A page of a document

Description automatically generated

A white paper with black text

Description automatically generated

A white paper with black text

Description automatically generated

import java.util.Arrays;

public class MultiThreadedMergeSort {

    public static void main(String[] args) {

        int[] A = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};

        int[] B = new int[A.length];

        PMergeSort(A, 0, A.length - 1, B, 0);

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

    }

    public static void PMergeSort(int[] A, int p, int r, int[] B, int s) {

        int n = r - p + 1;

        if (n == 1) {

            B[s] = A[p];

        } else {

            int[] T = new int[n];

            int q1 = (p + r) / 2;

            int q2 = q1 - p + 1;

            final int pValue = p;

            final int q1Value = q1;

            final int q2Value = q2;

            Thread thread1 = new Thread(() -> {

                PMergeSort(A, pValue, q1Value, T, 0);

            });

            Thread thread2 = new Thread(() -> {

                PMergeSort(A, q1Value + 1, r, T, q2Value);

            });

            thread1.start();

            thread2.start();

            try {

                thread1.join();

                thread2.join();

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            PMerge(T, 0, q2 - 1, q2, n - 1, B, s);

        }

    }

    public static void PMerge(int[] T, int p1, int r1, int p2, int r2, int[] A, int p3) {

        int n1 = r1 - p1 + 1;

        int n2 = r2 - p2 + 1;

        if (n1 < n2) {

            int temp = p1;

            p1 = p2;

            p2 = temp;

            temp = r1;

            r1 = r2;

            r2 = temp;

            temp = n1;

            n1 = n2;

            n2 = temp;

        }

        if (n1 == 0) {

            return;

        } else {

            int q1 = (p1 + r1) / 2;

            int q2 = BinarySearch(T[q1], T, p2, r2);

            int q3 = p3 + (q1 - p1) + (q2 - p2);

            A[q3] = T[q1];

            final int p1Value = p1;

            final int q1Value = q1;

            final int r1Value = r1;

            final int p2Value = p2;

            final int q2Value = q2;

            final int p3Value = p3;

            final int r2Value = r2; // Create final local variable for r2

            Thread thread1 = new Thread(() -> {

                PMerge(T, p1Value, q1Value - 1, p2Value, r2Value - 1, A, p3Value); // Use r2Value inside lambda

            });

            Thread thread2 = new Thread(() -> {

                PMerge(T, q1Value + 1, r1Value, q2Value, r2Value, A, q3 + 1); // Use r2Value inside lambda

            });

            thread1.start();

            thread2.start();

            try {

                thread1.join();

                thread2.join();

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

    public static int BinarySearch(int x, int[] T, int p, int r) {

        int low = p;

        int high = Math.max(p, r + 1);

        while (low < high) {

            int mid = (low + high) / 2;

            if (x <= T[mid]) {

                high = mid;

            } else {

                low = mid + 1;

            }

        }

        return high;

    }

}

Output:

A black screen with white text

Description automatically generated

Optimization :

In this modified version, the binary search operation is moved out of the parallel sections, which reduces the overhead of creating threads for smaller tasks. This should improve the overall performance of the algorithm.

import java.util.Arrays;

public class optimization {

    public static void main(String[] args) {

        int[] A = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};

        int[] B = new int[A.length];

        PMergeSort(A, 0, A.length - 1, B, 0);

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

    }

    public static void PMergeSort(int[] A, int p, int r, int[] B, int s) {

        int n = r - p + 1;

        if (n == 1) {

            B[s] = A[p];

        } else {

            int[] T = new int[n];

            int q1 = (p + r) / 2;

            int q2 = q1 - p + 1;

            PMergeSort(A, p, q1, T, 0);

            PMergeSort(A, q1 + 1, r, T, q2);

            PMerge(T, 0, q2 - 1, q2, n - 1, B, s);

        }

    }

    public static void PMerge(int[] T, int p1, int r1, int p2, int r2, int[] A, int p3) {

        int n1 = r1 - p1 + 1;

        int n2 = r2 - p2 + 1;

        if (n1 < n2) {

            int temp = p1;

            p1 = p2;

            p2 = temp;

            temp = r1;

            r1 = r2;

            r2 = temp;

            temp = n1;

            n1 = n2;

            n2 = temp;

        }

        if (n1 == 0) {

            return;

        } else {

            int q1 = (p1 + r1) / 2;

            int q2 = BinarySearch(T[q1], T, p2, r2);

            int q3 = p3 + (q1 - p1) + (q2 - p2);

            A[q3] = T[q1];

            PMerge(T, p1, q1 - 1, p2, q2 - 1, A, p3);

            PMerge(T, q1 + 1, r1, q2, r2, A, q3 + 1);

        }

    }

    public static int BinarySearch(int x, int[] T, int p, int r) {

        int low = p;

        int high = Math.max(p, r + 1);

        while (low < high) {

            int mid = (low + high) / 2;

            if (x <= T[mid]) {

                high = mid;

            } else {

                low = mid + 1;

            }

        }

        return high;

    }

}

Output:

A black background with white and orange text

Description automatically generated

All code works correctly!