Task 1:

What are Data Structures?

A data structure is simply a specific way of organizing data in a computer's memory so that we can use that data quickly and effectively.

It’s not just about storing the data, it’s about how it's arranged and connected. This arrangement determines what you can do with the data and how fast you can do it.

For example:

Imagine our bedroom closet. That's where we store our clothes (your data).

Bad Organization: We just throw all our clothes—shirts, socks, pants, jackets—into one giant pile on the floor.

Problem: When we need to find our favorite t-shirt, we have to dig through the entire messy pile. It's slow, frustrating, and a total mess.

Good Organization: We use tools to organize our clothes.

We use a hanger for our shirts. drawer for our socks, shelves for our sweaters.

These tools—hangers, drawers, shelves—are your Data Structures. They are different ways to organize your clothes (your data) to make your life easier. Each one is good for a specific type of clothing.

Why should we use data structures?

To Be FAST (Efficiency) - Imagine you're building a contact list app for a phone. You have 1,000 contacts. You want to find your friend "Maria."

-A simple list: If you store the contacts in a simple list, the computer might have to check every name one by one: "Is this Maria? No. Is this Maria? No..." This could take a while.

-A Hash Table: This is like a real-life address book with A-Z tabs. You can jump straight to the "M" section and find "Maria" almost instantly. This data structure is built for super-fast lookups.

To Solve Specific Problems - Different problems need different "shapes" of data.

To Manage Your Data Easily - Data structures give us simple commands to handle complex operations.

Task 2:

What are the types of data structures you know .. list them out..

there are 2 types:

Linear: Data is arranged in a sequence, one after the other, like houses on a single street.

\*Array

\*Linked List

\*Stack

\*Queue

Non-Linear: Data is arranged in a more complex, multi-level way, like a map of a whole city with interconnected streets.

Tree

Graph

A Special Case: Hash Table

Task 3:

What all operations can we do in Data structures?

The 4 Core Operations (The "CRUD" of Data Structures)

Traversing (or Iterating)

Inserting (or Adding)

Deleting (or Removing)

Searching

Sorting

Merging

Updating

Task 4:

What are static and dynamic arrays? Explain or summarize key points in a table like

Size, performance, memory, flexibility, limitations

|  |  |  |
| --- | --- | --- |
|  | Static Array | Dynamic Array |
| Size | Fixed. The size is set when you write the code and cannot be changed. | Changeable (Dynamic). The size can grow or shrink as you add or remove elements while the program runs. |
| performance | Extremely fast. Memory is allocated once. Accessing any element is instant because its location is predictable. No surprise delays. | Very fast on average. Accessing elements is also instant. However, there is a small performance cost (a brief slowdown) whenever the array needs to be resized. |
| memory | Memory efficient (if sized correctly). You use exactly the amount of memory you ask for. No more, no less. | Can have wasted space. It might allocate more memory than it's currently using (its "capacity") to avoid resizing too often. For example, it might hold 10 items but have space for 16. |
| flexibility | Inflexible. You cannot add more elements than the declared size. | Very flexible. This is its main advantage. You can add as many elements as you need (within the limits of your computer's memory). |
| Limitations | Risk of Overflow: If you try to add an item beyond its size, the program will crash or behave unexpectedly.  Risk of Waste: If you declare a huge array but only use a small part of it, you've wasted memory. | Resizing Overhead: The process of creating a new array and copying elements can be slow if the array is massive. This can cause a noticeable pause in performance-critical applications. |

Task 5:

The ASCII value for the character 'a' is 97.

this is represented as: 01100001

their "place values" are based on powers of 2: 1, 2, 4, 8, 16, 32, 64, 128, and so on.

2|97

2|48 -1

2|24 -0

2|12 -0

2|6 -0

2|3 -0

2|1 -1

Task 6:

Types of Computer memory with examples.. Explain

\*Primary Memory:This is the memory that the CPU can access directly and very quickly. It's also called main memory. Most of it is volatile, meaning it forgets everything when the power is turned off.

-RAM (Random Access Memory) : This is the main "workbench" for our computer. When we open an application (like a web browser or a game), it gets loaded from the slow filing cabinet (our hard drive) onto our super-fast desk (the RAM) so we can work with it instantly.

Volatile: When we shut down the computer, everything in RAM is erased.

Fast: It's thousands of times faster than secondary memory.

