

C:\Users\Rich\Documents\NetBeansProjects\Lab12\src\SortingVoterClient.java

```

1
2 import java.util.Scanner;
3
4 /**
5  * This class test the running time of different sorting algorithms.
6  * to test for smaller set uncomment the section. and change limitMax to be greater than limit.
7  * @author Richelin Metellus
8  * @version 04/21/2017
9  */
10 public class SortingVoterClient {
11
12     public static void main(String[] args) {
13         // System.out.println("How many voters to create? ");
14         // Scanner scan = new Scanner(System.in) ;
15         // int limit = scan.nextInt();
16
17         int limit = 1000000; // size for slower sorting algorithm
18         Voter[] voters = new Voter[limit];
19         Voter[] votersCopy;
20
21
22         int limitMax = 1000000; // size for faster sorting. need to modify for test sets
23         Voter[] largerVoters = new Voter[limitMax];
24         Voter[] largerVotersCopy;
25
26
27         //testing for some special case
28         // voters[0] = new Voter(115, "Ama", "democrat", "No");
29         // voters[1] = new Voter(112, "zor", "democrat", "Yes");
30         // voters[2] = new Voter(23, "Amet", "republican", "No");
31         // voters[3] = new Voter(12, "Joe", "independent", "No");
32         // voters[4] = new Voter(32, "Aman", "other", "Yes");
33         // voters[5] = new Voter(45, "Nadie", "democrat", "No");
34         // voters[6] = new Voter(12, "Joa", "republican", "Yes");
35         // voters[7] = new Voter(15, "For", "other", "Yes");
36
37         for(int i = 0; i < limitMax; i++){
38             if( i < limit)
39             {
40                 Voter temp = new Voter();
41                 voters[i] = temp;
42                 largerVoters[i] = temp;
43             }
44             else
45                 largerVoters[i] = new Voter(); // create more voter for larger set.
46
47         }
48
49         // printArray(voters);
50         // System.out.println("LargestArray");
51         // printArray(largerVoters);
52
53         // Comparator nameComp = new NameComparator();
54         // Sort.simpleBubbleSort(voters, nameComp);
55         // System.out.println("Sorted array by Name using bubble sort \n-----");
56         // printArray(voters);
57
58         // Comparator name2Comp = new NameComparator();
59         // Sort.insertionSort(voters, name2Comp);
60         // System.out.println("Sorted array by name using insertionSort\n-----");
61         // printArray(voters);
62

