# CHALLENGE - ARRAYS.

* I Case Study :-

1. Consider a healthcare information system that needs to quickly retrieve and display Patient records. Discuss the choice of data structures and algorithms for efficient searching and sorting. Justify with suitable example.

→ Data Structures :-

1) Hash Tables:
- Storing and retrieving Patient records based on a unique identifier, such as patient I.D.
- Hash tables provide constant average-case complexity for search, insert, and delete operations.

2) Binary Search Trees :-
- Sorting and searching Patient records based on certain criteria, like date of admission or medical history.
- It maintains the order and support efficient search operations in O(logn) time complexity.

* Algorithms :-

1) Merge sort :
- Sorting patient records for displaying in a tabular form based on different attributes like patient name, age, or admission date.
- Quick sort / Merge sort offer O(n logn) time complexity for sorting.

2) Binary Search:
- Searching for a specific patient record based on attribute like patient ID or admission date.
- Binary search, when applied on sorted array, Provides O(log n) time complexity.

2) Explore the use of arrays in handling financial data in real-time trading system. How can sorting algorithms be employed to quickly identify trends or anomalies? Discuss the implications of time and space complexity in this scenario. Justify with suitable examples.

→ Arrays can efficiently store this data, with each element representing a timestamped record containing relevant information like stock prices, trading volumes or other metrics.

* Sorting Algorithms
- A simple moving average involves sorting a subset of recent data points and calculating the average.

Example:

```
int i = low-1;
~~int~~ for(int j = low; j<high; j++)
      { if(arr[j] < pivot)
        { i++;
        swap (arr, i, j); }}
       swap (arr, i+1, high);
       return i+1;
```

• Time Complexity : O(n log n)
Space Complexity : O(1).

II Critical Thinking :

1. How would you choose between using one-dimensional array and a two-dimensional array for Storing data? Consider factors like access time, memory efficiency, and ease of manipulation. Justify using example.

→ One-dimensional Array :

- Advantages :
  - Accessing time is faster since only one index to consider.
  - Typically more memory-efficient as it uses a contigous block of memory.

- Disadvantages :
  - Limmited in organizing data in a structured way, especially for complex relationships.
  - It might be less intutive for representing data with multiple dimensions.

- Example :

```
int [] StudentScores = {30, 88, 92, 88, 94};
```

Two dimensional Arrays :-

- Provides a natural way to represent structured data, like a table or matrix.

- Easier to work with for applications that involve matrix operations or tabular data.
- Suitable to represent data with multiple dimensions.

Disadvantages:
- Accessing time might be slower since it involves two indices.

Example:-

```
double[] StockPrices = { {145.5, 150.2, 142.8},
                         {155.3, 148.7, 160.0},
                         {135.6, 142.0, 138.9}};
for { double{} stock: stockPrices)
    { Arrays.sort(stock); }.
```

2. Compare and contrast linear Search and Binary Search in the content of real time applications. When would you prefer one over ~~the~~ over the other, and why? Justify using suitable example.

→ Linear Search:

```
Public class LinearSearch{
    Public static void main(String[] args)
    { double[] StockPrices ={145.5, 150.2, 142.8,
      153.8, 148.7, 160.0, 135.6};
    double tagetPrice =148.7;
    int LinearSearchResult = LinearSearch(Stock Prices
        , targetPrice);
    if(linearSearchResult != -1)
    {   SOPln("Linear Search :TargetPrice" + targetP
    }else {SOPln ("Linear Search: Target Price" +
          + targetPrice + " not found in the array"
    }
  }
  Private static int linearSearch (double[] arr,
  double target) {
    for( int i=0; i<arr.length; i++ {
      if (arr[i]== target)
      { return i; }
      }
      return -1; }
  }
```

*. Advantages & Disadvantages:
- Simple, suitable for unsorted data.
- Highly efficient for unsorted data.
- Useful when the dataset is small.

- Time complexity O(n) in worst case.

