like C++ and C#, making it relatively easy for developers to learn and adopt. It also eliminates many of the complexities found in languages like C++ (e.g., explicit pointers and memory management).
4. **Robust and Secure:**
   * Java includes features for automatic memory management (garbage collection), exception handling, and strong type-checking, which contribute to the robustness of Java programs. Additionally, Java has a strong emphasis on security, with features such as the Java Sandbox for applets.
5. **Multi-threaded:**
   * Java has built-in support for multithreading, allowing developers to write programs that can perform multiple tasks concurrently. This is crucial for developing scalable and responsive applications.
6. **Distributed Computing:**
   * Java supports the development of distributed, networked applications through its extensive libraries (like Java RMI and Java Networking). This makes it suitable for building enterprise-level and web-based applications.
7. **Rich Standard Library:**
   * Java comes with a comprehensive set of libraries (Java Standard Edition API) that provides support for networking, file I/O, database connectivity, GUI development (Swing and JavaFX), and much more. This helps developers build applications efficiently.
8. **Community Support:**
   * Java has a large and active community of developers. There are numerous resources, frameworks, and tools available, contributing to the extensive ecosystem surrounding the language.

**JVM (Java Virtual Machine):**

* The JVM, or Java Virtual Machine, is an integral part of the Java Runtime Environment (JRE). It provides a runtime environment in which Java bytecode can be executed. The primary purpose of the JVM is to abstract the underlying hardware and operating system, allowing Java programs to be platform-independent. It interprets the compiled Java bytecode or, in some cases, employs a Just-In-Time (JIT) compiler to convert bytecode into native machine code for better performance.

**JRE (Java Runtime Environment):**

* The Java Runtime Environment is a package of software that includes the Java Virtual Machine (JVM), Java class libraries, and supporting files. The JRE is what enables a Java program to be executed on a specific device or platform. It provides the necessary runtime components for running Java applications but does not include the tools and utilities needed for Java development.

**JDK (Java Development Kit):**

* The Java Development Kit is a comprehensive software development kit for building Java applications. It includes the Java Runtime Environment (JRE) along with additional tools such as compilers, debuggers, and other utilities necessary for Java development. Developers use the JDK to write, compile, and debug Java code. It includes the JRE, so if you have the JDK installed, you also have the tools needed to run Java applications.

In summary:

* **JVM** is the virtual machine that executes Java bytecode.
* **JRE** is a package that includes the JVM, Java class libraries, and supporting files for running Java applications.
* **JDK** is a development kit that includes the JRE along with tools and utilities for Java development.



**Questions -** [**Click Here**](https://chatgpt.com/c/fbfbf6e8-35a5-4252-8a4e-bc822c07d27b)

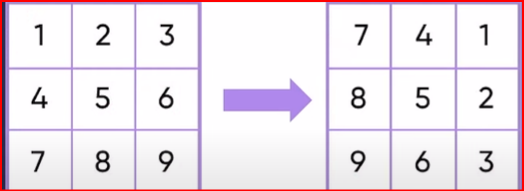
**LOOPS 🡪**

1. Print the sum of stream of integers in input.
2. Print the first multiple of 5 which is also multiple of 7.
3. Count the number of digits of given number x.
4. Find the sum of digits of a given number x.
5. Reverse the number.
6. Find the sum of the series S= 1-2+3-4+5-…. n.
7. Print factorial of n and factorial of 1 to n.
8. We have two numbers a and b. Find a raise to power b.
9. Print rectangular pattern.
10. Print only border of rectangle.
11. Print triangular pattern.
12. Print downward triangular pattern.
13. Print pyramid pattern.

