```
import java.util.Arrays;
final String B1_MSG="Enter length of list";
boolean display = true;
boolean displayRadix = false;
boolean displaySelection = false;
boolean displayMerge = false;
boolean displayShell = false;
boolean displayQuick = false;
boolean displayInsertion = false;
boolean displayBinaryInsertion = false;
boolean displayBubble = false;
int[] values;
int[] states;
int i = 0;
int j = 0;
String msg=B1_MSG;
int numOfcomparisons = 0;
int numOfmoves = 0;
int green = 200;
int background = 205;

void setup() {
  size(1000, 650);
  }
void draw() {
  background(background);


  if(display==true){
```

```
fill(255,255,255);
rect(75,75,150,45);

fill(0);
textAlign(CENTER);
textSize(16);
text(msg, 150, 100);
fill(0);
textSize(20);
text("MAX IS 100 ",150,140);

fill(0,green,0);
rect(230,75,60,45); /////
fill(255);
textSize(18);
text("ENTER", 260,100);

fill(150);
rect(10,75, 60,45);
textSize(18);
fill(255);
text(" CLEAR", 35, 100);

fill(255,255,0);
rect(350,75,150,45);
fill(0);
textAlign(CENTER);
textSize(15);
text("Bubble Sort", 425, 100);
```

```
fill(255,255,0);
rect(565,75,150,45);
fill(0);
textAlign(CENTER);
textSize(15);
text("Shell Sort", 640, 100);

fill(255,255,0);
rect(565,150,150,45);
fill(0);
textAlign(CENTER);
textSize(15);
text("Selection Sort", 640, 175);

fill(255,255,0);
rect(780,75,150,45);
fill(0);
textAlign(CENTER);
textSize(15);
text("Radix Sort", 855, 100);

fill(255,255,0);
rect(350,150,150,45);
fill(0);
textAlign(CENTER);
textSize(15);
text("Insertion Sort", 425, 175);
```

```
  fill(255,255,0);
  rect(350,225,150,45);
  fill(0);
  textAlign(CENTER);
  textSize(15);
  text("Quick Sort", 425, 250);


  fill(255,255,0);
  rect(780,150,150,45);
  fill(0);
  textAlign(CENTER);
  textSize(15);
  text("Merge Sort", 855, 175);


}

if(displayBubble == true){
  fill(0);
  textSize(24);
  text("Comparisons: "+numOfcomparisons, 125, 50);
  text("Moves: "+numOfmoves, 125, 100);
  bubbleSort(values);
}
if(displaySelection == true){
  fill(0);
  textSize(24);
  text("Comparisons: "+numOfcomparisons, 125, 50);
```

```
  text("Moves: "+numOfmoves, 125, 100);
  for (int i = 0; i < values.length; i++) {
    fill(50,100,values[i]-255);
    stroke(0);
    rect(((width/2)-(values.length/2)*10)+i*(10),height-values[i],10,values[i]);
  }
}
if(displayInsertion == true){
  fill(0);
  textSize(24);
  text("Comparisons: "+numOfcomparisons, 125, 50);
  text("Moves: "+numOfmoves, 125, 100);
  for (int i = 0; i < values.length; i++) {
    fill(50,100,values[i]-255);
    stroke(0);
    rect(((width/2)-(values.length/2)*10)+i*(10),height-values[i],10,values[i]);
  }
}
if(displayRadix == true){
  fill(0);
  textSize(24);
  text("Comparisons: "+numOfcomparisons, 125, 50);
  text("Moves: "+numOfmoves, 125, 100);
  for (int i = 0; i < values.length; i++) {
    fill(50,100,values[i]-255);
    stroke(0);
    rect(((width/2)-(values.length/2)*10)+i*(10),height-values[i],10,values[i]);
  }
}
```

```
if(displayBinaryInsertion == true){
  fill(0);
  textSize(24);
  text("Comparisons: "+numOfcomparisons, 125, 50);
  text("Moves: "+numOfmoves, 125, 100);
  for (int i = 0; i < values.length; i++) {
    fill(50,100,values[i]-255);
    stroke(0);
    rect(((width/2)-(values.length/2)*10)+i*(10),height-values[i],10,values[i]);
  }
}
if(displayShell == true){
  fill(0);
  textSize(24);
  text("Comparisons: "+numOfcomparisons, 125, 50);
  text("Moves: "+numOfmoves, 125, 100);
  for (int i = 0; i < values.length; i++) {
    fill(50,100,values[i]-255);
    stroke(0);
    rect(((width/2)-(values.length/2)*10)+i*(10),height-values[i],10,values[i]);
  }
}
if(displayQuick == true){
  fill(0);
  textSize(24);
  text("Comparisons: "+numOfcomparisons, 125, 50);