* Binary Search.

```
double[] sortedStockPrices = {135.6, 142.8, 145.5,
148.7, 150.2, 155.3, 160.0};
double targetPrice = 148.7;
int binarySearchResult = Arrays.binarySearch
(sortedStockPrices, targetPrice);
```

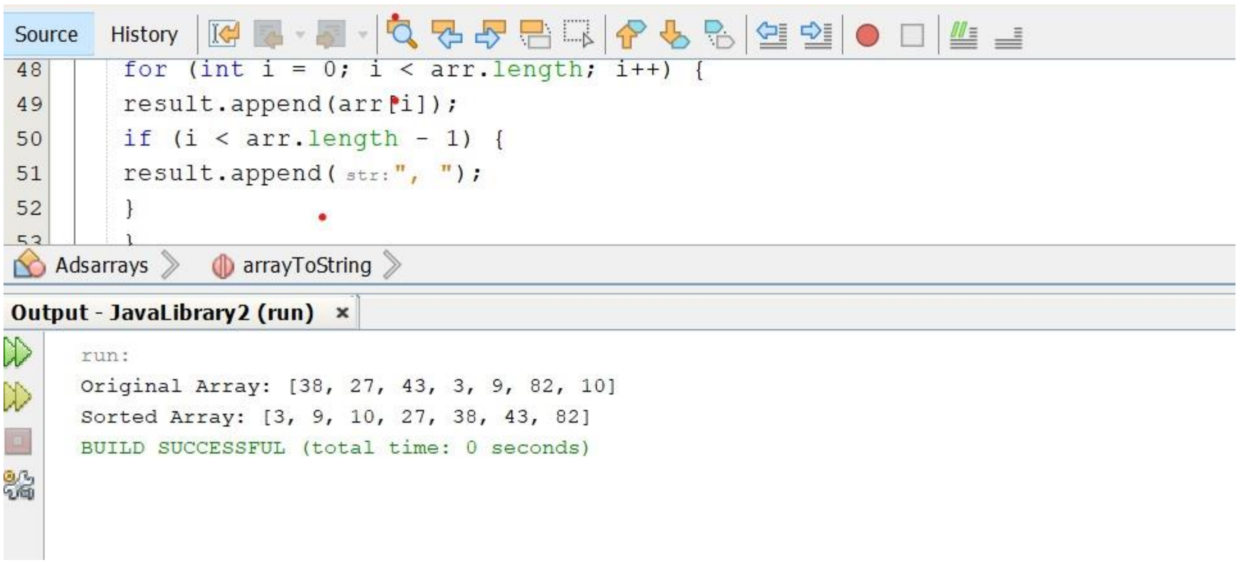
```
if (binarySearchResult >=0){
   SOPln ("Binary Search: Target Price"+
   targetPrice+ "found at Index"+ binarySearch
   Result); }
   else {
        S.O.Pln("Binary Search: Taget Price" +
        targetPrice + "not found in the
        array");
        }
     }
  }
```

* Advantages & Disadvantages :-

- Highly efficient for sorted data, with a time complexity of O(log n) in the worst case.
- Ideal for scenarios with large datasets, especially in real-time applications where quick response times are crucial.

**III Píogíamming Challenge (C/C++/Java/Python) : (30 maíks each )**
1) Implement the any soít algoíithm (inseítion, bubble, meíge, quick, selection)

```java
public class MergeSort {

    public static void main(String[] args) {
        int[] array = {38, 27, 43, 3, 9, 82, 10};

        System.out.println("Original Array: " + arrayToString(array));

        mergeSort(array, 0, array.length - 1);

        System.out.println("Sorted Array: " + arrayToString(array));
    }

    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;

            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);

            merge(arr, left, mid, right);
        }
    }

    public static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];

        System.arraycopy(arr, left, leftArray, 0, n1);
        System.arraycopy(arr, mid + 1, rightArray, 0, n2);


        int i = 0, j = 0;

        int k = left;
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                arr[k] = leftArray[i];
                i++;
            } else {
                arr[k] = rightArray[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = leftArray[i];
            i++;
            k++;
        }

        while (j < n2) {
            arr[k] = rightArray[j];
            j++;
            k++;
        }
    }

    public static String arrayToString(int[] arr) {
        StringBuilder result = new StringBuilder("[");
        for (int i = 0; i < arr.length; i++) {
            result.append(arr[i]);
            if (i < arr.length - 1) {
                result.append(", ");
            }
        }
        result.append("]");
        return result.toString();
    }

}
```