**1D ARRAYS 🡪**

1. Traversing of array.
2. Calculate the sum of all element of a given array.
3. Calculate the maximum value from a given array.
4. Search the value x in a given array and if present return its index else return -1.
5. Take array as input and print it.
6. Copy an array [shallow and deep copy arr. Clone ()].
7. Count the number of occurrences of x in array.
8. Find the last occurrence of x.
9. Count the total number elements that are strictly greater than x.
10. Check if the given array is sorted or not.
11. Find the total number of pairs whose sum is equals to x.
12. Count the number of triplets whose sum is equals to x.
13. Find the unique number in the given array where all others are present twice except that one number.
14. Find second largest element in array.
15. Return the first value that is repeated if no repetition is there return -1
16. Swap two integers value using temp and without temp.
17. Reverse the array consisting of integer values.
18. Rotate the given array by k steps where k is non-negative and greater than n.
19. Solve 18 without extra space.
20. Write code for q queries on array to know the presence x using frequency array.
21. Sort an array consist of only 0s and 1s.
22. In an array put all even element first.
23. Given an array ‘a’ sorted in non-decreasing order, return an of square of all elements in non-decreasing order.
24. Find the prefix sum of a given integer array without using extra space.
25. Given an array of integer size n. Answer q queries where you have to find the sum between two indices l to r on 1 based indexing.
26. Check if we can partition an array into two sub arrays of equal sum. More formally, check that the prefix sum of a part of array is equal to the suffix sum of the rest of the array.

**2D ARRAYS 🡪**

1. Take a 2D array as input and print it.
2. Addition of two matrices.
3. Multiplication of two matrices.
4. Find the transpose of a matrix.
5. Given a square matrix turn it by 90 degrees in clockwise direction without using extra space. 
6. Given an integer n, print the n rows of pascal’s triangle.
7. Traverse a matrix in spiral order.
8. Given a positive integer n, generate a matrix of n x n filled by the number 1 to n in spiral order.
9. Given a matrix of m x n and two coordinates (l1, r1) and (l2, r2).

Find the sum of the rectangle between these two coordinates.

1. Solve 9 by prefix sum approach.

**Array List 🡪**

Wrapper Class.

Creating object of integer class which is the wrapper class of int values.

Integer i=new Integer(vale:10); another way 🡪 Integer i=Integer.valueOf(10); sout(i);

Why we use Wrapper classes?

ArrayList is in 🡪 import java.util.ArrayList; package

It is of variable size. Its does not take primitive data type. It takes wrapper classes.

To create object of ArrayList 🡪 ArrayList<Integer> arr1=new ArrayList<>();

To add element 🡪 arr1.add(5);

To get element of ith index 🡪 arr.get(1);

To get length 🡪 arr1.size();

To add some element at I index 🡪 arr1.add(index:2,element:400);

To modify element at I index 🡪 arr1.set(index:3,element:15);

To remove element of I index 🡪 arr.remove(index:1);

To remove element 🡪 arr.remove(Integer.valueOf(15));

To check existence of element 🡪 arr.contains(Integer.valueOf(15));

If we don’t specify the wrapper class of any data type we can store any value in the form of array .

1. Write a programme to reverse the given array list.

[0,10,22,10,15]. Collection.reverse(list);

1. Write a programme to sort an Array list of string in descending order.

Collection.sort(list);

Collection..sort(list , reverseOrder() );

**RECURSION 🡪**

1. Print 1 to n natural number.
2. Write a programme to find n!
3. Programme to find n Fibonacci number.
4. Given an integer n, find the sum of its digits.
5. Given two numbers p and q, find p ^ q using recursive function

(even: odd).

1. Given a number n and a value k. Find the k multiples of n.
2. Given a number n. Find sum of natural number till n but with opposite sign. 1-2+3-4+5-6…. n.
3. Find GCD of two numbers x and y using recursion. Euclid’s Algorithm.
4. Print the max value of array.
5. Find sum of all element of array recursively.
6. Given an array of integers and a target value x. Print whether x exist in this array or not.
7. Return index where x present in array q11.
8. Print all indices where c present.
9. Return array list from q13.
10. Remove all occurrence of ‘a’ from the string “abaca”.
11. Write a programme to reverse the given string using recursion.
12. Write a programme to find that a given string is palindrome or not.
13. Given a string, write a method to return all its subsequence in an Array List. A String is a subsequence of given string that is generated by deleting some character without changing its order.
14. Write a method to print all subsequence of a string.
15. Given an array of integer print sum of all its subsets. Output sum can be in any order.
16. There are n stones numbered 0 to n-1. For each i(0<=i<n) the height of stone is hi. There is a frog who is initially at stone 0. He will repeat the following action to reach the stone n-1. If the frog is currently on stone i, jump to stone i+1 or i+2. Find the minimum cost of frog to reach at the stone n-1.