  text("Moves: "+numOfmoves, 125, 100);
  for (int i = 0; i < values.length; i++) {
    fill(50,100,values[i]-255);
```

```
   stroke(0);
   rect(((width/2)-(values.length/2)*10)+i*(10),height-values[i],10,values[i]);}
  }
  if(displayMerge == true){
   fill(0);
   textSize(24);
   text("Comparisons: "+numOfcomparisons, 125, 50);
   text("Moves: "+numOfmoves, 125, 100);
   for (int i = 0; i < values.length; i++) {
    fill(50,100,values[i]-255);
   stroke(0);
   rect(((width/2)-(values.length/2)*10)+i*(10),height-values[i],10,values[i]);}
  }
}

void mousePressed() {
  if(mouseX > 230 && mouseX < 290 &&  //// //// Enter button
     mouseY > 75 && mouseY < 120){
      green = 100;
  }
  if(mouseX > 350 && mouseX < 500 &&
     mouseY > 75 && mouseY < 120){    ///BubbleSort button
      values=new int[int(msg)];
      for (int i = 0; i < values.length; i++) {
        values[i] = int(random(height));
      }
      display = false;
     displayBubble = true;
     background = 175;
```

```
    }
if(mouseX > 565&& mouseX < 715 &&
   mouseY > 75 && mouseY < 120){///shell button
    Thread t = new Thread(){
   public void run(){
    shellSort(values);
   }
    };
   values=new int[int(msg)];
   //states=new int[int(msg)];
    for (int i = 0; i < values.length; i++) {
      values[i] = int(random(height));
       //states[i] = -1;
    }t.start();
    display = false;
     displayShell = true;
     background = 175;
    }
   if(mouseX > 780&& mouseX < 930 &&
   mouseY > 75 && mouseY < 120){///Radix button
    Thread t = new Thread(){
   public void run(){
    RadixSort(values);
   }
    };
   values=new int[int(msg)];
   //states=new int[int(msg)];
    for (int i = 0; i < values.length; i++) {
      values[i] = int(random(height));
```

```
      //states[i] = -1;
    }t.start();
    display = false;
    displayRadix = true;
    background = 175;
    }
if(mouseX > 565&& mouseX < 715 &&
  mouseY > 150 && mouseY < 195){///Selection button
    Thread t = new Thread(){
  public void run(){
    selectionSort(values);
  }
    };
  values=new int[int(msg)];
  //states=new int[int(msg)];
    for (int i = 0; i < values.length; i++) {
      values[i] = int(random(height));
      //states[i] = -1;
    }t.start();
    display = false;
     displaySelection = true;
     background = 175;
    }
  if(mouseX > 780&& mouseX < 930 &&
  mouseY > 150 && mouseY < 195){///mergeSort button
    Thread t = new Thread(){
  public void run(){
    mergeSort(values, 0, values.length-1);
  }
```

```
    };
   values=new int[int(msg)];
   //states=new int[int(msg)];
    for (int i = 0; i < values.length; i++) {
     values[i] = int(random(height));
     //states[i] = -1;
    }t.start();
    display = false;
     displayMerge = true;
     background = 175;
    }

if(mouseX > 350 && mouseX < 500 &&
   mouseY > 230 && mouseY < 295){    ///quickSort button
    Thread t = new Thread(){
   public void run(){
    quickSort(values,0,values.length-1);
   }
    };
   values=new int[int(msg)];
   states=new int[int(msg)];
    for (int i = 0; i < values.length; i++) {
     values[i] = int(random(height));
     states[i] = -1;
    }t.start();
    display = false;
     displayQuick = true;
     background = 175;
    }
```

```
  if(mouseX > 350 && mouseX < 500 &&
     mouseY > 150 && mouseY < 195){    ///Insertion button
       Thread t = new Thread(){
     public void run(){
       insertionSort(values);
     }
       };
     values=new int[int(msg)];
     //states=new int[int(msg)];
       for (int i = 0; i < values.length; i++) {
         values[i] = int(random(height));
         //states[i] = -1;
       }t.start();
       display = false;
        displayInsertion = true;
        background = 175;
       }