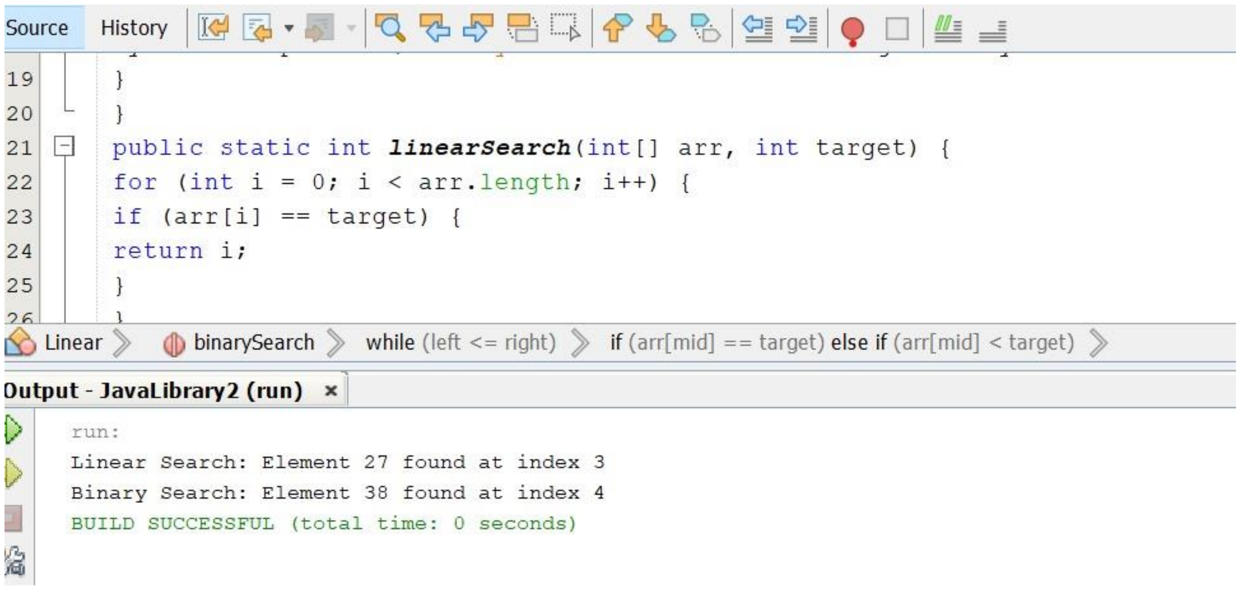
```
for (int i = 0; i < arr.length; i++) {
result.append(arr[i]);
if (i < arr.length - 1) {
result.append( str: ", ");
}
```

```
run:
Original Array: [38, 27, 43, 3, 9, 82, 10]
Sorted Array: [3, 9, 10, 27, 38, 43, 82]
BUILD SUCCESSFUL (total time: 0 seconds)
```

2) Wíite a function foí LineaíSeaích and BinaíySeaích which takes the above soíted aííay v as input.

```
public class SearchAlgorithms {
```

```
public static void main(String[] args) {
        int[] sortedArray = {3, 9, 10, 27, 38, 43, 82};

        int targetLinear = 27;
        int targetBinary = 38;

        int linearSearchResult = linearSearch(sortedArray, targetLinear);
        if (linearSearchResult != -1) {
            System.out.println("Linear Search: Element " + targetLinear + " found at index " +
 linearSearchResult);
        } else {
            System.out.println("Linear Search: Element " + targetLinear + " not found in the array");
        }

        int binarySearchResult = binarySearch(sortedArray, targetBinary);
        if (binarySearchResult != -1) {
            System.out.println("Binary Search: Element " + targetBinary + " found at index " +
 binarySearchResult);
        } else {
            System.out.println("Binary Search: Element " + targetBinary + " not found in the array");
        }
    }

    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }

    public static int binarySearch(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                return mid; // Return the index if the target is found
            } else if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1;
    }
}
```

