**1- Reverse the array**

Input : arr[] = {1, 2, 3}

Output : arr[] = {3, 2, 1}

Input : arr[] = {4, 5, 1, 2}

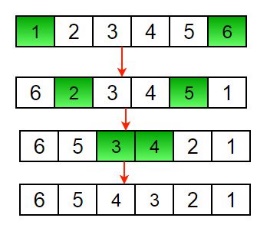
Output : arr[] = {2, 1, 5, 4}

**Algo:**

1) Initialize start and end indexes as start = 0, end = n-1

2) In a loop, swap arr[start] with arr[end] and change start and end as follows :

start = start +1, end = end – 1



public class ReverseArrayIterative {  
 static void rvereseArray(int arr[], int start, int end) {  
 int temp;  
 while (start < end) {  
 temp = arr[start];  
 arr[start] = arr[end];  
 arr[end] = temp;  
 start++;  
 end--;  
 }  
 }  
  
 static void printArray(int arr[], int size) {  
 for (int i = 0; i < size; i++)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 }  
  
 public static void main(String args[]) {  
 int arr[] = {1, 2, 3, 4, 5, 6};  
 *printArray*(arr, 6);  
 *rvereseArray*(arr, 0, 5);  
 System.*out*.print("Reversed array is \n");  
 *printArray*(arr, 6);  
  
 }  
}

public class ReverseArrayRecurssive {  
 static void rvereseArray(int arr[], int start, int end) {  
 int temp;  
 if (start >= end)  
 return;  
 temp = arr[start];  
 arr[start] = arr[end];  
 arr[end] = temp;  
 *rvereseArray*(arr, start + 1, end - 1);  
 }  
  
 static void printArray(int arr[], int size) {  
 for (int i = 0; i < size; i++)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println();  
 }  
  
 public static void main(String args[]) {  
 int arr[] = {1, 2, 3, 4, 5, 6};  
 *printArray*(arr, 6);  
 *rvereseArray*(arr, 0, 5);  
 System.*out*.print("Reversed array is \n");  
 *printArray*(arr, 6);  
 }  
}

**1 2 3 4 5 6**

**Reversed array is**

**6 5 4 3 2 1**

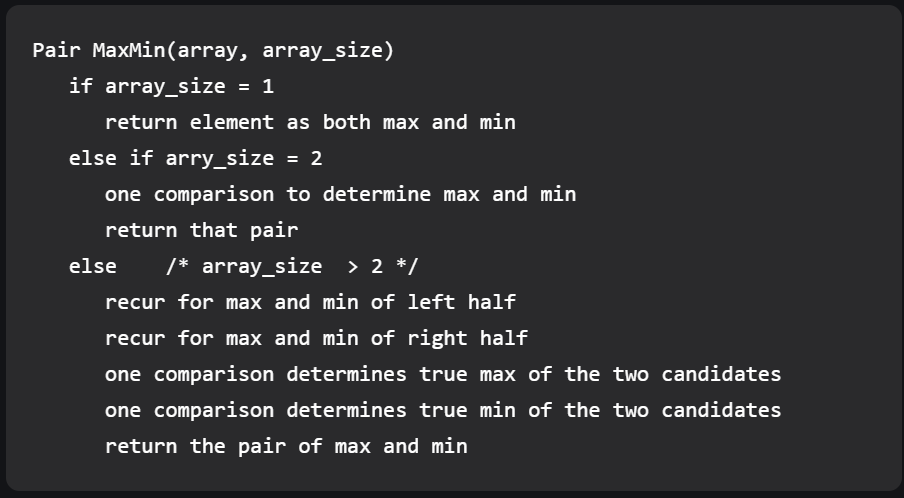
**2-Find the maximum and minimum element in an array**

**METHOD 1 (Simple Linear Search)**

* Initialize values of min and max as minimum and maximum of the first two elements respectively.
* Starting from 3rd, compare each element with max and min, and change max and min accordingly
* (i.e., if the element is smaller than min then change min, else if the element is greater than max then change max, else ignore the element)

**METHOD 2 (Tournament Method) – Merge sort – Divide part 😊**

* Divide the array into two parts and compare the maximums and minimums of the two parts to get the maximum and the minimum of the whole array.



public class MinMaxOfArray {  
 static class Pair {  
 int min;  
 int max;  
 }  
  
 static Pair getMinMax(int arr[], int low, int high) {  
 Pair minMax = new Pair();  
 Pair mmLeft = new Pair();  
 Pair mmRight = new Pair();  
 int mid;  
  