  if(mouseX > 10 && mouseX < 70 &&  /// Clear button
     mouseY > 75 && mouseY < 120){
  msg=B1_MSG;
  green = 200;
  }
}

void keyPressed() {

  //Prepare when writing a new message. Next resets message container
  if (msg.equals(B1_MSG)) {
    msg="";
```

```
  }

//Detects only alphanumeric chars
  if (key>='0' && key<='9')
    {
    msg+=key;
    //println(msg);
   }
  }
  void swap(int[] arr, int a, int b) {
  int temp = arr[a];
  arr[a] = arr[b];
  arr[b] = temp;



  }
void binaryInsertionSort(int [] arr){
  for (int i = 1; i < arr.length; i++){
    try{Thread.sleep(175);}catch(Exception e){}; //delay the thread operating this function
    redraw();
    int x = arr[i];
    int j = Math.abs(Arrays.binarySearch(arr, 0,i, x) + 1);
    System.arraycopy(arr, j,arr, j + 1, i - j);
    arr[j] = x;
  }
}
void selectionSort(int arr[]){
  int n = arr.length;
   for (int i = 0; i < n-1; i++)
```

```
  {
    try{Thread.sleep(175);}catch(Exception e){}; //delay the thread operating this function
    redraw();
    int min_idx = i;
    for (int j = i+1; j < n; j++)
    if (arr[j] < arr[min_idx])
    {
    min_idx = j;
    numOfcomparisons++;
    }
    int temp = arr[min_idx];
    arr[min_idx] = arr[i];
    numOfmoves++;
    arr[i] = temp;
    numOfmoves++;
   }
   }
void insertionSort(int[] arr){
  int i;
  int key;
  for (int j = 1; j < arr.length; j++) {
    try{Thread.sleep(175);}catch(Exception e){}; //delay the thread operating this function
    redraw();
    key = arr[ j ];
    i = j - 1;
    while ((i >= 0) && (arr[ i ] > key)) {
      numOfcomparisons++;
      arr[ i + 1 ] = arr[ i ];
      i--;
```

```
      }
      numOfmoves++;
      arr[ i + 1 ] = key;
    }
}
void shellSort(int arr[])
   {
       int n = arr.length;
       for (int gap = n/2; gap > 0; gap /= 2)
       {
          try{Thread.sleep(30);}catch(Exception e){}; //delay the thread operating this function
          redraw();
          for (int i = gap; i < n; i += 1)
          {try{Thread.sleep(30);}catch(Exception e){}; //delay the thread operating this function
          redraw();
             int temp = arr[i];
             int j;
             for (j = i; j >= gap && arr[j - gap] > temp; j -= gap){
                arr[j] = arr[j - gap];
                numOfmoves++;
             }
             arr[j] = temp;
             numOfcomparisons++;
          }
       }
   }
void bubbleSort(int[] arr){
  if (i < arr.length) {
     for (int j = 0; j < arr.length-i-1; j++) {
```

```
      try{Thread.sleep(2);}catch(Exception e){}; //delay the thread operating this function
      redraw();
      float a = arr[j];
      float b = arr[j+1];
      if (a > b) {
        numOfmoves++;
        swap(arr, j, j+1);
      }
      numOfcomparisons++;
    }
  }else {
    println("finished");
    noLoop();
  }
  i++;
  for (int i = 0; i < arr.length; i++) {
    stroke(0);
    fill(50,100,values[i]-255);
    rect(((width/2)-(arr.length/2)*10)+i*(10),height-arr[i],10,arr[i]);}
}
void quickSort(int []arr,int start,int end) {
  if (start >= end) {
    try{Thread.sleep(50);}catch(Exception e){}; //delay the thread operating this function
    redraw();
    return;
  }
  int index = partition(arr, start, end);
  states[index] = -1;
```

```
  quickSort(arr, start, index - 1);
  quickSort(arr, index + 1, end);
}
public int partition(int []arr,int start,int end) {
  for (int i = start; i < end; i++) {
    try{Thread.sleep(10);}catch(Exception e){}; //delay the thread operating this function
    redraw();
    states[i] = 1;
  }

  int pivotValue = arr[end];
  int pivotIndex = start;
  states[pivotIndex] = 0;
  for (int i = start; i < end; i++) {
    try{Thread.sleep(5);}catch(Exception e){}; //delay the thread operating this function
