**Task 1:**

**What is the binary 8 bit representation of A?**

ASCII value of A is 65

65/2=1

32/2=0

16/2=0

8/2=0

4/2=0

2/2=0

1/2=1

It will be written from bottom to top A==  **01000001**

**Task 2:**

**What is the binary value of a?**

ASCII code for a is 97

97/2= 1

48/2= 0

24/2= 0

12/2= 0

6/2= 0

3/2= 1

1/2= 1

It will be written from bottom to top a==  **01100001**

**Task 3:**

**Types of Computer memory with examples.. Explain ..**

It is classified into

1. Primary memory; Used by the system to store data temporarily while programs are running. It's volatile (data is lost when power is off).

* RAM(Random Access Memory)- Temporary storage for running programs and processes Ex DDR4 RAM, LPDDR5 RAM
* ROM (Read-Only Memory)- Permanent storage for firmware and bootloader Ex BIOS ROM, EEPROM

1. Secondary Memory (Storage Devices) : Non-volatile storage used for long-term data storage.

* HDD (Hard Disk Drive) : Magnetic storage with large capacity, slower speed Ex 1TB Seagate HDD
* SSD(Solid State Drive) : Faster, flash-based storage Ex 512GB Samsung SSD
* Optical Discs Uses lasers to read/write data Ex CD, DVD, Blu-ray
* Flash Drives Portable, USB-based storage Ex 64GB SanDisk Pendrive
* Memory Cards Used in mobile devices and cameras Ex 32GB microSD card

1. Cache Memory Very fast memory located close to the CPU to speed up processing.

* L1 Cache Closest to CPU core, smallest and fastest Built into CPU
* L2 Cache Slightly larger, slower than L1 Shared in CPU
* L3 Cache Shared among CPU cores, larger Intel i7 L3 cache

1. Other Types of Memory

* Virtual Memory - Uses part of HDD/SSD as RAM when RAM is full Ex Pagefile in Windows
* Registers - Smallest and fastest memory, inside CPU Ex Program Counter, Accumulator
* Buffer/Cache (Disk, Web) - Temporary storage for quick access to frequently used data ex Browser cache, Disk buffer

**Task 4:**

**What do you understand by data structures..?**

A data structure is like a **container** that helps you store and manage data efficiently depending on our needs.

**Task 5:**

**What are the operations on data structures ?**

| **Operation** | **Description** | **Example** |
| --- | --- | --- |
| Insertion | Add new data | arr[2] = 50 |
| Deletion | Remove existing data | removeNode(3) |
| Traversal | Visit each element | for loop on array |
| Searching | Find a specific data | binarySearch(arr, x) |
| Sorting | Arrange data | Arrays.sort(arr) |
| Updation | Modify existing data | arr[1] = 99 |
| Merging | Combine two structures | merge(list1, list2) |

**Task 6:**

**What are static and dynamic arrays key points summarize in a table**

**Size, performance, memory, flexibility**

| **Feature** | **Static Array** | **Dynamic Array** |
| --- | --- | --- |
| **Size** | Fixed at compile time | Can grow or shrink at runtime |
| **Performance** | Faster (no resizing overhead) | Slightly slower (resizing may occur) |
| **Memory** | Allocates memory once, may waste unused space | Efficient memory usage but overhead on resizing |
| **Flexibility** | Less flexible (cannot change size) | Highly flexible (can adjust size as needed) |

**Task 7:**

**Reverse an array. write a code**

import java.util.ArrayList;

import java.util.Collections;

public class ReverseList {

public static void main(String[] args) {

// Creating and adding elements to the list

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

numbers.add(40);

numbers.add(50);

// Display original list

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

System.out.println(numbers);

// Reverse the list using Collections utility

Collections.reverse(numbers);

// Display reversed list

System.out.println("Reversed List:");

System.out.println(numbers);

}

}

**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 ReverseString {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Take input from user

System.out.print("Enter your name: ");

String name = sc.nextLine();

// Reverse the string manually

String reversed = "";

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

reversed += name.charAt(i);

}

// Display the reversed name

System.out.println("Reversed Name: " + reversed);

sc.close();

}

}

**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 ReverseString {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Take input from user

System.out.print("Enter your name: ");

String name = sc.nextLine();

// Reverse the string manually

String reversed = "";

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

reversed += name.charAt(i);

}

// Display the reversed name

System.out.println("Reversed Name: " + reversed);

sc.close();

}

}

**Task 10:**

**public class Example {**

**public static void main (String[] args) {**

**int[] arr1 = {11, 34, 66, 75};**

**int n1 = arr1.length;**

**int[] arr2 = {1, 5, 19, 50, 89, 100};**

**int n2 = arr2.length;**

**int[] merge = new int[n1 + n2];**

**int i = 0, j = 0, k = 0, x;**

**System.out.print("Array 1: ");**

**for (x = 0; x < n1; x++)**

**System.out.print(arr1[x] + " ");**

**System.out.print("\nArray 2: ");**

**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("\nArray after merging: ");**

**for (x = 0; x < n1 + n2; x++)**

**System.out.print(merge[x] + " ");**

**}**

**}**

**What is the above code snippet doing..?**

This code merges two already sorted arrays into one sorted merged array using the efficient merge step of merge sort.