**SEARCHING AND SORTING 🡪**

1. Write a programme to sort an array of integer using *BUBBLE SORT* algorithm. SC (0) and O(n^2) BC🡪O(n) also optimize it. Stable and in-place.
2. Implement *SELECTION SORT.* O(n^2) in BC, WC and AC. Not stable and in-place.
3. Implement *INSERTION SORT.* BC 🡪 O(n), WC and AC 🡪 O(n^2). Stable and in-place.
4. Given an integer array, move all zeros to the end of the it while maintaining the relative order of non-zero element without using extra space.
5. Given an array of names of fruit, you are supposed to sort it in lexicographical order using the selection sort.
6. Implement *MERGE SORT*. O(nlogn) space complexity O(n) stable, not in place.
7. Implement *QUICK SORT*.
8. Implementation of basic *COUNT SORT.*
9. Implementation of *REDIX SORT (digit by digit sorting).*
10. Implementation of *BUCKET SORT.*

***LinkedList***

**SINGLY LINKED LIST 🡪**

**BASIC QUESTIONS**

1. Create a linked list and print it using node temp in loop.
2. Display linked list by recursion.
3. Reverse linked list using recursion.
4. Write a method to find the length of linked list.
5. Insert an element at the end of list.
6. Implement LinkedList as an object as class and create various function like addAtIndex, addAtEnd, addAtStart, getSize, elementAtIndex, deleteAtIndex, etc.

**MODRATE TO ADVANCE QUESTIONS**

1. Can we delete a node that node is given as a parameter itself (delete a node) on LC.
2. Find nth node from the end of linked list only head is given.
3. Do q8 in one traversal using fast and slow pointer.
4. Find intersection of two Linked List on LC.
5. Find middle element of linked list using two pointer 1sx=2fx on LC.
6. Delete the middle element on LC.
7. Detect cycle in a linked list on LC.
8. Find out the node from which the cycle begins on LC.
9. Merge two sorted linked list with extra space and without extra space on LC.
10. Given a linked list split it into two linked list, such that one contains odd value and other contains even value. Try another version on LC.
11. Remove duplicates from the sorted list on LC.
12. Reverse a linked list and return its new head node. Do it by iterative and recursive method on LC.
13. Check whether a linked list is palindrome or not on LC.
14. Find maximum twin sum of a linked list of even length. The twin of any node (i) index is the node at (n-i-1) index. Twin sum is the sum of value of a node and its twins. Solve it on LC.
15. Odd even linked list on LC.
16. Copy list with random pointer. Construct a deep copy of linked list where each node contains an additional random pointer, which could point to any node in the list or null. LC Question.

**DOUBLY LINKED LIST 🡪**

1. Implement doubly linked list. Write function to print from head, tail and from any random node.
2. Implement doubly linked list as class and write the function for adding element and deleting element from various nodes.
3. Circular linked list: Singly and Doubly Circular Linked List.
4. Palindrome in doubly linked list.
5. Two sums in a sorted doubly linked list.

**STACKS 🡪**

1. Use inbuilt stack and perform various operation like push, pop, peek, is Empty, and size.
2. Copy content of one stack to another in same order.
3. Insert elements at bottom or at any index.
4. Implementation of Stacks using array.
5. Linked List implementation of Stacks.
6. Check whether a given sequence of brackets are balanced or not.
7. Leet code 20.
8. Given a sequence of number. Remove all the consecutive subsequence of length greater than or equals to 2 that contains the same element.
9. Next greater element using stacks.
10. GFG 🡪 Stock span problem (Hint- previous greater element).
11. Largest area Histogram. (HARD).
12. Min stack leet code
13. Infix expression evaluation in stacks.
14. Infix to Prefix expression in stacks.
15. Infix to Postfix expression in stacks.
16. Evaluation of Postfix expression.
17. Evaluation of Prefix expression using stacks.
18. Convert Prefix to Postfix expression.
19. Convert Postfix to Prefix expression.
20. Conversion of a Prefix expression to Postfix expression.
21. Conversion of Prefix to Infix Expression.
22. Conversion of Postfix to Infix expression.
23. Celebrity Problem GFG.
24. Sliding window maximum problem on LC.

