Array

What is an Array ?

Java array is an object which contains elements of a similar data type. The elements of an array are stored in a contiguous memory location.

It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array can contain primitives (int, char, etc.) as well as object(or non-primitive) references of a class depending on the definition of the array. In case of primitive data types, the actual values are stored in contiguous memory locations.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimension Array :

A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one-dimensional array. The array subscript or index begins from 0.

Syntax: data_type array_name[Number_of Elements] ;

2-Dimension Array :

This array can be defined as an array of several one-dimensional arrays. It can be visualized in the form of a table with rows and columns. It is an array with two indexes. One index represents the row number and the other the column number.

Syntax: data_type array_name [row size][column size] ;

Multi Dimension Array :

A 2-D array can be defined as an array of several 1-D arrays. Likewise, a 3-D array can be defined as an array of several 2-D arrays. In general, an n-dimensional

array can be defined as an array of several (n-1) dimensional arrays.

Example: arr[2][2][2] is a 3-D array with 2 two-D arrays. Each 2-D array has 2 rows and 2 columns.

```
import java.util.Scanner;
public class Main
{
   public static void main(String[] args)
   {
      int n;
      Scanner s = new Scanner(System.in);
      System.out.print("Enter no. of elements you want in array: ");
      n = s.nextInt();
      int a[] = new int[n];
      System.out.println("Enter all the elements: ");
      for (int i = 0; i < n; i++)
      {
         a[i] = s.nextInt();
      }
      System.out.print("Odd numbers: ");
      for(int i = 0 ; i < n ; i++)
      {
         if(a[i] % 2 != 0)
         {
            System.out.print(a[i]+" ");
         }
      }
      System.out.println("");
      System.out.print("Even numbers: ");
      for(int i = 0 ; i < n ; i++)
      {
         if(a[i] % 2 == 0)
         {
            System.out.print(a[i]+" ");
         }
```

```
      }
    }
}
```

Sort the array in Java

```
public class Main {
    public static void main(String[] args) {

        //Initialize array
        int [] arr = new int [] {10, 40, 30, 20};
        int temp = 0;

        //Sort the array in ascending order
        for (int i = 0; i < arr.length; i++) {
            for (int j = i+1; j < arr.length; j++) {
                 if(arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
              }
            }
        }


    //Displaying elements of array after sorting
    for (int i = 0; i < arr.length; i++) {
     System.out.print(arr[i] + " ");
   }
  }
}
```

Rotation of an array at the given Index

array is entered Arr={1,2,3,4,5,6,7,8,9} and it is asked to rotate this array to its left by 2 positions then the Output will be Arr={3,4,5,6,7,8,9,1,2} or to its right by 2 positions then the output will be Arr={8,9,1,2,3,4,5,6,7}.

Algorithm for Array Rotation

- Rotate(int [] arr, int n)
- FOR i from 1 to pivot index:
- temp= last element/initial element
- FOR i=0 to i=arr.length
- arr[j]=arr[j-1]/arr[j+1]
- END of FOR
- arr[0] / arr[arr.length} = temporary
- END of FOR

```
import java.util.*;

public class Rotation {

  void rightRotate(int[] arr, int n) {
    int temp;
    for (int i = 1; i <= n; i++) {
      temp = arr[arr.length - 1];
      for (int j = arr.length - 1; j > 0; j--) {
        arr[j] = arr[j - 1];
      }
      arr[0] = temp;
    }
    System.out.println(
      "Input Array After Right Rotation By " + n + " Positions :"
    );
    System.out.println(Arrays.toString(arr));
  }
```

```
void leftRotate(int[] arr, int n) {
  int temp;
  for (int i = 0; i < n; i++) {
    temp = arr[0];
    for (int j = 0; j < arr.length - 1; j++) {
      arr[j] = arr[j + 1];
    }
    arr[arr.length - 1] = temp;
  }
  System.out.println(
    "Input Array After Left Rotation By " + n + " Positions :"
  );
  System.out.println(Arrays.toString(arr));
}

public static void main(String[] args) {
  Scanner s = new Scanner(System.in);

  System.out.println("Enter the length of the array you wish to rotate: ");
  int l = s.nextInt();
  int arr1[] = new int[l];
  int arr2[] = new int[l];

  System.out.println("Enter the Elements");
  for (int i = 0; i < l; i++) {
    arr1[i] = s.nextInt();
    arr2[i] = arr1[i];
  }

  System.out.println("Enter the index of rotation");
  int r = s.nextInt();
  System.out.println("Input Array Before Rotation :");
  System.out.println(Arrays.toString(arr1));

  Rotation m = new Rotation();
```

```
    m.rightRotate(arr1, r);
    m.leftRotate(arr2, r);
  }
}
```

