Divide & Conquor question

Count Inversions in an Array: Use merge sort to count the number of inversions in a given array.
Closest Pair of Points: Given a set of points on a plane, find the closest pair using a divide and conquer strategy.
Matrix Multiplication using Strassen’s Algorithm: Implement Strassen’s algorithm for matrix multiplication.
K-th Largest Element: Find the K-th largest element in an unsorted array using a modified version of quicksort.
Merge k Sorted Arrays: Given k sorted arrays, merge them into one sorted array using the divide and conquer technique.
Skyline Problem: Given a set of buildings in the form of (left, right, height), use the divide and conquer approach to determine the skyline silhouette.
Finding the Median of Two Sorted Arrays: Given two sorted arrays, find the median using the divide and conquer approach with O(log(min(m,n))) time complexity.
Count the Number of Reverse Pairs in an Array: Use a divide and conquer approach to count reverse pairs (i, j) such that i < j and nums[i] > 2 * nums[j].
Convex Hull Algorithm (Divide and Conquer): Implement the divide and conquer approach to find the convex hull of a set of points.

explain me problem statement with example in simple way and give java simple and optimizied code with time complexity

Let's break down each problem and provide an explanation, along with Java code and its time complexity.

---

1. Count Inversions in an Array:

Problem Statement: Inversions are pairs of elements in an array where the earlier element is larger than the later one. You are required to count the number of such inversions using a modified merge sort.
Example: For array: `[2, 4, 1, 3, 5]` Inversions are: (2, 1), (4, 1), and (4, 3) → Total = 3 inversions.
Java Code:

```java
class InversionCount {
    static int mergeAndCount(int[] arr, int left, int mid, int right) {
        int[] leftArr = Arrays.copyOfRange(arr, left, mid + 1);
        int[] rightArr = Arrays.copyOfRange(arr, mid + 1, right + 1);

        int i = 0, j = 0, k = left, swaps = 0;

        while (i < leftArr.length && j < rightArr.length) {
            if (leftArr[i] <= rightArr[j]) {
                arr[k++] = leftArr[i++];
            } else {
                arr[k++] = rightArr[j++];
                swaps += (mid + 1) - (left + i);
            }
        }

        while (i < leftArr.length) arr[k++] = leftArr[i++];
        while (j < rightArr.length) arr[k++] = rightArr[j++];

        return swaps;
    }

    static int mergeSortAndCount(int[] arr, int left, int right) {
        int count = 0;
        if (left < right) {
            int mid = left + (right - left) / 2;

            count += mergeSortAndCount(arr, left, mid);
            count += mergeSortAndCount(arr, mid + 1, right);
            count += mergeAndCount(arr, left, mid, right);
        }
        return count;
    }

    public static void main(String[] args) {
        int[] arr = {2, 4, 1, 3, 5};
        System.out.println("Number of inversions: " + mergeSortAndCount(arr, 0, arr.length
- 1));
    }
}
```

Time Complexity: O(n log n)