```

```
63     ArrayBag<Comparator<Voter>> compBag = new ArrayBag(4);
64     compBag.add(new PartyComparator()); // index 0
65     compBag.add(new DecisionComparator()); // index 1
66     compBag.add(new NameComparator()); // index 2; lower priority index.
67     System.out.println("");
68
69     Comparator idComp = new IdComparator();
70
71 //***** Runtime for mergeSort *****
72     Comparator voterNameComp = compBag.get(2);
73     largerVotersCopy = arrayClone(voters);
74     long mergStartTime = System.currentTimeMillis();
75     Sort.mergeSort(largerVotersCopy, voterNameComp);
76     long mergEndTime = System.currentTimeMillis();
77     long mergElapsedTime = mergEndTime - mergStartTime;
78     System.out.printf("Runtime of merge Sort(Name) \t for N\t = %,7d \t time \t = %,10d miliseconds \n",limitMax,mergElapsedTime);
79 //     System.out.println("Sorted array by Name using merge sort \n-----");
80 //     printArray(largerVotersCopy);
81
82 //***** Runtime for quickSort *****
83     Comparator voterPartyComp = compBag.get(0);
84     largerVotersCopy = arrayClone(voters);
85     long quickStartTime = System.currentTimeMillis();
86     Sort.quickSortInPlace(largerVotersCopy, voterPartyComp,0,largerVotersCopy.length-1);
87     long quickEndTime = System.currentTimeMillis();
88     long quickElapsedTime = quickEndTime - quickStartTime;
89     System.out.printf("Runtime of quickSort(Party) \t for N\t = %,7d \t time \t = %,10d miliseconds \n",limitMax,quickElapsedTime);
90 //     System.out.println("Sorted array by party using quick sort \n-----");
91 //     printArray(largerVotersCopy);
92
93 //***** Runtime for bubbleSort *****
94     votersCopy = arrayClone(voters);
95     long bubbleStartTime = System.currentTimeMillis();
96     Sort.simpleBubbleSort(votersCopy, idComp);
97     long bubbleEndTime = System.currentTimeMillis();
98     long bubbleElapsedTime = bubbleEndTime - bubbleStartTime;
99     System.out.printf("Runtime of bubbleSort(ID) \t for N\t = %,7d \t time \t = %,10d miliseconds \n",limit,bubbleElapsedTime);
100 //     System.out.println("Sorted array by id using bubble sort \n-----");
101 //     printArray(votersCopy);
102
103 //***** Runtime for InsertionSort *****
104     Comparator votedComp = compBag.get(1);
105     votersCopy = arrayClone(voters);
106     long inserStartTime = System.currentTimeMillis();
107     Sort.insertionSort(votersCopy, votedComp);
108     long inserEndTime = System.currentTimeMillis();
109     long inserElapsedTime = inserEndTime - inserStartTime;
110     System.out.printf("Runtime of insertionSort (Voted) \t for N\t = %,7d \t time \t = %,10d miliseconds \n",limit,inserElapsedTime);
111 //     System.out.println("Sorted array by voted status using insertion sort \n-----");
112 //     printArray(votersCopy);
113
114 //***** Runtime for SelectionSort *****
115     Comparator partyComp = compBag.get(0);
116     votersCopy = arrayClone(voters);
117     long selStartTime = System.currentTimeMillis();
118     Sort.selectionSort(votersCopy, partyComp);
119     long selEndTime = System.currentTimeMillis();
120     long selElapsedTime = selEndTime - selStartTime;
121     System.out.printf("Runtime of selectionSort(Party) \t for N\t = %,7d \t time \t = %,10d miliseconds \n",limit,selElapsedTime);
122 //     System.out.println("Sorted array by party using selectionSort\n-----");
123 //     printArray(votersCopy);
124
125 //***** Runtime for radixSort *****
126     largerVotersCopy = arrayClone(largerVoters);
127     long radixStartTime = System.currentTimeMillis();
```

```

128 Sort.radixSort(largerVotersCopy, compBag);
129 long radixEndTime = System.currentTimeMillis();
130 long radixElapsedTime = radixEndTime - radixStartTime;
131 System.out.printf("Runtime of radix Sort          \t for N\t = %,7d \t time \t= %,10d miliseconds \n\n",limitMax,radixElapsedTime);
132
133 //***** Printing run time of each soring algorithm *****
134 System.out.printf("Runtime of bubbleSort(ID)          \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,bubbleElapsedTime);
135 System.out.printf("Runtime of insertionSort (Voted) \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,inserElapsedTime);
136 System.out.printf("Runtime of selectionSort(Party) \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,selElapsedTime);
137 System.out.printf("Runtime of quickSort(Party) \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,quickElapsedTime);
138 System.out.printf("Runtime of merge Sort(Name) \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,mergElapsedTime);
139 System.out.printf("Runtime of radix Sort          \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,radixElapsedTime);
140
141
142 //      System.out.println("Sorted array by party using radixSort\n-----");
143 //      printArray(largerVotersCopy);
144 //
145 //      votedComp = new DecisionComparator();
146 //      Sort.mergeSort(voters,votedComp );
147 //      System.out.println("Sorted array by decision using merge sort \n-----");
148 //      printArray(voters);
149
150
151
152
153
154 }
155 public static void printArray( Voter[] data )
156 {
157     for (Voter legalVoter : data) {
158         System.out.println(legalVoter);
159     }
160     System.out.println("");
161 }
162 public static Voter[] arrayClone(Voter[] parent)
163 {
164     int parentSize = parent.length;
165     Voter[] clone = new Voter [parentSize];
166     for(int i = 0; i < parentSize; ++i)
167     {
168         clone[i] = parent[i];
169     }
170     return clone;
171 }
172 }

```