    redraw();
    if (arr[i] < pivotValue) {
      numOfmoves++;
      swap(arr, i, pivotIndex);
      numOfcomparisons++;
      states[pivotIndex] = -1;
      pivotIndex++;
      states[pivotIndex] = 0;
    }
  }
  swap(arr, pivotIndex, end);
  numOfmoves++;

  for (int i = start; i < end; i++) {
```

```
    if (i != pivotIndex) {
      states[i] = -1;
    }
  }

  return pivotIndex;
}
void merge(int arr[], int l, int m, int r) {
        int sizeOfFirstSubArray = m - l + 1;
        int sizeOfSecondSubArray = r - m;

        //Creating two temp arrays
        int leftArr[] = new int[sizeOfFirstSubArray];
        int rightArr[] = new int[sizeOfSecondSubArray];

        //Copying data to these two temp arrays
        int i = 0, j = 0;//starting indices for two temp arrays
        for (i = 0; i < sizeOfFirstSubArray; i++) {try{Thread.sleep(15);}catch(Exception e){}; //delay the thread operating this function
        redraw();
           leftArr[i] = arr[l + i];
        }
        for (j = 0; j < sizeOfSecondSubArray; j++) {try{Thread.sleep(15);}catch(Exception e){}; //delay the thread operating this function
        redraw();
           rightArr[j] = arr[m + j + 1];
        }

        i = 0;
```

```
    j = 0;//re-setting starting indices of temp arrays
    int k = l;//starting index of merged sub-array (arr[])

    //Merging these two temp arrays
    while (i < sizeOfFirstSubArray && j < sizeOfSecondSubArray) {try{Thread.sleep(10);}catch(Exception
e){}; //delay the thread operating this function
    redraw();
       if (leftArr[i] <= rightArr[j]) {
         numOfmoves++;
          arr[k] = leftArr[i];
          numOfcomparisons++;
          i++;
       } else {
          arr[k] = rightArr[j];
          numOfcomparisons++;
          j++;
       }
       k++;
    }
    //copying remaining elements from leftArr
    for (; i < sizeOfFirstSubArray; i++) {
       arr[k++] = leftArr[i];
    }
    //copying remaining elements from rightArr
    for (; j < sizeOfSecondSubArray; j++) {
       arr[k++] = rightArr[j];
    }

    //printArray(arr, l, r);
```

```
    }

void mergeSort(int arr[], int l, int r) {
        if (l < r) {
            int m = (l + r) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }
int RadixGetMaxLength(int [] array, int arraySize)
{
  int maxDigits = 0;
  for (int i = 0; i < arraySize; i++)
  {
     int digitCount = RadixGetLength(array[i]);
     if (digitCount > maxDigits)
       maxDigits = digitCount;
  }
  return maxDigits;
}
void RadixSort(int [] array)
{
  int arraySize = array.length;

  ArrayList<Integer> [] buckets = new ArrayList[10];
  for (int i=0; i < buckets.length; i++)
  {
     buckets[i] = new  ArrayList();
```

```
  }

  int maxDigits = RadixGetMaxLength(array, arraySize);

  // Start with the least significant digit
  int pow10 = 1;
  for (int digitIndex = 0; digitIndex < maxDigits; digitIndex++)
  {
    for (int i = 0; i < arraySize; i++)
    {
      int bucketIndex = Math.abs(array[i] / pow10) % 10;
      buckets[bucketIndex].add(array[i]);
      numOfmoves++;
    }
    int arrayIndex = 0;
    for (int i = 0; i < 10; i++) {
     numOfmoves++;
      for (int j = 0; j < buckets[i].size(); j++)
        array[arrayIndex++] = buckets[i].get(j);
        try{Thread.sleep(65);}catch(Exception e){}; //delay the thread operating this function
   redraw();
    }
    pow10 = 10 * pow10;
    for (int i = 0; i < 10; i++)
       buckets[i].clear();
  }
}
// Returns the length, in number of digits, of value
int RadixGetLength(int value)
```

```
{
  if (value == 0)
    return 1;
  int digits = 0;
  while (value != 0)
  {
    digits = digits + 1;
    value = value / 10;
  }
  return digits;
}
```