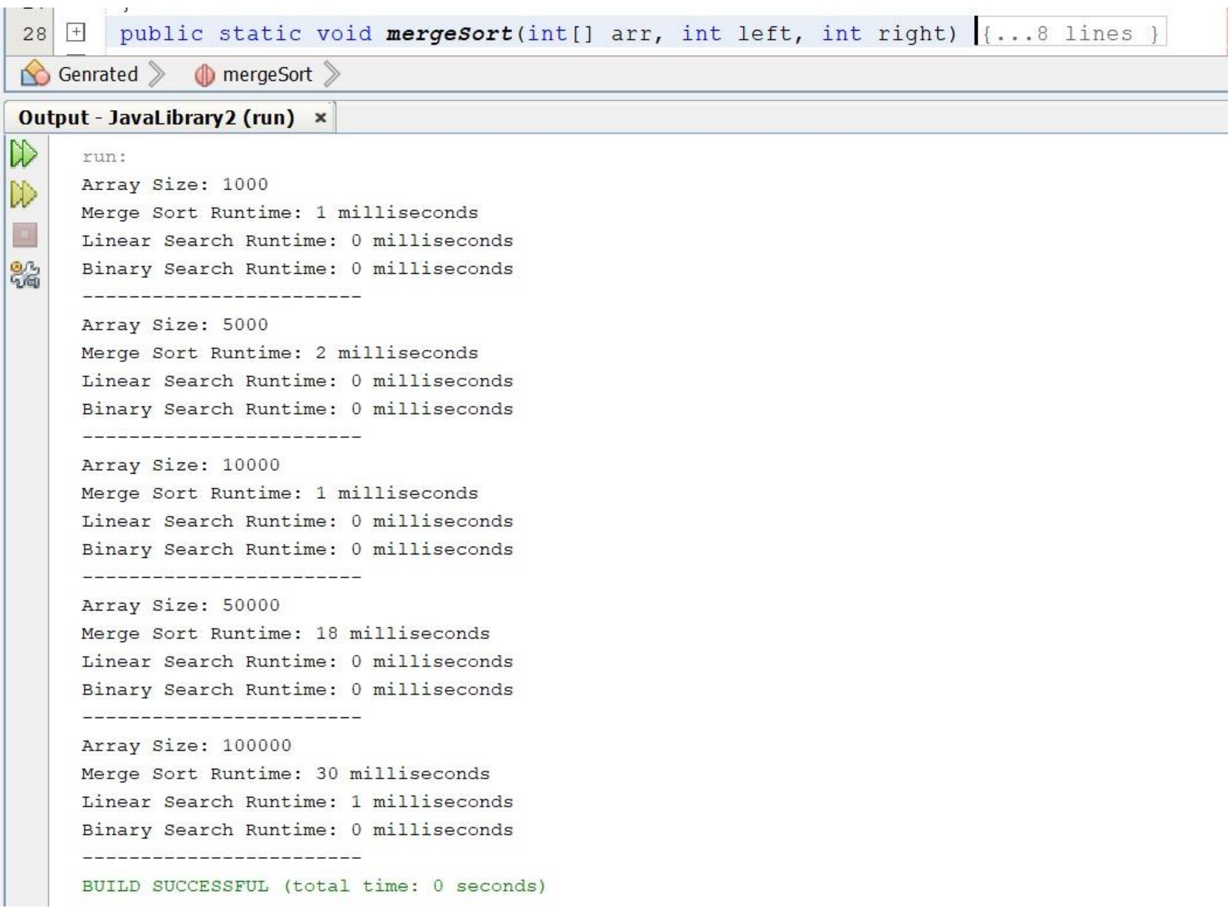

Source
History
Linear
binarySearch
while (left <= right)
if (arr[mid] == target) else if (arr[mid] < target)
Output - JavaLibrary2 (run)

```
  }
  }
  public static int linearSearch(int[] arr, int target) {
  for (int i = 0; i < arr.length; i++) {
  if (arr[i] == target) {
  return i;
  }
  }
```

```
run:
Linear Search: Element 27 found at index 3
Binary Search: Element 38 found at index 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

3) Now we aíe inteíested in how the íun-time of the algoíithms change as the size of the aííay changes. Ґo
analyze this, you will wíite a single scíipt that geneíates íandom aííays, íuns the soíting and seaích methods
on the geneíated aííays, and evaluates the íun-times.