 // If there is only one element  
 if (low == high) {  
 minMax.max = arr[low];  
 minMax.min = arr[low];  
 return minMax;  
 }  
  
 /\* If there are two elements \*/  
 if (high == low + 1) {  
 if (arr[low] > arr[high]) {  
 minMax.max = arr[low];  
 minMax.min = arr[high];  
 } else {  
 minMax.max = arr[high];  
 minMax.min = arr[low];  
 }  
 return minMax;  
 }  
  
 /\* If there are more than 2 elements \*/  
 mid = (low + high) / 2;  
 mmLeft = *getMinMax*(arr, low, mid);  
 mmRight = *getMinMax*(arr, mid + 1, high);  
  
 /\* compare minimums of two parts\*/  
 if (mmLeft.min < mmRight.min) {  
 minMax.min = mmLeft.min;  
 } else {  
 minMax.min = mmRight.min;  
 }  
  
 /\* compare maximums of two parts\*/  
 if (mmLeft.max > mmRight.max) {  
 minMax.max = mmLeft.max;  
 } else {  
 minMax.max = mmRight.max;  
 }  
 return minMax;  
 }  
  
 public static void main(String args[]) {  
 int arr[] = {1000, 11, 445, 1, 330, 3000};  
 int arr\_size = 6;  
 Pair minmax = *getMinMax*(arr, 0, arr\_size - 1);  
 System.*out*.printf("\nMinimum element is %d", minmax.min);  
 System.*out*.printf("\nMaximum element is %d", minmax.max);  
 }  
}

Minimum element is 1

Maximum element is 3000

**3- Find the "Kth" max and min element of an array**

**Method 1 (Simple Solution)**

* A simple solution is to sort the given array using a O(N log N) sorting algorithm like Merge Sort, Heap Sort, etc, and return the element at index k-1 in the sorted array.
* Time Complexity of this solution is **O(N Log N)**

**Method 3 (Using Min Heap – HeapSelect)**

* We can find k’th smallest element in time complexity better than O(N Log N).
* A simple optimization is to create a Min Heap of the given n elements and call extractMin() k times.
* Time Complexity of this solution is **O(N Log K)**

public class KthSmallestNumber {  
 // Function to find the k'th smallest element in an array using min-heap  
 public static int findKthSmallest(List<Integer> input, int k) {  
 // base case  
 if (input == null || input.size() < k) {  
 System.*exit*(-1);  
 }  
  
 // create an empty min-heap and initialize it with all input elements  
 PriorityQueue<Integer> pq = new PriorityQueue<>(input);  
  
 // pop from min-heap exactly `k-1` times  
 while (--k > 0) {  
 pq.poll();  
 }  
  
 // return the root of min-heap  
 return pq.peek();  
 }  
  
 public static void main(String[] args) {  
 List<Integer> input = Arrays.*asList*(7, 4, 6, 3, 9, 1);  
 int k = 3;  
 System.*out*.println("k'th smallest array element is " + *findKthSmallest*(input, k));  
 }  
}

**4- Given an array which consists of only 0, 1 and 2. Sort the array without using any sorting algo**

**Algo (Dutch national flag problem)**

* Mid iterates through all the array
* If mid points to 0 => swap it with low
* If mid points to 1 => Leave it as it is and move forward. it’s a fixed pole 😊
* If mid points to 2 => swap it with high



public class SortZerosOnesTwos {  
 public static void main(String[] args) {  
 int arr[] = {0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1};  
 int arr\_size = arr.length;  
 *sort012*(arr, arr\_size);  
 System.*out*.println("Array after seggregation ");  
 *printArray*(arr, arr\_size);  
 }  
  
 static void sort012(int input[], int arr\_size) {  
 int low = 0, mid = 0;  
 int high = arr\_size - 1;  
 while (mid <= high) {//mid iterates whole array from low to high  
 switch (input[mid]) {  
 case 0: {  
 input = *swap*(input, low, mid);  
 low++;  
 mid++;  
 break;  
 }  
 case 1:  
 mid++;  
 break;  
 case 2: {  
 input = *swap*(input, mid, high);  
 high--;  
 break;  
 }  
 }  
 }  
 }  
  
 static void printArray(int arr[], int arr\_size) {  
 int i;  
 for (i = 0; i < arr\_size; i++)  
 System.*out*.print(arr[i] + " ");  
 System.*out*.println("");  
 }  
  
 static int[] swap(int arr[], int index1, int index2) {  
 int temp = arr[index1];  
 arr[index1] = arr[index2];  
 arr[index2] = temp;  
 return arr;  
 }  
}