Selected files

4 printable files

Ex1.java
Ex2.java
Ex3.java
Ex4.java

Ex1.java

```java
// PartitionApp.java
// demonstrates partitioning an array
////////////////////////////////////////////////////////////////
// Exercises:
// 1. Add counters for the number of comparisons and swaps and display
//    them after partitioning

class ArrayPar {
    private long[] theArray; // reference to array theArray
    private int nElems; // number of data items
    private int compCount; // counter for comparisons
    private int swapCount; // counter for swaps

    // Constructor
    public ArrayPar(int max) {
        theArray = new long[max]; // create the array
        nElems = 0; // no items yet
        compCount = 0; // initialize comparison count
        swapCount = 0; // initialize swap count
    }

    // Insert an element into array
    public void insert(long value) {
        theArray[nElems] = value; // insert it
        nElems++; // increment size
    }

    // Return number of items
    public int size() {
        return nElems;
    }

    // Display array contents
    public void display() {
        System.out.print("A=");
        for (int j = 0; j < nElems; j++) {
            System.out.print(theArray[j] + " "); // display it
        }
        System.out.println("");
    }

    // Partition the array and count comparisons and swaps
    public int partitionIt(int left, int right, long pivot) {
```

```
        int leftPtr = left - 1; // right of first element
        int rightPtr = right + 1; // left of pivot
        while (true) {
            // Find bigger item (increment comparisons counter)
            while (leftPtr < right && theArray[++leftPtr] < pivot) {
                compCount++; // Increment comparison counter
            }

            // Find smaller item (increment comparisons counter)
            while (rightPtr > left && theArray[--rightPtr] > pivot) {
                compCount++; // Increment comparison counter
            }

            if (leftPtr >= rightPtr) // If pointers cross, partition done
                break;
            else { // Swap elements
                swap(leftPtr, rightPtr); // Swap elements
                swapCount++; // Increment swap counter
            }
        }
        return leftPtr; // Return partition index
    }

    // Swap two elements
    public void swap(int dex1, int dex2) {
        long temp = theArray[dex1]; // A into temp
        theArray[dex1] = theArray[dex2]; // B into A
        theArray[dex2] = temp; // Temp into B
    }

    // Get the number of comparisons
    public int getCompCount() {
        return compCount;
    }

    // Get the number of swaps
    public int getSwapCount() {
        return swapCount;
    }
}

public class PartitionApp {
    public static void main(String[] args) {
        int maxSize = 16; // array size
        ArrayPar arr; // reference to array
        arr = new ArrayPar(maxSize); // create the array

        // Fill array with random numbers
        for (int j = 0; j < maxSize; j++) {
            long n = (int) (java.lang.Math.random() * 199);
            arr.insert(n);
        }
        arr.display(); // Display unsorted array

```

```
        long pivot = 99; // pivot value
        System.out.print("Pivot is " + pivot);
        int size = arr.size();

        // Partition the array
        int partDex = arr.partitionIt(0, size - 1, pivot);

        System.out.println(", Partition is at index " + partDex);
        arr.display(); // Display partitioned array

        // Display the number of comparisons and swaps
        System.out.println("Number of comparisons: " + arr.getCompCount());
        System.out.println("Number of swaps: " + arr.getSwapCount());
    } // end main()
} // end class PartitionApp

