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

```java

import java.util.Scanner;

/**
 * This class test the running time of different sorting algorithms.
 * to test for smaller set uncomment the section. and change limitMax to be greater than limit.
 * @author Richelin Metellus
 * @version 04/21/2017
 */
public class SortingVoterClient {

    public static void main(String[] args) {
//        System.out.println("How many  voters to create? ");
//        Scanner scan = new Scanner(System.in) ;
//        int limit = scan.nextInt();

        int limit = 1000000;           // size for slower sorting algorithm
        Voter[] voters = new Voter[limit];
        Voter[] votersCopy;


        int limitMax = 1000000;     // size for faster sorting. need to modify for test sets
        Voter[] largerVoters = new Voter[limitMax];
        Voter[] largerVotersCopy;


        //testing for some special case
//        voters[0] = new Voter(115, "Ama", "democrat", "No");
//        voters[1] = new Voter(112, "zor", "democrat", "Yes");
//        voters[2] = new Voter(23, "Amet", "republican", "No");
//        voters[3] = new Voter(12, "Joe", "independent", "No");
//        voters[4] = new Voter(32, "Aman", "other", "Yes");
//        voters[5] = new Voter(45, "Nadie", "democrat", "No");
//        voters[6] = new Voter(12, "Joa", "republican", "Yes");
//        voters[7] = new Voter(15, "For", "other", "Yes");

        for(int i = 0; i < limitMax; i++){
           if( i < limit)
           {
               Voter temp = new Voter();
               voters[i]  = temp;
               largerVoters[i] = temp;
           }
           else
               largerVoters[i] = new Voter();      // create more voter for larger set.

        }

//        printArray(voters);
//        System.out.println("LargestArray");
//        printArray(largerVoters);

//        Comparator nameComp = new NameComparator();
//        Sort.simpleBubbleSort(voters, nameComp);
//        System.out.println("Sorted array by Name using bubble sort \n------------------");
//        printArray(voters);
//
//        Comparator name2Comp = new NameComparator();
//        Sort.insertionSort(voters, name2Comp);
//        System.out.println("Sorted array by name using insertionSort\n----------------------");
//        printArray(voters);

```

```java
        ArrayBag<Comparator<Voter>> compBag = new ArrayBag(4);
        compBag.add(new PartyComparator());     // index 0
        compBag.add(new DecisionComparator());  // index 1
        compBag.add(new NameComparator());      // index 2; lower priority index.
        System.out.println("");

        Comparator idComp = new IdComparator();

//************************* Runtime for mergeSort ******************************
        Comparator voterNameComp = compBag.get(2);
        largerVotersCopy = arrayClone(voters);
        long mergStartTime = System.currentTimeMillis();
        Sort.mergeSort(largerVotersCopy, voterNameComp);
        long mergEndTime = System.currentTimeMillis();
        long mergElapsedTime = mergEndTime - mergStartTime;
        System.out.printf("Runtime of merge Sort(Name)         \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,mergElapsedTime);
//        System.out.println("Sorted array by Name using merge sort \n------------------");
//        printArray(largerVotersCopy);

//***************************** Runtime for quickSort *****************
        Comparator voterPartyComp = compBag.get(0);
        largerVotersCopy = arrayClone(voters);
        long quickStartTime = System.currentTimeMillis();
        Sort.quickSortInPlace(largerVotersCopy, voterPartyComp,0,largerVotersCopy.length-1);
        long quickEndTime = System.currentTimeMillis();
        long quickElapsedTime = quickEndTime - quickStartTime;
        System.out.printf("Runtime of quickSort(Party)          \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,quickElapsedTime);
//        System.out.println("Sorted array by party using quick sort \n------------------");
//        printArray(largerVotersCopy);

//************************** Runtime for bubbleSort *******************
        votersCopy = arrayClone(voters);
        long bubbleStartTime = System.currentTimeMillis();
        Sort.simpleBubbleSort(votersCopy, idComp);
        long bubbleEndTime = System.currentTimeMillis();
        long bubbleElapsedTime = bubbleEndTime - bubbleStartTime;
        System.out.printf("Runtime of bubbleSort(ID)          \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,bubbleElapsedTime);
//        System.out.println("Sorted array by id using bubble sort \n------------------");
//        printArray(votersCopy);

//************************** Runtime for InsertionSort *****************
         Comparator votedComp = compBag.get(1);
         votersCopy = arrayClone(voters);
         long inserStartTime = System.currentTimeMillis();
         Sort.insertionSort(votersCopy, votedComp);
         long inserEndTime  = System.currentTimeMillis();
         long inserElapsedTime = inserEndTime - inserStartTime;
         System.out.printf("Runtime of insertionSort (Voted)   \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,inserElapsedTime);
//        System.out.println("Sorted array by voted status using insertion sort \n------------------");
//        printArray(votersCopy);

//************************** Runtime for SelectionSort ****************
         Comparator partyComp = compBag.get(0);
         votersCopy = arrayClone(voters);
         long selStartTime = System.currentTimeMillis();
         Sort.selectionSort(votersCopy, partyComp);
         long selEndTime = System.currentTimeMillis();
         long selElapsedTime = selEndTime - selStartTime;
         System.out.printf("Runtime of selectionSort(Party)     \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,selElapsedTime);
//        System.out.println("Sorted array by party using selectionSort\n----------------------");
//        printArray(votersCopy);

//************************* Runtime for radixSort ***********************
         largerVotersCopy = arrayClone(largerVoters);
         long radixStartTime = System.currentTimeMillis();
         Sort.radixSort(largerVotersCopy, compBag);
         long radixEndTime = System.currentTimeMillis();
         long radixElapsedTime = radixEndTime - radixStartTime;
         System.out.printf("Runtime of radix Sort                \t for N\t = %,7d \t time \t= %,10d miliseconds \n\n",limitMax,radixElapsedTime);

//**************** Printing run time of each soring algorithm ***********
         System.out.printf("Runtime of bubbleSort(ID)          \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,bubbleElapsedTime);
         System.out.printf("Runtime of insertionSort (Voted)   \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,inserElapsedTime);
         System.out.printf("Runtime of selectionSort(Party)    \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limit,selElapsedTime);
         System.out.printf("Runtime of quickSort(Party)        \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,quickElapsedTime);
         System.out.printf("Runtime of merge Sort(Name)        \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,mergElapsedTime);
         System.out.printf("Runtime of radix Sort              \t for N\t = %,7d \t time \t= %,10d miliseconds \n",limitMax,radixElapsedTime);


//        System.out.println("Sorted array by party using radixSort\n----------------------");
//        printArray(largerVotersCopy);
//
//        votedComp = new DecisionComparator();
//        Sort.mergeSort(voters,votedComp );
//        System.out.println("Sorted array by decision using merge sort \n------------------");
//        printArray(voters);





}
    public static void printArray( Voter[] data )
    {
         for (Voter legalVoter : data) {
             System.out.println(legalVoter);
         }
         System.out.println("");
    }
         public static Voter[] arrayClone(Voter[] parent)
    {
         int parentSize = parent.length;
         Voter[] clone =  new Voter [parentSize];
         for(int i = 0; i < parentSize; ++i)
         {
             clone[i] = parent[i];
         }
         return clone;
    }
}
```