```
import java.util.Arrays;
import java.util.Random;

public class RuntimeAnalysis {

    public static void main(String[] args) {
        int[] arraySizes = {1000, 5000, 10000, 50000, 100000};

        for (int size : arraySizes) {
            int[] randomArray = generateRandomArray(size);

            long mergeSortStartTime = System.currentTimeMillis();
            mergeSort(randomArray, 0, randomArray.length - 1);
            long mergeSortEndTime = System.currentTimeMillis();
            long mergeSortRuntime = mergeSortEndTime - mergeSortStartTime;

            int targetLinear = randomArray[randomArray.length / 2];
            long linearSearchStartTime = System.currentTimeMillis();
            linearSearch(randomArray, targetLinear);
            long linearSearchEndTime = System.currentTimeMillis();
            long linearSearchRuntime = linearSearchEndTime - linearSearchStartTime;

            long binarySearchStartTime = System.currentTimeMillis();
            binarySearch(randomArray, targetLinear);
            long binarySearchEndTime = System.currentTimeMillis();
            long binarySearchRuntime = binarySearchEndTime - binarySearchStartTime;

            System.out.println("Array Size: " + size);
            System.out.println("Merge Sort Runtime: " + mergeSortRuntime + " milliseconds");
            System.out.println("Linear Search Runtime: " + linearSearchRuntime + " milliseconds");
            System.out.println("Binary Search Runtime: " + binarySearchRuntime + " milliseconds");
            System.out.println(" --------------------- ");
        }
    }

    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            int mid = (left + right) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    public static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];

        System.arraycopy(arr, left, leftArray, 0, n1);
        System.arraycopy(arr, mid + 1, rightArray, 0, n2);

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (leftArray[i] <= rightArray[j]) {
                arr[k] = leftArray[i];
                i++;
            } else {
                arr[k] = rightArray[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            arr[k] = leftArray[i];
            i++;
            k++;
        }
        while (j < n2) {
            arr[k] = rightArray[j];
            j++;
            k++;
        }
    }

    public static int linearSearch(int[] arr, int target) {
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] == target) {
                return i;
            }
        }
        return -1;
    }

    public static int binarySearch(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                return mid;
            } else if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return -1;
    }

    public static int[] generateRandomArray(int size) {
        Random random = new Random();
        int[] array = new int[size];
        for (int i = 0; i < size; i++) {
            array[i] = random.nextInt(1000); // Adjust the range as needed
        }
        return array;
    }
}
```



```
public static void mergeSort(int[] arr, int left, int right) {...8 lines }
```

```
run:
Array Size: 1000
Merge Sort Runtime: 1 milliseconds
Linear Search Runtime: 0 milliseconds
Binary Search Runtime: 0 milliseconds
------------------------
Array Size: 5000
Merge Sort Runtime: 2 milliseconds
Linear Search Runtime: 0 milliseconds
Binary Search Runtime: 0 milliseconds
------------------------
Array Size: 10000
Merge Sort Runtime: 1 milliseconds
Linear Search Runtime: 0 milliseconds
Binary Search Runtime: 0 milliseconds
------------------------
Array Size: 50000
Merge Sort Runtime: 18 milliseconds
Linear Search Runtime: 0 milliseconds
Binary Search Runtime: 0 milliseconds
------------------------
Array Size: 100000
Merge Sort Runtime: 30 milliseconds
Linear Search Runtime: 1 milliseconds
Binary Search Runtime: 0 milliseconds
------------------------
BUILD SUCCESSFUL (total time: 0 seconds)
```