---

2. Closest Pair of Points:

Problem Statement: Given a set of points on a 2D plane, you need to find the two closest points using a divide and conquer approach.
Example: For points: `[(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]` Closest points are: `(2, 3)` and `(3, 4)` with a distance of approximately 1.41.
Java Code:

```java
import java.util.Arrays;

class ClosestPair {
    static class Point {
        int x, y;
        Point(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }

    static double dist(Point p1, Point p2) {
        return Math.sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y));
    }

    static double closestPair(Point[] points, int n) {
        if (n <= 3) return bruteForce(points, n);

        int mid = n / 2;
        Point midPoint = points[mid];

        double dl = closestPair(Arrays.copyOfRange(points, 0, mid), mid);
        double dr = closestPair(Arrays.copyOfRange(points, mid, n), n - mid);
        double d = Math.min(dl, dr);

        Point[] strip = new Point[n];
        int j = 0;
        for (int i = 0; i < n; i++) {
            if (Math.abs(points[i].x - midPoint.x) < d) {
                strip[j++] = points[i];
            }
        }

        return Math.min(d, stripClosest(strip, j, d));
    }

    static double bruteForce(Point[] points, int n) {
        double min = Double.MAX_VALUE;
        for (int i = 0; i < n; ++i)
            for (int j = i + 1; j < n; ++j)
                if (dist(points[i], points[j]) < min)
                    min = dist(points[i], points[j]);
        return min;
    }

    static double stripClosest(Point[] strip, int size, double d) {
        double min = d;
        Arrays.sort(strip, 0, size, (p1, p2) -> p1.y - p2.y);
        for (int i = 0; i < size; ++i)
            for (int j = i + 1; j < size && (strip[j].y - strip[i].y) < min; ++j)
                if (dist(strip[i], strip[j]) < min)
                    min = dist(strip[i], strip[j]);
        return min;
    }

    public static void main(String[] args) {
        Point[] points = {new Point(2, 3), new Point(12, 30), new Point(40, 50), new
Point(5, 1), new Point(12, 10), new Point(3, 4)};
        Arrays.sort(points, (p1, p2) -> p1.x - p2.x);
        System.out.println("The smallest distance is " + closestPair(points,
points.length));
    }
}
```

Time Complexity: O(n log n)

---

3. Matrix Multiplication using Strassen's Algorithm:

Problem Statement: Given two matrices, multiply them using Strassen's algorithm, which reduces the time complexity compared to the standard method.

Example: Matrix A = $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$,

Matrix B = $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

Result: Matrix C = $\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

Java Code:

```java
class StrassenMatrixMultiplication {
    static int[][] add(int[][] A, int[][] B) {
        int n = A.length;
        int[][] C = new int[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                C[i][j] = A[i][j] + B[i][j];
        return C;
    }

    static int[][] subtract(int[][] A, int[][] B) {
        int n = A.length;
        int[][] C = new int[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                C[i][j] = A[i][j] - B[i][j];
        return C;
    }

    static int[][] strassen(int[][] A, int[][] B) {
        int n = A.length;
        if (n == 1) {
            int[][] C = new int[1][1];
            C[0][0] = A[0][0] * B[0][0];
            return C;
        }

        int[][] A11 = new int[n/2][n/2], A12 = new int[n/2][n/2], A21 = new int[n/2][n/2],
A22 = new int[n/2][n/2];
        int[][] B11 = new int[n/2][n/2], B12 = new int[n/2][n/2], B21 = new int[n/2][n/2],
B22 = new int[n/2][n/2];

        split(A, A11, 0, 0);
        split(A, A12, 0, n/2);
        split(A, A21, n/2, 0);
        split(A, A22, n/2, n/2);

        split(B, B11, 0, 0);
        split(B, B12, 0, n/2);
        split(B, B21, n/2, 0);
        split(B, B22, n/2, n/2);

        int[][] M1 = strassen(add(A11, A22), add(B11, B22));
        int[][] M2 = strassen(add(A21, A22), B11);
        int[][] M3 = strassen(A11, subtract(B12, B22));
        int[][] M4 = strassen(A22, subtract(B21, B11));
        int[][] M5 = strassen(add(A11, A12), B22);
        int[][] M6 = strassen(subtract(A21, A11), add(B11, B12));
        int[][] M7 = strassen(subtract(A12, A22), add(B21, B22));

        int[][] C11 = add(subtract(add(M1, M4), M5), M7);
        int[][] C12 = add(M3, M5);
        int[][] C21 = add(M2, M4);
        int[][] C22 = add(subtract(add(M1, M3), M2), M6);

        int[][] C = new int[n][n];
        join(C11, C, 0, 0);
        join(C12, C, 0, n/2);
        join(C21, C, n/2, 0);
        join(C22, C, n/2, n/2);

        return C;
    }

    static void split(int[][] P, int[][] C, int iB, int jB) {
        for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)
            for (int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)
                C[i1][j1] = P[i2][j2];
    }

    static void join(int[][] C, int[][] P, int iB, int jB) {
        for (int i1 = 0, i2 = iB; i1 < C.length; i1++, i2++)
            for (int j1 = 0, j2 = jB; j1 < C.length; j1++, j2++)
                P[i2][j2] = C[i1][j1];
    }

    public static void main(String[] args) {
        int[][] A = { {1, 2}, {3, 4} };
        int[][] B = { {5, 6}, {7, 8} };
        int[][] C = strassen(A, B);
        System.out.println("Product of matrices: ");
        for (int i = 0; i < C.length; i++) {
            for (int j = 0; j < C.length; j++)
                System.out.print(C[i][j] + " ");
            System.out.println();
        }
    }
}
```

