

Selected files

4 printable files

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

Ex1.java

```
1 // PartitionApp.java
2 // demonstrates partitioning an array
3 ///////////////////////////////////////////////////////////////////
4 // Exercises:
5 // 1. Add counters for the number of comparisons and swaps and display
6 //    them after partitioning
7
8 class ArrayPar {
9     private long[] theArray; // reference to array theArray
10    private int nElems; // number of data items
11    private int compCount; // counter for comparisons
12    private int swapCount; // counter for swaps
13
14    // Constructor
15    public ArrayPar(int max) {
16        theArray = new long[max]; // create the array
17        nElems = 0; // no items yet
18        compCount = 0; // initialize comparison count
19        swapCount = 0; // initialize swap count
20    }
21
22    // Insert an element into array
23    public void insert(long value) {
24        theArray[nElems] = value; // insert it
25        nElems++; // increment size
26    }
27
28    // Return number of items
29    public int size() {
30        return nElems;
31    }
32
33    // Display array contents
34    public void display() {
35        System.out.print("A=");
36        for (int j = 0; j < nElems; j++) {
37            System.out.print(theArray[j] + " "); // display it
38        }
39        System.out.println("");
40    }
41
42    // Partition the array and count comparisons and swaps
43    public int partitionIt(int left, int right, long pivot) {
```

```

44     int leftPtr = left - 1; // right of first element
45     int rightPtr = right + 1; // left of pivot
46     while (true) {
47         // Find bigger item (increment comparisons counter)
48         while (leftPtr < right && theArray[++leftPtr] < pivot) {
49             compCount++; // Increment comparison counter
50         }
51
52         // Find smaller item (increment comparisons counter)
53         while (rightPtr > left && theArray[--rightPtr] > pivot) {
54             compCount++; // Increment comparison counter
55         }
56
57         if (leftPtr >= rightPtr) // If pointers cross, partition done
58             break;
59         else { // Swap elements
60             swap(leftPtr, rightPtr); // Swap elements
61             swapCount++; // Increment swap counter
62         }
63     }
64     return leftPtr; // Return partition index
65 }
66
67 // Swap two elements
68 public void swap(int dex1, int dex2) {
69     long temp = theArray[dex1]; // A into temp
70     theArray[dex1] = theArray[dex2]; // B into A
71     theArray[dex2] = temp; // Temp into B
72 }
73
74 // Get the number of comparisons
75 public int getCompCount() {
76     return compCount;
77 }
78
79 // Get the number of swaps
80 public int getSwapCount() {
81     return swapCount;
82 }
83 }
84
85 public class PartitionApp {
86     public static void main(String[] args) {
87         int maxSize = 16; // array size
88         ArrayPar arr; // reference to array
89         arr = new ArrayPar(maxSize); // create the array
90
91         // Fill array with random numbers
92         for (int j = 0; j < maxSize; j++) {
93             long n = (int) (java.lang.Math.random() * 199);
94             arr.insert(n);
95         }
96         arr.display(); // Display unsorted array
97

```

```

98     long pivot = 99; // pivot value
99     System.out.print("Pivot is " + pivot);
100    int size = arr.size();
101
102    // Partition the array
103    int partDex = arr.partitionIt(0, size - 1, pivot);
104
105    System.out.println(", Partition is at index " + partDex);
106    arr.display(); // Display partitioned array
107
108    // Display the number of comparisons and swaps
109    System.out.println("Number of comparisons: " + arr.getCompCount());
110    System.out.println("Number of swaps: " + arr.getSwapCount());
111 } // end main()
112 } // end class PartitionApp
113