**Task 11:**

**What do you know about hash table?**

A Hash Table is a data structure that stores data in a key-value pair format, allowing for fast insertion, deletion, and lookup operations. We can also say that it is like a super-fast dictionary that lets us to look up, add, or remove data instantly by using a unique key for each piece of data

| **Feature** | **Description** |
| --- | --- |
| **Speed** | Fast O(1) access time (average case) |
| **Key-based** | Uses keys to access values |
| **Efficient** | Ideal for search-heavy operations |
| **Collision Handling** | Deals with multiple keys mapping to same index |

**Task 12:**

**Wap to create a hash table and display them..**

**Hint 👍**

**Import java.util.Hashtable;**

**Import java.util.Map;**

import java.util.Hashtable;

import java.util.Map;

public class HashTableDemo {

public static void main(String[] args) {

// Create a Hashtable

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

// Add key-value pairs to the hashtable

ht.put(101, "Ali");

ht.put(102, "Bobby");

ht.put(103, "Charlie");

ht.put(104, "David");

// Display the contents of the hashtable

System.out.println("Hashtable contents:");

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

System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());

}

}

}

**Task 13:**

**Wap to create a hash map and display them..**

import java.util.HashMap;

import java.util.Map;

public class Task013\_DS\_HashMap {

public static void main(String[] args) {

HashMap<String, Integer> hm = new HashMap<>();

hm.put("Anitha", 101);

hm.put("Kavitha", 102);

hm.put("Meera", 103);

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

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

}

}

**Add object 2 to the above code**

import java.util.HashMap;

import java.util.Map;

public class Task013\_DS\_HashMap {

public static void main(String[] args) {

HashMap<String, Integer> studentIDs = new HashMap<>();

studentIDs.put("Anitha", 101);

studentIDs.put("Kavitha", 102);

studentIDs.put("Meera", 103);

HashMap<String, String> attendance = new HashMap<>();

attendance.put("Anitha", "Present");

attendance.put("Kavitha", "Absent");

attendance.put("Meera", "Present");

System.out.println("Student Details with Attendance:");

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

String name = entry.getKey();

Integer id = entry.getValue();

String status = attendance.get(name);

System.out.println("Name: " + name + ", ID: " + id + ", Attendance: " + status);

}

}

}

**Task 14:**

**Hash table advantages and disadvantages**

Advantages of Hash Table

* **Fast Access** Time Average case: O(1) time complexity for insert, delete, and search.
* **Efficient for Lookups** Best for scenarios where frequent key-based access is needed (e.g., caching, dictionaries).
* Handles Large Datasets Well Can store a large number of key-value pairs efficiently.
* **Flexible Key Types** Can use many object types as keys (e.g., String, Integer) if they implement proper hashCode() and equals().
* **Easy to Implemen**t Standard in most programming languages with built-in libraries (HashMap in Java, dict in Python).
* **Supports Deletion and Update** Existing entries can be easily modified or removed.

Disadvantages of Hash Table

* **No Order Guaranteed** Does not maintain any order of keys or values.
* **Hash Collisions** Multiple keys may map to the same index, requiring collision resolution (e.g., linear probing, chaining).
* **Performance Drops if Poor Hash Function**Bad hash functions lead to clustering and performance degradation.
* **Memory Overhead**May use more memory due to extra space for empty or tombstoned slots.
* **Not Suitable for Range Queries** Cannot perform ordered operations like finding min, max, or sorting.
* **Thread Safety Issue** Not thread-safe by default (HashMap in Java is not synchronized; use Hashtable or ConcurrentHashMap if needed).
* **Complex Deletion Logic** Deletion may require rehashing or marking slots (in open addressing).

**Task 16:**

**Methods of Hash table plz list them.. No rating Task**

Hash table methods List .. for your ref..

* put(K key, V value): Inserts a key-value mapping into the Hashtable. If the key already exists, the old value is replaced with the new one.
* get(Object key): Returns the value associated with the specified key. Returns null if the key is not found.
* remove(Object key): Removes the key-value mapping for the specified key from the Hashtable.
* containsKey(Object key): Returns true if the Hashtable contains a mapping for the specified key, otherwise returns false.
* containsValue(Object value): Returns true if the Hashtable maps one or more keys to the specified value, otherwise returns false.
* isEmpty(): Returns true if the Hashtable contains no key-value mappings, otherwise returns false.
* size(): Returns the number of key-value mappings in the Hashtable.
* clear(): Removes all key-value mappings from the Hashtable.
* keySet(): Returns a Set view of the keys contained in the Hashtable.
* values(): Returns a Collection view of the values contained in the Hashtable.
* entrySet(): Returns a Set view of the key-value mappings contained in the Hashtable.
* rehash(): Increases the size of the Hashtable and rehashes all of its keys. This method is protected and typically handled internally by the Hashtable for performance optimization.
* clone(): Returns a shallow copy of the Hashtable instance.