Random Access: The CPU can jump to any piece of data in RAM directly, just as we can grab any paper on our desk without having to shuffle through all the others.

Cache Memory : A smaller, even faster memory that is built right into or very close to the CPU. It acts as a buffer for RAM. It stores the data and instructions that the CPU uses most frequently.

Extremely Fast: The fastest memory in the computer.

Very Small: Much smaller than RAM (measured in megabytes or even kilobytes).

Volatile: Also erased when the power is off.

ROM (Read-Only Memory):A type of memory that holds instructions that are permanent and cannot be changed by the user.

Non-Volatile: The information stays forever, even with no power.

Read-Only: Its contents are set at the factory.

\*Secondary Memory:This is where data is stored for the long term. It's non-volatile, meaning it holds onto its data even when the power is off. It's much slower than primary memory but can hold vastly more information.

HDD (Hard Disk Drive):A traditional storage device that uses spinning magnetic disks (platters) and a moving read/write head to store data.

Slower: The mechanical moving parts make it slower.

Large Capacity: Can store huge amounts of data (terabytes).

Cheap: Offers the most storage for the lowest cost.

-SSD (Solid-State Drive):

-Flash Memory:

Task 7:

 Reverse an array. write a code.

Hint : take a list of nos and display in reverse order..

public class DisplayInReverse {

public static void main(String[] args) {

int[] numbers = {10, 20, 30, 40, 50};

System.out.println("Original Order:");

for (int i = 0; i < numbers.length; i++) {

System.out.print(numbers[i] + " ");

}

System.out.println("\n\nReverse Order (The Countdown!):");

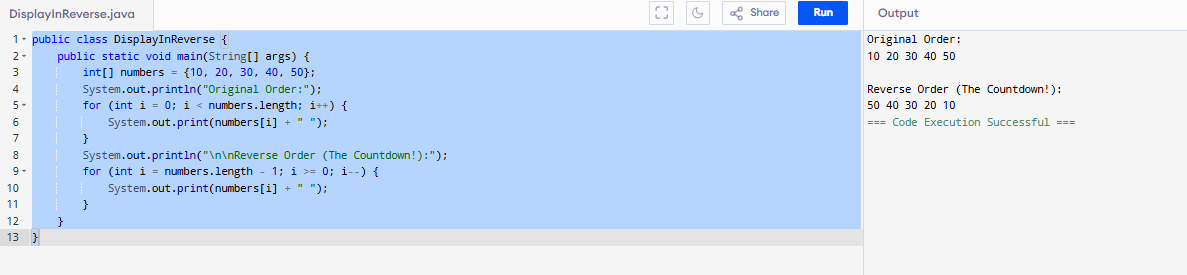
for (int i = numbers.length - 1; i >= 0; i--) {

System.out.print(numbers[i] + " ");

}

}

}



Task 8:

Reverse a string .. write a code.

Hint: take a name from the user and display the name in reverse order..

import java.util.Scanner;

public class ReverseNameSimple {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Please enter your name: ");

String originalName = scanner.nextLine();

String reversedName = "";

System.out.println("----------- The Countdown Begins! -----------");

for (int i = originalName.length() - 1; i >= 0; i--) {

char letter = originalName.charAt(i);

reversedName = reversedName + letter;

System.out.println("Current letter: " + letter + " | Reversed name so far: " + reversedName);

}

System.out.println("-------------------------------------------");

System.out.println("\nOriginal name: " + originalName);

System.out.println("Name in reverse: " + reversedName);