```

Ex2.java

```

1 // PartitionApp.java
2 // demonstrates partitioning an array
3 ///////////////////////////////////////////////////////////////////
4 // Exercises:
5 // 2. Investigate the relationship between the index of partitioning,
6 //    the number of comparison, and the number of swaps.
7 class ArrayPar {
8     private long[] theArray; // reference to the array theArray
9     private int nElems; // number of data items
10    private int compCount; // counter for comparisons
11    private int swapCount; // counter for swaps
12
13    // Constructor
14    public ArrayPar(int max) {
15        theArray = new long[max]; // create the array
16        nElems = 0; // no items yet
17        compCount = 0; // initialize comparison count
18        swapCount = 0; // initialize swap count
19    }
20
21    // Insert an element into array
22    public void insert(long value) {
23        theArray[nElems] = value; // insert it
24        nElems++; // increment size
25    }
26
27    // Return number of items
28    public int size() {
29        return nElems;
30    }
31
32    // Display array contents
33    public void display() {
34        System.out.print("A=");
35        for (int j = 0; j < nElems; j++) {

```

```

36         System.out.print(theArray[j] + " "); // display it
37     }
38     System.out.println("");
39 }
40
41 // Partition the array and count comparisons and swaps
42 public int partitionIt(int left, int right, long pivot) {
43     int leftPtr = left - 1; // right of first element
44     int rightPtr = right + 1; // left of pivot
45     while (true) {
46         // Find bigger item (increment comparisons counter)
47         while (leftPtr < right && theArray[++leftPtr] < pivot) {
48             compCount++; // Increment comparison counter
49         }
50
51         // Find smaller item (increment comparisons counter)
52         while (rightPtr > left && theArray[--rightPtr] > pivot) {
53             compCount++; // Increment comparison counter
54         }
55
56         if (leftPtr >= rightPtr) // If pointers cross, partition done
57             break;
58         else { // Swap elements
59             swap(leftPtr, rightPtr); // Swap elements
60             swapCount++; // Increment swap counter
61         }
62     }
63     return leftPtr; // Return partition index
64 }
65
66 // Swap two elements
67 public void swap(int dex1, int dex2) {
68     long temp = theArray[dex1]; // A into temp
69     theArray[dex1] = theArray[dex2]; // B into A
70     theArray[dex2] = temp; // Temp into B
71 }
72
73 // Get the number of comparisons
74 public int getCompCount() {
75     return compCount;
76 }
77
78 // Get the number of swaps
79 public int getSwapCount() {
80     return swapCount;
81 }
82
83 // Reset comparison and swap counts for new tests
84 public void resetCounters() {
85     compCount = 0;
86     swapCount = 0;
87 }
88
89 // 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    }

```

```
143 }
144
```

Ex3.java

```
1
2 // PartitionApp.java
3 // demonstrates partitioning an array
4 ///////////////////////////////////////////////////////////////////
5 // Exercises:
6 // 3. Do Exercise 2 with different pivots:
7 //   - beginning, end, or middle of the interval;
8 //   - selected at random from the interval or from a larger interval;
9 //   - last item in the array.
10 import java.util.Random;
11
12 class ArrayPar {
13     private long[] theArray; // reference to the array theArray
14     private int nElems; // number of data items
15     private int compCount; // counter for comparisons
16     private int swapCount; // counter for swaps
17
18     // Constructor
19     public ArrayPar(int max) {
20         theArray = new long[max]; // create the array
21         nElems = 0; // no items yet
22         compCount = 0; // initialize comparison count
23         swapCount = 0; // initialize swap count
24     }
25
26     // Insert an element into array
27     public void insert(long value) {
28         theArray[nElems] = value; // insert it
29         nElems++; // increment size
30     }
31
32     // Return number of items
33     public int size() {
34         return nElems;
35     }
36
37     // Display array contents
38     public void display() {
39         System.out.print("A=");
40         for (int j = 0; j < nElems; j++) {
41             System.out.print(theArray[j] + " "); // display it
42         }
43         System.out.println("");
44     }
45
46     // Partition the array and count comparisons and swaps
47     public int partitionIt(int left, int right, long pivot) {
48         int leftPtr = left - 1; // right of first element
49         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: " +
146     avgComparisons);
147     System.out.println("Average number of swaps over " + runs + " runs: " + avgSwaps);
148 }
149
150 // Method to select pivot based on different strategies
151 private static long selectPivot(ArrayPar arr, Random rand) {
152     int size = arr.size();
153
154     // Select pivot based on different strategies:
155     // 1. First element (beginning of the interval)
156     long pivot1 = arr.getArray()[0];