How To Reverse Integer Or String Array

Input : {2, 4, 6, 8, 10}

Output : {10, 8, 6, 4, 2}

Algorithm to reverse an array using third variable :
1. START
2. Initialize an array with values
3. Take two variables left and right with left containing 0 and right containing rightmost index of array.
4. Take a new variable as temp.
5. Swap the values of left indices with the right indices using temp.
6. Increment the left variable and decrement the right variable value.
7. Repeat the steps from 4 to 5 till left is less than right.
8. Print reverse array.
9. END

```
import java.util.*;

public class Reverse_Array {

  public static void main(String[] args) {
    // Given input array
    int[] inputArray = { 2, 4, 6, 8, 10 };
    // Print array before reverse
    System.out.println(
      "Array without reverse : " + Arrays.toString(inputArray)
    );
    // Calling method to swap elements
    reverse(inputArray);
```

```
  }

  public static void reverse(int[] inputArray) {
    for (
      int left = 0, right = inputArray.length - 1;
      left < right;
      left++, right--
    ) {
      // swap the values at the left and right indices
      int temp = inputArray[left];
      inputArray[left] = inputArray[right];
      inputArray[right] = temp;
    }
    // Printing the Array after reverse
    System.out.print("Reverse Array :");
    for (int i : inputArray) {
      System.out.println(" " + i);
    }
  }
}
```

Algorithm to Reverse a String in JAVA

1. START
2. Initialize the String Array with values
3. Take two variables left and right with left containing 0 and right containing rightmost index of array.
4. Take an new variable as temp.
5. Swap the values of left indices with the right indices using temp.
6. Increment the left variable and decrement the right variable value.
7. Repeat the steps from 4 to 5 till left is less than right.
8. Print reverse array.
9. END

```
import java.util.*;

public class Reverse_String {

  public static void main(String[] args) {
    // Given input array
    String[] inputArray = { "India", "UK", "Pakistan", "Australia" };
    // Print array before reverse
    System.out.println("Initial String array : " + Arrays.toString(inputArray));
    // Calling method to swap elements
    reverse(inputArray);
  }

  public static void reverse(String[] inputArray) {
    for (
      int left = 0, right = inputArray.length - 1;
      left < right;
      left++, right--
    ) {
      // swap the values at the left and right indices
      String temp = inputArray[left];
      inputArray[left] = inputArray[right];
      inputArray[right] = temp;
    }
    // Printing the Array after reverse
    System.out.print("Reverse Array : " + Arrays.toString(inputArray));
  }
}
```

Pairs in Array with Given Sum

To find pairs in an array with a given sum in Java, we can use two methods

1. **Brute Force** (Time complexity: O(n^2))
2. **Using Sorting** (Time complexity: O(n log n))

Both methods have their own pros and cons. The **brute force method** can be slow for large arrays but it is very simple and easy to implement . The **sorting method** is faster for large arrays, but requires extra space for sorting and modifying the original array.

Method 1 : Brute Force Method

- In this approach, we check every pair of elements in the array to see if their sum is equal to the given target sum.
- This method has a time complexity of O(n^2) as we have to compare each element with every other element

```
public class Pair_Sum {

  static void p_sum(int a[], int size, int sum) {
    System.out.print("The pairs present are : ");
    //traverse through the array to find pairs
    for (int i = 0; i < size; i++) {
      for (int j = i + 1; j < size; j++) {
        if (a[i] + a[j] == sum) {
          System.out.println("(" + a[i] + ", " + a[j] + ") ");
        }
      }
    }
  }

  // Driver Code
  public static void main(String[] arg) {
    int a[] = { 5, 2, 3, 4, 1, 6, 7 };
    int size = a.length;
    int sum = 7;
    p_sum(a, size, sum);
  }
}
```