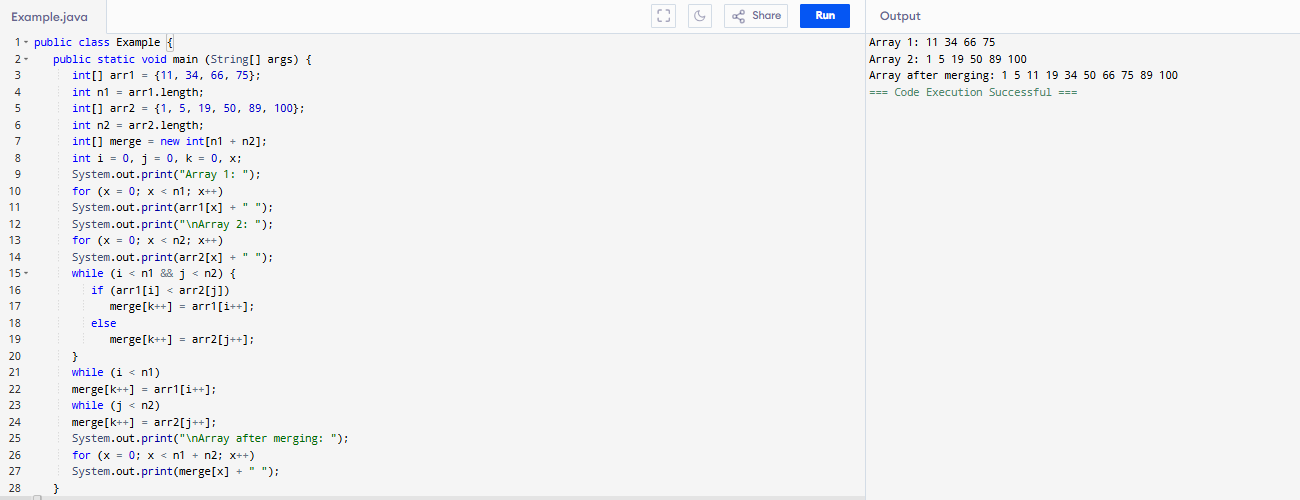
scanner.close();

}

}



Task 10:



Task11:

A hash table is a data structure that stores data in (key, value) pairs. A hash function is used to map each key to an index in an array. A hash table offers very fast access, insertion and deletion operations.

Task12:

import java.util.\*;

public class Task12 {

public static void main(String[] args) {

Hashtable<String, Integer> ht = new Hashtable<>();

ht.put("Raghu", 101);

ht.put("Vasu", 102);

ht.put("Meera", 103);

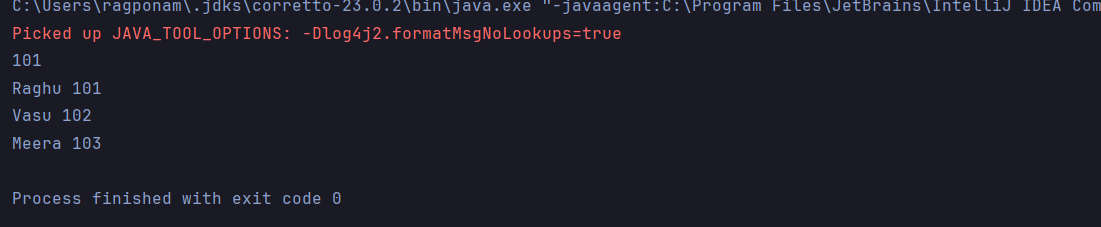
System.*out*.println( ht.get("Raghu"));

for (Map.Entry<String, Integer> e : ht.entrySet())

System.*out*.println(e.getKey() + " " + e.getValue());

}

}



Task13:

import java.util.HashMap;

import java.util.Map;

public class Task13 {

public static void main(String[] args) {

Map<String, Integer> hashMap = new HashMap<>();

hashMap.put("Raghu", 101);

hashMap.put("Vasu", 102);

hashMap.put("Meera", 103);

for (Map.Entry<String, Integer> entry : hashMap.entrySet()) {

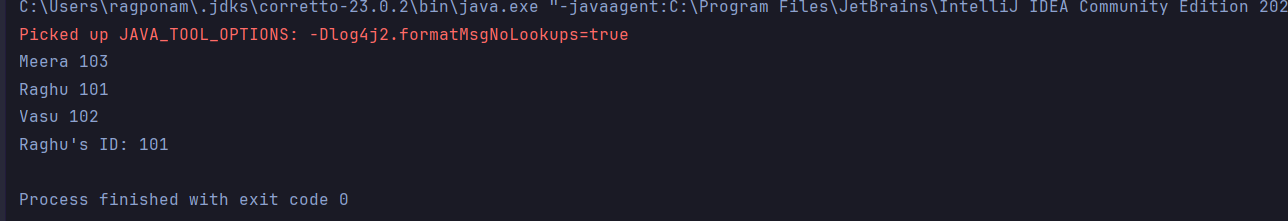
System.*out*.println(entry.getKey() + " " + entry.getValue());

}

System.*out*.println("Raghu's ID: " + hashMap.get("Raghu"));

}

}



Task14:

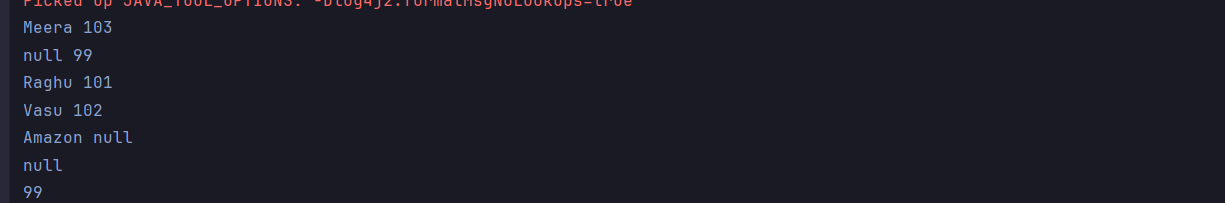
* Hash table is an abstract data structure, whereas hash map is an implementation of hash table.
* Hash tables are thread safe whereas, hash maps are not.
* Hash maps allow one null key and multiple null values, whereas hash tables do not allow null in either keys or values.
* Hash maps are faster as they are asynchronous where as hash tables are slower due being synchronous.

Task15:

public class Task15<Key, Value> {  
 private class HashTableNode {  
 private Key key;  
 private Value value;  
 private boolean active;  
 private boolean tombstoned; // Allow reuse of removed slots  
 public HashTableNode() {  
// All nodes in array will begin initialized this way  
 key = null;  
 value = null;  
 active = false;  
 tombstoned = false;  
 }  
 public HashTableNode(Key initKey, Value initData) {  
 key = initKey;  
 value = initData;  
 active = true;  
 tombstoned = false;  
 }  
 }  
  
 private final static int *TABLE\_SIZE* = 9;  
 private Object[] table;  
 public Task15() {  
// Since HashNodeTable has generics, we can not have  
// a new HashNodeTable[], so use Object[]  
 table = new Object[*TABLE\_SIZE*];  
 for (int j = 0; j < *TABLE\_SIZE*; j++)  
 table[j] = new HashTableNode();  
 }  
}

Task16:

import java.util.HashMap;  
import java.util.Map;  
  
public class Task16 {  
 public static void main(String[] args) {  
 Map<String, Integer> hashMap = new HashMap<>();  
  
 hashMap.put("Raghu", 101);  
 hashMap.put("Vasu", 102);  
 hashMap.put("Meera", 103);  
 hashMap.put("Amazon", null); //prints null when input Amazon  
 hashMap.put(null, 99);//prints 99 when input null  
 for (Map.Entry<String, Integer>entry: hashMap.entrySet()) {  
 System.*out*.println(entry.getKey()+" "+entry.getValue());  
 }  
 System.*out*.println(hashMap.get("Amazon"));  
 System.*out*.println(hashMap.get(null));  
  
  
 }  
}



Hometask:

import java.util.Arrays;  
  
public class Hometask {  
  
 public static void main(String[] args) {  
  
 int[] arr1 = {75, 11, 34, 66};  
 int n1 = arr1.length;  
  
 int[] arr2 = {100, 5, 1, 89, 19, 50};  
 int n2 = arr2.length;  
  
 Arrays.*sort*(arr1);  
 Arrays.*sort*(arr2);  
  
 int[] merge = new int[n1 + n2];  
  
 int i = 0, j = 0, k = 0, x;  
  
 System.*out*.print("Array 1 (Sorted): ");  
 for (x = 0; x < n1; x++)  
 System.*out*.print(arr1[x] + " ");  
  
 System.*out*.print("**\n**Array 2 (Sorted): ");  
 for (x = 0; x < n2; x++)  
 System.*out*.print(arr2[x] + " ");  
  
 while (i < n1 && j < n2) {  
 if (arr1[i] < arr2[j])  
 merge[k++] = arr1[i++];  
 else  
 merge[k++] = arr2[j++];  
 }  
  
 while (i < n1)  
 merge[k++] = arr1[i++];  
  
 while (j < n2)  
 merge[k++] = arr2[j++];  
  
 System.*out*.print("**\n**Array after merging (Sorted): ");  
 for (x = 0; x < n1 + n2; x++)  
 System.*out*.print(merge[x] + " ");  
 System.*out*.println();  
 }  
}