Time Complexity: O(n^log7), which is approximately O(n^2.81)

I'll continue with the remaining problems in subsequent responses.

give remaining

Let's continue with the explanations, examples, and Java code for the remaining problems.

---

4. K-th Largest Element:

Problem Statement: Find the K-th largest element in an unsorted array using a modified version of QuickSort (also known as QuickSelect).
Example: For array: `[3, 2, 1, 5, 6, 4]`
K = 2
K-th largest element is `5`.
Java Code:

```java
    class KthLargest {
        public static int partition(int[] arr, int low, int high) {
            int pivot = arr[high];
            int i = low - 1;
            for (int j = low; j < high; j++) {
                if (arr[j] > pivot) { // for K-th largest, we use '>'
                    i++;
                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
            int temp = arr[i + 1];
            arr[i + 1] = arr[high];
            arr[high] = temp;
            return i + 1;
        }

        public static int quickSelect(int[] arr, int low, int high, int k) {
            if (low <= high) {
                int pi = partition(arr, low, high);
                if (pi == k - 1)
                    return arr[pi];
                else if (pi > k - 1)
                    return quickSelect(arr, low, pi - 1, k);
                else
                    return quickSelect(arr, pi + 1, high, k);
            }
            return Integer.MAX_VALUE;
        }
        public static void main(String[] args) {
            int[] arr = {3, 2, 1, 5, 6, 4};
            int k = 2;
            System.out.println("K-th largest element is: " + quickSelect(arr, 0, arr.length -
    1, k));
        }
    }
```

Time Complexity: O(n) on average, but worst case is O(n^2).

---

5. Merge k Sorted Arrays:

Problem Statement: You are given `k` sorted arrays, merge them into one sorted array using a divide and conquer technique.
Example: For arrays:
`arr1 = [1, 4, 5]`
`arr2 = [1, 3, 4]`
`arr3 = [2, 6]`
The merged result is: `[1, 1, 2, 3, 4, 4, 5, 6]`.
Java Code:

```java
import java.util.PriorityQueue;

class MergeKSortedArrays {
    static class ArrayContainer implements Comparable<ArrayContainer> {
        int[] arr;
        int index;

        public ArrayContainer(int[] arr, int index) {
            this.arr = arr;
            this.index = index;
        }

        public int compareTo(ArrayContainer o) {
            return this.arr[this.index] - o.arr[o.index];
        }
    }

    public static int[] mergeKSortedArrays(int[][] arrays) {
        PriorityQueue<ArrayContainer> pq = new PriorityQueue<>();
        int totalLength = 0;
        for (int[] array : arrays) {
            totalLength += array.length;
            if (array.length > 0) {
                pq.add(new ArrayContainer(array, 0));
            }
        }

        int[] result = new int[totalLength];
        int i = 0;

        while (!pq.isEmpty()) {
            ArrayContainer ac = pq.poll();
            result[i++] = ac.arr[ac.index];

            if (ac.index < ac.arr.length - 1) {
                pq.add(new ArrayContainer(ac.arr, ac.index + 1));
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int[][] arrays = {
            {1, 4, 5},
            {1, 3, 4},
            {2, 6}
        };

        int[] result = mergeKSortedArrays(arrays);
        System.out.println("Merged array: ");
        for (int i : result) {
            System.out.print(i + " ");
        }
    }
}
```

Time Complexity: O(n log k), where n is the total number of elements, and k is the number of arrays.