**Task 17:**

**In Task 13 of hash Map .. we were using string and integer / integer and string**

**Like HashMap<Integer, String>**

**Can you change to String and string and c if it works**

**Like this HashMap<String, String> ? will this work?**

import java.io.\*;

import java.util.\*;

class Task013\_DS\_HashMap {

public static void main(String args[]) {

HashMap<String, String> hmobj1 = new HashMap<>();

HashMap<String, String> hmobj2 = new HashMap<String, String>();

hmobj1.put("Id10", "Anitha");

hmobj1.put("Id20", "Saritha");

hmobj1.put("Id30", "Ankitha");

hmobj2.put("Emp44", "John");

hmobj2.put("Emp55", "Steve");

hmobj2.put("Emp66", "Jack");

System.out.println("Mapping HashMap hmobj1: " + hmobj1);

System.out.println("Mapping HashMap hmobj2: " + hmobj2);

}

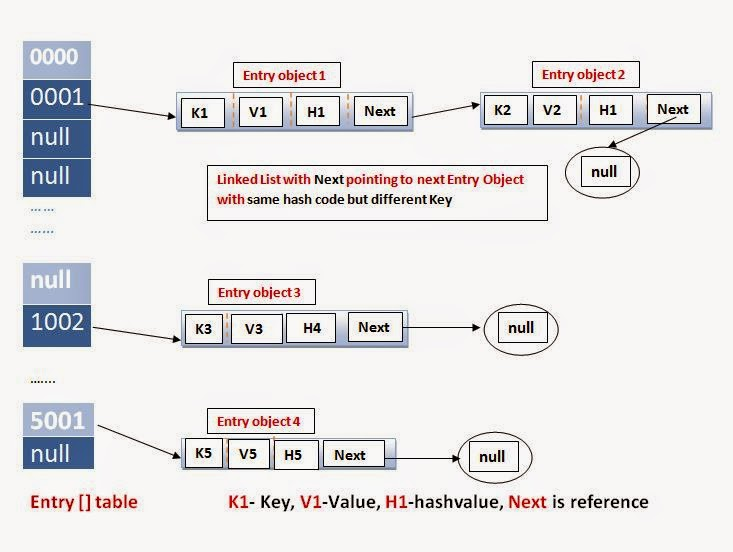
}

**Task 18:**

### **Explain the internal working of a HashMap. With diagram..**

HashMap is a data structure that stores key-value pairs and uses hashing to quickly retrieve values based on keys.

* HashMap internally uses an array of linked lists to store key-value pairs.
* When a key-value pair is added, the key is hashed to determine the index in the array where the pair will be stored.
* If multiple keys hash to the same index, a collision occurs and the pairs are stored in a linked list at that index.
* To retrieve a value, the key is hashed again to find the index and then the linked list is traversed to find the matching key.
* HashMap in Java uses an array of Entry objects, where each Entry contains the key, value, and a reference to the next Entry in case of collisions.



**Task 19:**

**Try to add 1 null value in the key and run the hash map code..**

**Also add one more null value to the key and see the result..**

import java.util.\*;

import java.io.\*;

public class Task019\_Ds\_HashMapNull {

public static void main(String[] args) { HashMap<Integer, String> hmap = new HashMap<>();

hmap.put(101,"Prasunamba");

hmap.put(null,"Meher");

hmap.put(null,".MK");

System.out.println(hmap);

}

}

**Task 20:**

**How many methods are there to create a hash Map?**

There are multiple ways to create a Hash Map in java depending on our needs like such as specifying initial capacity, using another map, or creating immutable maps.

1) Constructing a hashmap with default capacity

ex:

HashMap<String, Integer> hm1 = new HashMap<String, Integer>();

2) Constructing a hashmap with a capacity 10

ex:

HashMap<String, Integer> hm2 = new HashMap<String, Integer>(10);

3)copy one map to another map

ex:

HashMap<String, Integer> hm3 = new HashMap<String, Integer>( hm2);

4)

Specifying load factor along with the capacity

ex:

HashMap<String, Integer> hm4= new HashMap<String, Integer>(10, 0.75f);

Initial capacity ===10

Load factor === 0.75f

**Task 21**

**Wap to make a Hashmap synchronized..**

**Plz note : Hash Map are - asynchronous in nature..**

import java.util.Collections;

import java.util.HashMap;

import java.util.Map;

public class task021\_DS\_HashMap\_SyncMap {

public static void main(String[] args) {

HashMap<String, Integer> Hm1 = new HashMap<String, Integer>();

Map<String, Integer> syncMap = Collections.synchronizedMap(Hm1);

}

}