```

Ex2.java

```
// PartitionApp.java
// demonstrates partitioning an array
////////////////////////////////////////////////////////////////
// Exercises:
// 2. Investigate the relationship between the index of partitioning,
//    the number of comparison, and the number of swaps.
class ArrayPar {
    private long[] theArray; // reference to the array theArray
    private int nElems; // number of data items
    private int compCount; // counter for comparisons
    private int swapCount; // counter for swaps

    // Constructor
    public ArrayPar(int max) {
        theArray = new long[max]; // create the array
        nElems = 0; // no items yet
        compCount = 0; // initialize comparison count
        swapCount = 0; // initialize swap count
    }

    // Insert an element into array
    public void insert(long value) {
        theArray[nElems] = value; // insert it
        nElems++; // increment size
    }

    // Return number of items
    public int size() {
        return nElems;
    }

    // Display array contents
    public void display() {
        System.out.print("A=");
        for (int j = 0; j < nElems; j++) {
```

```
            System.out.print(theArray[j] + " "); // display it
        }
        System.out.println("");
    }

    // Partition the array and count comparisons and swaps
    public int partitionIt(int left, int right, long pivot) {
        int leftPtr = left - 1; // right of first element
        int rightPtr = right + 1; // left of pivot
        while (true) {
            // Find bigger item (increment comparisons counter)
            while (leftPtr < right && theArray[++leftPtr] < pivot) {
                compCount++; // Increment comparison counter
            }

            // Find smaller item (increment comparisons counter)
            while (rightPtr > left && theArray[--rightPtr] > pivot) {
                compCount++; // Increment comparison counter
            }

            if (leftPtr >= rightPtr) // If pointers cross, partition done
                break;
            else { // Swap elements
                swap(leftPtr, rightPtr); // Swap elements
                swapCount++; // Increment swap counter
            }
        }
        return leftPtr; // Return partition index
    }

    // Swap two elements
    public void swap(int dex1, int dex2) {
        long temp = theArray[dex1]; // A into temp
        theArray[dex1] = theArray[dex2]; // B into A
        theArray[dex2] = temp; // Temp into B
    }

    // Get the number of comparisons
    public int getCompCount() {
        return compCount;
    }

    // Get the number of swaps
    public int getSwapCount() {
        return swapCount;
    }

    // Reset comparison and swap counts for new tests
    public void resetCounters() {
        compCount = 0;
        swapCount = 0;
    }

    // Getter for theArray to access it from outside
```

```
90      public long[] getArray() {
91          return theArray;
92      }
93  }
94
95  public class PartitionApp {
96      public static void main(String[] args) {
97          int maxSize = 16; // Array size
98          int runs = 100; // Number of runs for average
99          long totalComparisons = 0;
100         long totalSwaps = 0;
101
102         // Perform multiple runs to collect data
103         for (int i = 0; i < runs; i++) {
104             ArrayPar arr = new ArrayPar(maxSize); // Reset array for each run
105
106             // Fill array with random numbers
107             for (int j = 0; j < maxSize; j++) {
108                 long n = (int) (java.lang.Math.random() * 199);
109                 arr.insert(n);
110             }
111
112             // Pivot selection strategy
113             long pivot = arr.getArray()[0]; // First element as pivot
114             // Uncomment one of the following lines for different pivot strategies
115             // long pivot = arr.getArray()[arr.size() - 1]; // Last element as pivot
116             // long pivot = arr.getArray()[arr.size() / 2]; // Middle element as pivot
117
118             // Partition the array and track comparisons and swaps
119             arr.resetCounters(); // Reset comparison and swap counters for each run
120             int partitionIndex = arr.partitionIt(0, arr.size() - 1, pivot);
121
122             totalComparisons += arr.getCompCount();
123             totalSwaps += arr.getSwapCount();
124
125             // Print detailed information for investigation
126             System.out.println("Run " + (i + 1) + ": ");
127             System.out.println("Pivot = " + pivot);
128             System.out.println("Partition Index = " + partitionIndex);
129             System.out.println("Comparisons = " + arr.getCompCount());
130             System.out.println("Swaps = " + arr.getSwapCount());
131             System.out.println("Array after partition: ");
132             arr.display();
133             System.out.println();
134         }
135
136         // Compute average comparisons and swaps
137         double avgComparisons = totalComparisons / (double) runs;
138         double avgSwaps = totalSwaps / (double) runs;
139
140         System.out.println("Average number of comparisons over " + runs + " runs: " +
    avgComparisons);
141         System.out.println("Average number of swaps over " + runs + " runs: " + avgSwaps);
142     }
```

```
}