---

6. Skyline Problem:

Problem Statement: Given a list of buildings in the form `(left, right, height)`, determine the skyline silhouette. Each building is a rectangle. The skyline is represented as the key points where the shape of the buildings changes.
Example: Buildings: `[(2, 9, 10), (3, 7, 15), (5, 12, 12), (15, 20, 10), (19, 24, 8)]` Skyline: `[(2, 10), (3, 15), (7, 12), (12, 0), (15, 10), (20, 8), (24, 0)]`
Java Code:

```java
import java.util.*;

class SkylineProblem {
    public List<int[]> getSkyline(int[][] buildings) {
        List<int[]> result = new ArrayList<>();
        List<int[]> height = new ArrayList<>();

        for (int[] b : buildings) {
            height.add(new int[]{b[0], -b[2]}); // start of building
            height.add(new int[]{b[1], b[2]});  // end of building
        }

        height.sort((a, b) -> (a[0] != b[0]) ? a[0] - b[0] : a[1] - b[1]);

        PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> b - a);
        pq.offer(0);

        int prevMaxHeight = 0;

        for (int[] h : height) {
            if (h[1] < 0) {
                pq.offer(-h[1]); // Add building height
            } else {
                pq.remove(h[1]); // Remove building height
            }

            int currMaxHeight = pq.peek();
            if (prevMaxHeight != currMaxHeight) {
                result.add(new int[]{h[0], currMaxHeight});
                prevMaxHeight = currMaxHeight;
            }
        }
        return result;
    }
    public static void main(String[] args) {
        SkylineProblem skyline = new SkylineProblem();
        int[][] buildings = {{2, 9, 10}, {3, 7, 15}, {5, 12, 12}, {15, 20, 10}, {19, 24,
8}};
        List<int[]> result = skyline.getSkyline(buildings);

        for (int[] point : result) {
            System.out.println(Arrays.toString(point));
        }
    }
}
```

Time Complexity: O(n log n), where n is the number of buildings.

---

7. Finding the Median of Two Sorted Arrays:

Problem Statement: Given two sorted arrays of size `m` and `n`, find the median in O(log(min(m,n))) time using the divide and conquer approach.
Example: For arrays:

`arr1 = [1, 3]`

`arr2 = [2]`

The median is `2.0`.
Java Code:

```java
class MedianOfTwoSortedArrays {
    public static double findMedianSortedArrays(int[] nums1, int[] nums2) {
        if (nums1.length > nums2.length) {
            return findMedianSortedArrays(nums2, nums1); // Ensure nums1 is smaller than
nums2
        }

        int x = nums1.length;
        int y = nums2.length;

        int low = 0, high = x;

        while (low <= high) {
            int partitionX = (low + high) / 2;
            int partitionY = (x + y + 1) / 2 - partitionX;

            int maxX = (partitionX == 0) ? Integer.MIN_VALUE : nums1[partitionX - 1];
            int minX = (partitionX == x) ? Integer.MAX_VALUE : nums1[partitionX];

            int maxY = (partitionY == 0) ? Integer.MIN_VALUE : nums2[partitionY - 1];
            int minY = (partitionY == y) ? Integer.MAX_VALUE : nums2[partitionY];

            if (maxX <= minY && maxY <= minX) {
                if ((x + y) % 2 == 0) {
                    return ((double)Math.max(maxX, maxY) + Math.min(minX, minY)) / 2;
                } else {
                    return (double)Math.max(maxX, maxY);
                }
            } else if (maxX > minY) {
                high = partitionX - 1;
            } else {
                low = partitionX + 1;
            }
        }

        throw new IllegalArgumentException("Input arrays are not sorted");
    }
    public static void main(String[] args) {
        int[] nums1 = {1, 3};
        int[] nums2 = {2};
        System.out.println("The median is " + findMedianSortedArrays(nums1, nums2));
    }
}
```

