Object Oriented Computing (Java)

Merge Sort vs. Bubble Sort   
  
  
 This Java program, named SortComparison, compares the execution times of Bubble Sort and Merge Sort algorithms for sorting a randomly generated array of integers. The main method generates a large random array and creates two copies of it, one for each sorting algorithm. It then measures the execution time of both sorting algorithms using System.currentTimeMillis() before and after sorting. Finally, it prints out the execution times of Bubble Sort and Merge Sort. This program serves as a demonstration of the efficiency contrast between these two sorting techniques.  
  
  
  
  
Code:

import java.util.Arrays;

import java.util.Random;

public class SortComparison {

public static void main(String[] args) {

int[] randomArray = generateRandomArray(100000);

int[] bubbleSortArray = Arrays.copyOf(randomArray, randomArray.length);

int[] mergeSortArray = Arrays.copyOf(randomArray, randomArray.length);

long bubbleStartTime = System.currentTimeMillis();

bubbleSort(bubbleSortArray);

long bubbleEndTime = System.currentTimeMillis();

long bubbleExecutionTime = bubbleEndTime - bubbleStartTime;

long mergeStartTime = System.currentTimeMillis();

mergeSort(mergeSortArray, 0, mergeSortArray.length - 1);

long mergeEndTime = System.currentTimeMillis();

long mergeExecutionTime = mergeEndTime - mergeStartTime;

System.out.println("Bubble Sort execution time: " + bubbleExecutionTime + " milliseconds");

System.out.println("Merge Sort execution time: " + mergeExecutionTime + " milliseconds");

}

private static int[] generateRandomArray(int size) {

int[] array = new int[size];

Random random = new Random();

for (int i = 0; i < size; i++) {

array[i] = random.nextInt();

}

return array;

}

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 arr[j] and arr[j+1]

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

private static void mergeSort(int[] arr, int left, int right) {

if (left < right) {

int mid = (left + right) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

private static void merge(int[] arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int[] L = new int[n1];

int[] R = new int[n2];

for (int i = 0; i < n1; ++i)

L[i] = arr[left + i];

for (int j = 0; j < n2; ++j)

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

int i = 0, j = 0;

int k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

}