```

Ex3.java

```

// PartitionApp.java
// demonstrates partitioning an array
////////////////////////////////////////////////////////////////
// Exercises:
// 3. Do Exercise 2 with different pivots:
//    - beginning, end, or middle of the interval;
//    - selected at random from the interval or from a larger interval;
//    - last item in the array.
import java.util.Random;

class ArrayPar {
    private long[] theArray; // reference to the array theArray
    private int nElems; // number of data items
    private int compCount; // counter for comparisons
    private int swapCount; // counter for swaps

    // Constructor
    public ArrayPar(int max) {
        theArray = new long[max]; // create the array
        nElems = 0; // no items yet
        compCount = 0; // initialize comparison count
        swapCount = 0; // initialize swap count
    }

    // Insert an element into array
    public void insert(long value) {
        theArray[nElems] = value; // insert it
        nElems++; // increment size
    }

    // Return number of items
    public int size() {
        return nElems;
    }

    // Display array contents
    public void display() {
        System.out.print("A=");
        for (int j = 0; j < nElems; j++) {
            System.out.print(theArray[j] + " "); // display it
        }
        System.out.println("");
    }

    // Partition the array and count comparisons and swaps
    public int partitionIt(int left, int right, long pivot) {
        int leftPtr = left - 1; // right of first element
        int rightPtr = right + 1; // left of pivot
```

```
50          while (true) {
51              // Find bigger item (increment comparisons counter)
52              while (leftPtr < right && theArray[++leftPtr] < pivot) {
53                  compCount++; // Increment comparison counter
54              }
55
56              // Find smaller item (increment comparisons counter)
57              while (rightPtr > left && theArray[--rightPtr] > pivot) {
58                  compCount++; // Increment comparison counter
59              }
60
61              if (leftPtr >= rightPtr) // If pointers cross, partition done
62                  break;
63              else { // Swap elements
64                  swap(leftPtr, rightPtr); // Swap elements
65                  swapCount++; // Increment swap counter
66              }
67          }
68          return leftPtr; // Return partition index
69      }
70
71      // Swap two elements
72      public void swap(int dex1, int dex2) {
73          long temp = theArray[dex1]; // A into temp
74          theArray[dex1] = theArray[dex2]; // B into A
75          theArray[dex2] = temp; // Temp into B
76      }
77
78      // Get the number of comparisons
79      public int getCompCount() {
80          return compCount;
81      }
82
83      // Get the number of swaps
84      public int getSwapCount() {
85          return swapCount;
86      }
87
88      // Reset comparison and swap counts for new tests
89      public void resetCounters() {
90          compCount = 0;
91          swapCount = 0;
92      }
93
94      // Getter for theArray to access it from outside
95      public long[] getArray() {
96          return theArray;
97      }
98  }
99
100 public class PartitionApp {
101     public static void main(String[] args) {
102         int maxSize = 16; // Array size
103         int runs = 100; // Number of runs for average
```

```
104         long totalComparisons = 0;
105         long totalSwaps = 0;
106 
107         // Create an instance of Random for selecting random pivots
108         Random rand = new Random();
109 
110         // Perform multiple runs to collect data for different pivot strategies
111         for (int i = 0; i < runs; i++) {
112             ArrayPar arr = new ArrayPar(maxSize); // Reset array for each run
113 
114             // Fill array with random numbers
115             for (int j = 0; j < maxSize; j++) {
116                 long n = (int) (java.lang.Math.random() * 199);
117                 arr.insert(n);
118             }
119 
120             // Select pivot based on one of the strategies
121             long pivot = selectPivot(arr, rand);
122 
123             // Partition the array and track comparisons and swaps
124             arr.resetCounters(); // Reset comparison and swap counters for each run
125             int partitionIndex = arr.partitionIt(0, arr.size() - 1, pivot);
126 
127             totalComparisons += arr.getCompCount();
128             totalSwaps += arr.getSwapCount();
129 
130             // Print detailed information for investigation
131             System.out.println("Run " + (i + 1) + ": ");
132             System.out.println("Pivot = " + pivot);
133             System.out.println("Partition Index = " + partitionIndex);
134             System.out.println("Comparisons = " + arr.getCompCount());
135             System.out.println("Swaps = " + arr.getSwapCount());
136             System.out.println("Array after partition: ");
137             arr.display();
138             System.out.println();
139         }
140 
141         // Compute average comparisons and swaps
142         double avgComparisons = totalComparisons / (double) runs;
143         double avgSwaps = totalSwaps / (double) runs;
144 
145         System.out.println("Average number of comparisons over " + runs + " runs: " +
    avgComparisons);
146         System.out.println("Average number of swaps over " + runs + " runs: " + avgSwaps);
147     }
148 
149     // Method to select pivot based on different strategies
150     private static long selectPivot(ArrayPar arr, Random rand) {
151         int size = arr.size();
152 
153         // Select pivot based on different strategies:
154         // 1. First element (beginning of the interval)
155         long pivot1 = arr.getArray()[0];
156 
```

```
        // 2. Last element (end of the interval)
        long pivot2 = arr.getArray()[size - 1];

        // 3. Middle element (middle of the interval)
        long pivot3 = arr.getArray()[size / 2];

        // 4. Random element from the array
        long pivot4 = arr.getArray()[rand.nextInt(size)];

        // 5. Last item in the array
        long pivot5 = arr.getArray()[size - 1];

        // We will experiment with one of these pivot strategies at a time
        // For demonstration, let's choose the pivot strategy at random:
        int strategy = rand.nextInt(5); // Randomly select one of the 5 pivot strategies

        // Return the pivot based on the strategy chosen
        switch (strategy) {
            case 0:
                return pivot1; // First element
            case 1:
                return pivot2; // Last element
            case 2:
                return pivot3; // Middle element
            case 3:
                return pivot4; // Random element
            case 4:
                return pivot5; // Last item (same as last element)
            default:
                return pivot1; // Default to first element if something goes wrong
        }
    }
}