Time Complexity: O(log(min(m, n)))

---

8. Count the Number of Reverse Pairs in an Array:

Problem Statement: Count reverse pairs in an array where `` `(i, j)` `` is a reverse pair if `` `i < j` `` and `` `nums[i] > 2 * nums[j]` ``.
Example: For array: `` `[1, 3, 2, 3, 1]` ``,
The number of reverse pairs is `` `2` ``.
Java Code:

```java
class ReversePairs {
    public static int mergeAndCount(int[] nums, int left, int mid, int right) {
        int count = 0;
        int j = mid + 1;

        for (int i = left; i <= mid; i++) {
            while (j <= right && nums[i] > 2L * nums[j]) {
                j++;
            }
            count += (j - (mid + 1));
        }

        int[] temp = new int[right - left + 1];
        int i1 = left, i2 = mid + 1, k = 0;

        while (i1 <= mid && i2 <= right) {
            if (nums[i1] <= nums[i2]) {
                temp[k++] = nums[i1++];
            } else {
                temp[k++] = nums[i2++];
            }
        }

        while (i1 <= mid) temp[k++] = nums[i1++];
        while (i2 <= right) temp[k++] = nums[i2++];

        for (i1 = left, k = 0; i1 <= right; i1++, k++) {
            nums[i1] = temp[k];
        }

        return count;
    }

    public static int mergeSortAndCount(int[] nums, int left, int right) {
        if (left >= right) return 0;
        int mid = left + (right - left) / 2;
        int count = mergeSortAndCount(nums, left, mid);
        count += mergeSortAndCount(nums, mid + 1, right);
        count += mergeAndCount(nums, left, mid, right);
        return count;
    }

    public static void main(String[] args) {
        int[] nums = {1, 3, 2, 3, 1};
        System.out.println("Number of reverse pairs: " + mergeSortAndCount(nums, 0,
nums.length - 1));
```

```
        }
    }
```

Time Complexity: O(n log n)

---

9. Convex Hull Algorithm (Divide and Conquer):

Problem Statement: Find the convex hull of a set of points using the divide and conquer approach.
Example: Given points: ``[(0, 0), (1, 1), (2, 2), (0, 3), (2, 1)]``,
The convex hull is the polygon formed by connecting points: ``[(0, 0), (0, 3), (2, 1), (2, 2)]``.
Java Code:

```java
import java.util.ArrayList;
import java.util.List;

class ConvexHull {

    public static int orientation(int[] p, int[] q, int[] r) {
        int val = (q[1] - p[1]) * (r[0] - q[0]) - (q[0] - p[0]) * (r[1] - q[1]);
        if (val == 0) return 0; // collinear
        return (val > 0) ? 1 : 2; // 1: clockwise, 2: counterclockwise
    }

    public static List<int[]> convexHull(int[][] points) {
        if (points.length < 3) return null; // Convex hull not possible

        List<int[]> hull = new ArrayList<>();
        int leftmost = 0;

        for (int i = 1; i < points.length; i++) {
            if (points[i][0] < points[leftmost][0]) {
                leftmost = i;
            }
        }

        int p = leftmost, q;

        do {
            hull.add(points[p]);
            q = (p + 1) % points.length;

            for (int i = 0; i < points.length; i++) {
                if (orientation(points[p], points[i], points[q]) == 2) {
                    q = i;
                }
            }
            p = q;
        } while (p != leftmost);

        return hull;
    }

    public static void main(String[] args) {
        int[][] points = {{0, 0}, {1, 1}, {2, 2}, {0, 3}, {2, 1}};
        List<int[]> hull = convexHull(points);
        System.out.println("Convex Hull: ");
        for (int[] point : hull) {
            System.out.println("(" + point[0] + ", " + point[1] + ")");
        }
    }
}
```

Time Complexity: O(n log n)

These are the remaining problems, explained with examples and Java code. Let me know if you'd like further clarifications on any of the topics!