* **QUESTIONS ON SEARCHING & SORTING ALGORITHMS**

**Q1. Implement binary search on a sorted array to find a target element.**

public class BinarySearch {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9};

int target = 7;

int result = binarySearch(arr, target);

if (result != -1) {

System.out.println("Element found at index " + result);

} else {

System.out.println("Element not found in the array");

}

}

private 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; // Element not found

}

}

**Output:**

Element found at index 6

**Explanation:**

Binary search has a time complexity of O(log n) as it halves the search space in each iteration.

**Q2. Sort an array of integers using the Bubble Sort algorithm.**

public class BubbleSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 5};

bubbleSort(arr);

System.out.println("Sorted array: " + Arrays.toString(arr));

}

private static void bubbleSort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap elements if they are in the wrong order

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

}

**Output:**

Sorted array: [1, 2, 5, 5, 9]

**Explanation:**

Bubble Sort has a time complexity of O(n^2) as it iterates through the array and compares/swaps adjacent elements.

**Q3. Sort an array of integers using the Selection Sort algorithm.**

public class SelectionSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 5};

selectionSort(arr);

System.out.println("Sorted array: " + Arrays.toString(arr));

}

private static void selectionSort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Swap the found minimum element with the first element

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

}

**Output:**

Sorted array: [1, 2, 5, 5, 9]

**Explanation:**

Selection Sort also has a time complexity of O(n^2) as it iterates through the array to find the minimum element in each iteration.

**Q4. Perform a linear search on an array to find a target element.**

public class LinearSearch {

public static void main(String[] args) {

int[] arr = {4, 2, 7, 1, 9};

int target = 7;

int result = linearSearch(arr, target);

if (result != -1) {

System.out.println("Element found at index " + result);

} else {

System.out.println("Element not found in the array");

}

}

private static int linearSearch(int[] arr, int target) {

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

if (arr[i] == target) {

return i;

}

}

return -1; // Element not found

}

}

**Output:**

Element found at index 2

**Explanation:**

Linear search has a time complexity of O(n) as it iterates through the array until it finds the target element or reaches the end.

**Q5. Sort an array of integers using the Insertion Sort algorithm.**

public class InsertionSort {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 5};

insertionSort(arr);

System.out.println("Sorted array: " + Arrays.toString(arr));

}

private static void insertionSort(int[] arr) {

int n = arr.length;

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

// Move elements of arr[0..i-1] that are greater than key

// to one position ahead of their current position

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

}

**Output:**

Sorted array: [1, 2, 5, 5, 9]

**Explanation:**

Insertion Sort has a time complexity of O(n^2) in the worst case, as it iterates through the array and inserts each element into its proper position.