```

Ex4.java

```
// 4. Compute the average number of comparisons and swaps over 100 runs.
class ArrayPar {
    private long[] theArray; // reference to array theArray
    private int nElems; // number of data items
    private int compCount; // counter for comparisons
    private int swapCount; // counter for swaps

    // Constructor
    public ArrayPar(int max) {
        theArray = new long[max]; // create the array
        nElems = 0; // no items yet
        compCount = 0; // initialize comparison count
        swapCount = 0; // initialize swap count
    }

    // Insert an element into array
    public void insert(long value) {
```

```
        theArray[nElems] = value; // insert it
        nElems++; // increment size
    }

    // Return number of items
    public int size() {
        return nElems;
    }

    // Display array contents
    public void display() {
        System.out.print("A=");
        for (int j = 0; j < nElems; j++) {
            System.out.print(theArray[j] + " "); // display it
        }
        System.out.println("");
    }

    // Partition the array and count comparisons and swaps
    public int partitionIt(int left, int right, long pivot) {
        int leftPtr = left - 1; // right of first element
        int rightPtr = right + 1; // left of pivot
        while (true) {
            // Find bigger item (increment comparisons counter)
            while (leftPtr < right && theArray[++leftPtr] < pivot) {
                compCount++; // Increment comparison counter
            }

            // Find smaller item (increment comparisons counter)
            while (rightPtr > left && theArray[--rightPtr] > pivot) {
                compCount++; // Increment comparison counter
            }

            if (leftPtr >= rightPtr) // If pointers cross, partition done
                break;
            else { // Swap elements
                swap(leftPtr, rightPtr); // Swap elements
                swapCount++; // Increment swap counter
            }
        }
        return leftPtr; // Return partition index
    }

    // Swap two elements
    public void swap(int dex1, int dex2) {
        long temp = theArray[dex1]; // A into temp
        theArray[dex1] = theArray[dex2]; // B into A
        theArray[dex2] = temp; // Temp into B
    }

    // Get the number of comparisons
    public int getCompCount() {
        return compCount;
    }
```

```
72
73      // Get the number of swaps
74      public int getSwapCount() {
75          return swapCount;
76      }
77
78      // Reset comparison and swap counts for new tests
79      public void resetCounters() {
80          compCount = 0;
81          swapCount = 0;
82      }
83
84      // Getter for theArray
85      public long[] getArray() {
86          return theArray;
87      }
88  }
89
90  public class PartitionApp {
91      public static void main(String[] args) {
92          int maxSize = 16; // array size
93          int runs = 100; // number of runs for average
94          long totalComparisons = 0;
95          long totalSwaps = 0;
96
97          // Perform 100 runs to compute average comparisons and swaps
98          for (int i = 0; i < runs; i++) {
99              ArrayPar arr = new ArrayPar(maxSize); // reset array for each run
100
101             // Fill array with random numbers
102             for (int j = 0; j < maxSize; j++) {
103                 long n = (int) (java.lang.Math.random() * 199);
104                 arr.insert(n);
105             }
106
107             // Pivot selection strategy
108             long pivot = arr.getArray()[0]; // First element as pivot
109             // Uncomment one of the following lines for different pivot strategies
110             // long pivot = arr.getArray()[arr.size() - 1]; // Last element as pivot
111             // long pivot = arr.getArray()[arr.size() / 2]; // Middle element as pivot
112
113             // Partition the array and get partition index
114             arr.resetCounters(); // Reset comparison and swap counters for each run
115             arr.partitionIt(0, arr.size() - 1, pivot);
116
117             totalComparisons += arr.getCompCount();
118             totalSwaps += arr.getSwapCount();
119         }
120
121         // Compute average comparisons and swaps
122         double avgComparisons = totalComparisons / (double) runs;
123         double avgSwaps = totalSwaps / (double) runs;
124
```

```
        System.out.println("Average number of comparisons over " + runs + " runs: " +
avgComparisons);
        System.out.println("Average number of swaps over " + runs + " runs: " + avgSwaps);
    }
}

```