**QUEUES**🡪

1. Use in built Queue and perform various operation. add (), remove (), element (), poll (), peek (), size (), is Empty ().
2. Print all the elements of a queue using helper queue.
3. Array implementation of Queue as class.
4. Linked List implementation of Queue as class.
5. Implementation of Circular Queue using array.
6. Implementation of Circular Queue using Linked List.
7. Go through with Deque (Doubly ended queue).
8. Reverse the Queue.
9. Reverse k elements from starting of Queue.
10. Implement Stack using Queue LC 225.
11. Implement Queue using Stacks LC 232.
12. First negative integer in every window of size k GFG.
13. Reordering of Queue (Interleave first half with second half) using stacks.

**HASHMAP**🡪

1. Use built in Hash-map and perform various operation like put, get, remove, contains Key, put if absent, keyset, values, entry set and traverse across the entries.
2. Given an array find the most frequent element in it. If there are multiple element that appears maximum number of times, print anyone of them.
3. Custom HashMap implementation.
4. Valid Anagram: Check whether the given two strings are anagram or not. Use space effective technique on LC.
5. Isomorphic String: Given two strings find that these are isomorphic or not on LC 05.
6. Two Sum: Given an array of length n and a target, return a pair(indices) whose sum is equals to target. If there is no pair present return -1 on LC.
7. Largest subarray with 0 sum GFG.

**HASHSET** 🡪

1. Use inbuilt HashSet and its various function like add, contains, size, remove.
2. Given an unsorted array of integers n, return the length of longest consecutive element sequence on LC 128.

**Back-Tracking**🡪

1. Rat in a maze. Find the total number of ways we have only two directions left and down.
2. Rat in a maze. Print the different ways.
3. Rat in a maze of four direction.
4. Rat in a maze of four direction with some blockage or dead block.
5. Permutations LC 46.
6. N Queen problem on LC 51.
7. Check knight tour configuration LC 2596.
8. Place k knights such that they do not attack each other.
9. Valid Sudoku LC 36.
10. Sudoku Solver LC 37.

**BINARY TREES**🡪

1. General implementation of binary tree and write display function to print in preorder using recursion.
2. Find the sum of all nodes, size of tree, and height of binary tree.
3. Perform preorder, in order and post order traversal.
4. Find maximum, minimum value of tree and product of tree.
5. Print nth level of tree.
6. Level order traversal (BFS).
7. BFS using Queue data structure iterative method.
8. DFS tree traversal using Stack data structure iterative method LC 94.
9. Diameter of binary tree LC 543.
10. Balance binary tree LC 110.
11. Same tree LC 100.
12. Binary tree paths LC 257.
13. Lowest common ancestor of binary tree LC 236.
14. Invert binary tree LC 226.
15. Symmetric tree LC 101.
16. Zigzag level order traversal LC 103.
17. Level order traversal LC 102.
18. Construct or implementation of tree using level order traversal.
19. Print boundary element of binary tree.
20. Binary tree right side view LC 199.
21. Construct a binary tree from a given preorder and in order traversal LC 105.
22. Path sum LC 112.
23. Path sum LC 113.

**BINARY SEARCH TREE🡪**

1. Search in a BST LC 700.
2. Insertion in a BST LC 701.
3. Deletion of a node in BST LC 450.
4. Lowest common ancestor BST LC 235.
5. In order predecessor and successor of a BST.
6. Validate a binary tree as it is BTS or not. (If in order traversal of any binary tree gives a sorted array the it will be a BST) LC 98.
7. Convert a sorted array into a BST LC 108.
8. Kth smallest number in a BST LC 230.

**GENRIC TREE, HEAP TREE & PRIORITY QUEUE** 🡪

**GRAPH** 🡪

1. Graph implementation by Array of Array List.
2. Print neighbours of edges.
3. Create edge weighted graph.
4. Implement graph by adjacency matrix.
5. Disadvantages of graph by adjacency matrix over adjacency list.
6. BFS of graph using queue data structure.
7. DFS of graph.
8. All path from source to destination.