```



```

157 // 2. Last element (end of the interval)
158 long pivot2 = arr.getArray()[size - 1];
159
160 // 3. Middle element (middle of the interval)
161 long pivot3 = arr.getArray()[size / 2];
162
163 // 4. Random element from the array
164 long pivot4 = arr.getArray()[rand.nextInt(size)];
165
166 // 5. Last item in the array
167 long pivot5 = arr.getArray()[size - 1];
168
169 // We will experiment with one of these pivot strategies at a time
170 // For demonstration, let's choose the pivot strategy at random:
171 int strategy = rand.nextInt(5); // Randomly select one of the 5 pivot strategies
172
173 // Return the pivot based on the strategy chosen
174 switch (strategy) {
175     case 0:
176         return pivot1; // First element
177     case 1:
178         return pivot2; // Last element
179     case 2:
180         return pivot3; // Middle element
181     case 3:
182         return pivot4; // Random element
183     case 4:
184         return pivot5; // Last item (same as last element)
185     default:
186         return pivot1; // Default to first element if something goes wrong
187 }
188 }
189 }
190

```

Ex4.java

```

1 // 4. Compute the average number of comparisons and swaps over 100 runs.
2 class ArrayPar {
3     private long[] theArray; // reference to array theArray
4     private int nElems; // number of data items
5     private int compCount; // counter for comparisons
6     private int swapCount; // counter for swaps
7
8     // Constructor
9     public ArrayPar(int max) {
10         theArray = new long[max]; // create the array
11         nElems = 0; // no items yet
12         compCount = 0; // initialize comparison count
13         swapCount = 0; // initialize swap count
14     }
15
16     // Insert an element into array
17     public void insert(long value) {

```

```

18     theArray[nElems] = value; // insert it
19     nElems++; // increment size
20 }
21
22 // Return number of items
23 public int size() {
24     return nElems;
25 }
26
27 // Display array contents
28 public void display() {
29     System.out.print("A=");
30     for (int j = 0; j < nElems; j++) {
31         System.out.print(theArray[j] + " "); // display it
32     }
33     System.out.println("");
34 }
35
36 // Partition the array and count comparisons and swaps
37 public int partitionIt(int left, int right, long pivot) {
38     int leftPtr = left - 1; // right of first element
39     int rightPtr = right + 1; // left of pivot
40     while (true) {
41         // Find bigger item (increment comparisons counter)
42         while (leftPtr < right && theArray[++leftPtr] < pivot) {
43             compCount++; // Increment comparison counter
44         }
45
46         // Find smaller item (increment comparisons counter)
47         while (rightPtr > left && theArray[--rightPtr] > pivot) {
48             compCount++; // Increment comparison counter
49         }
50
51         if (leftPtr >= rightPtr) // If pointers cross, partition done
52             break;
53         else { // Swap elements
54             swap(leftPtr, rightPtr); // Swap elements
55             swapCount++; // Increment swap counter
56         }
57     }
58     return leftPtr; // Return partition index
59 }
60
61 // Swap two elements
62 public void swap(int dex1, int dex2) {
63     long temp = theArray[dex1]; // A into temp
64     theArray[dex1] = theArray[dex2]; // B into A
65     theArray[dex2] = temp; // Temp into B
66 }
67
68 // Get the number of comparisons
69 public int getCompCount() {
70     return compCount;
71 }

```

```

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

```

```
125         System.out.println("Average number of comparisons over " + runs + " runs: " +  
    avgComparisons);  
126         System.out.println("Average number of swaps over " + runs + " runs: " + avgSwaps);  
127     }  
128 }  
129
```