Method 2 : Sorting Method

- The array is first sorted in ascending order in the sorting technique.
- Then, we employ two pointers, one of which points to the first element and the other to the last element.
- These two pointers values are added, and we then check to see if the result equals the specified sum.The left pointer is increased if the value is less than the total, and the right pointer is decreased if the value is higher.
- This method has a time complexity of O(n log n) due to the sorting algorithm used.

```
import java.util.*;

public class Array_Sort_Sum {

  public static void findPairs(int arr[], int n, int targetSum) {
    Arrays.sort(arr);

    int left = 0, right = n - 1;

    System.out.println("The pairs present are : ");

    while (left < right) {
      int currSum = arr[left] + arr[right];

      if (currSum == targetSum) {
        System.out.println("(" + arr[left] + ", " + arr[right] + ") ");
        left++;
        right--;
      } else if (currSum < targetSum) {
        left++;
      } else {
        right--;
      }
    }
```

```
  }

  public static void main(String args[]) {
    int arr[] = { 5, 8, 1, 4, 6, 3, 2, 7 };
    int n = arr.length;
    int targetSum = 10;
    findPairs(arr, n, targetSum);
  }
}
```

Sort Array in Waveform

Sorting an array in waveform refers to the process of rearranging the elements of the array in a specific manner, such that the resulting array represents a waveform-like pattern. In waveform sorting, the elements of the array are arranged in a way that alternates between ascending and descending order, mimicking the shape of a waveform with peaks and troughs.

[10, 5, 6, 2, 20, 3, 48, 30]

Method 1 : Brute Force Solution (using sorting)

The brute force solution using sorting involves the following steps:

Step 1: Sort
the array in ascending order. For example, if the input array is {20, 30, 5, 25, 10, 15}, after sorting, we get {5, 10, 15, 20, 25, 30}.

Step 2:
Swap all adjacent elements of the array. Using the sorted array from step 1, we swap adjacent elements to obtain the final result. For example, swapping adjacent elements in the sorted array {5, 10, 15, 20, 25, 30} would give us {10, 5, 20, 15, 30, 25}.

This method is
simple but may not be the most efficient solution for larger arrays, as sorting has a time complexity of O(n log n) and swapping adjacent

elements has a time complexity of O(n), resulting in an overall time complexity of O(n log n) + O(n), which can be further optimized.

```
public class makeWaveform1 {

  public static void makeWF1(int n, int arr[]) {
    int temp;
    for (int i = 0; i < n - 1; i++) {
      for (int j = i + 1; j < n; j++) {
        if (arr[i] > arr[j]) {
          temp = arr[i];
          arr[i] = arr[j];
          arr[j] = temp;
        }
      }
    }

    for (int i = 0; i < n; i += 2) {
      if (i < n - 1) {
        temp = arr[i];
        arr[i] = arr[i + 1];
        arr[i + 1] = temp;
      }
    }

    System.out.println("Array in Waveform Pattern : ");
    for (int ele : arr) {
      System.out.println(ele + " ");
    }
  }

  public static void main(String args[]) {
    int arr[] = { 5, 8, 1, 4, 6, 3, 2, 7 };
    int n = arr.length;
    makeWF1(n, arr);
```

```
  }
}
```

Method 2 : Incrementing the loop by 2

The
above method using sorting has a time complexity of O(n log n), which may not be the most optimal solution for this problem. The Incrementing the loop by 2 method can indeed solve the problem in O(n) time complexity by making use of a single traversal of the array. Here are the steps for solving the problem in O(n) complexity:

Step 1: Traverse all even position elements of the array (i.e. elements at index 0, 2, 4, 6, etc.).

Step 2: For each even position element, check the following two conditions:

Condition
1: If the current element is smaller than the previous odd element (i.e. element at index (i-1)), swap the previous and current element.

Condition
2: If the current element is smaller than the next odd element (i.e. element at index (i+1)), swap the next and current element.

By
following these steps, you can rearrange the array in O(n) time complexity, as you are traversing the array once and performing constant time swaps. This method is more efficient compared to the sorting method in terms of time complexity for this specific problem.

```
public class makeWaveform2 {

  public static void makeWF2(int n, int arr[]) {
    int temp;
    for (int i = 0; i < n; i += 2) {
      if (i > 0 && arr[i - 1] > arr[i]) {
        temp = arr[i];
        arr[i] = arr[i - 1];
```

```
        arr[i - 1] = temp;
      }
      if (i < n - 1 && arr[i] < arr[i + 1]) {
        temp = arr[i];
        arr[i] = arr[i + 1];
        arr[i + 1] = temp;
      }
    }

    System.out.println("Array in Waveform Pattern : ");
    for (int ele : arr) {
      System.out.println(ele + " ");
    }
  }

  public static void main(String args[]) {
    int arr[] = { 5, 8, 1, 4, 6, 3, 2, 7 };
    int n = arr.length;
    makeWF2(n, arr);
  }
}
```