```java
package COSC2P03_A3;

public class DataItem
{
    // Instance Variables
    public String smiles, fragments;
    public int n_fragments, C, F, N, O, Other, SINGLE, DOUBLE, TRIPLE, Tri, Quad, Pent, Hex;
    public float logP, mr, qed, SAS;
    public int sumRank;

    // Constructor
    public DataItem()
    {

    }
}

```

```java
package COSC2P03_A3;
// Imports
import java.io.*;

public class DrugDesign
{
    // Instance Variables
    String path = "C:\\Users\\Admin\\Desktop\\COSC 2P03\\ASSIGNMENTS\\ASSIGNMENT#3\\COSC2P03_Assignment3\\ZINCSubet.csv";
    // put in while loop   // created an instance of the DataItem class
    public DataItem [] originalData = new DataItem [12000];
    public DataItem [] finalSortedData = new DataItem [12000];

    public float [] HeapLogp = new float [12000];
    public int [] IndexArr = new int [HeapLogp.length];

    public float [] mr = new float [12000];
    public int [] mrIndexArr = new int [mr.length];

    public float [] SAS = new float [12000];
    public float [] tempSAS = new float[SAS.length];
    public float [] SASArr = new float [SAS.length];
    public int [] SASIndexArr = new int [SAS.length];



    // Constructor
    public DrugDesign() throws IOException
    {
        loadData(path);

        // Heap Sort
        for(int i = 0; i < IndexArr.length; i++)
        {
            IndexArr[i] = i;
        }
        heapSort(HeapLogp);
        System.out.println("----------------------------------------------------------Heap Sort: logp ----------------------------------------------");
        printArray(IndexArr);

        // Quick Sort
        for(int i = 0; i < mrIndexArr.length; i++)
        {
            mrIndexArr[i] = i;
        }
        for(int i = 0; i < mr.length; i++)
        {
            mr[i] = originalData[i].mr;
        }
        QuickSort(mr, 0, mr.length - 1);
        System.out.println("----------------------------------------------------------Quick Sort: mr ----------------------------------------------");
        printArray(mrIndexArr);

        // Merge Sort
        for(int i = 0; i < SAS.length; i++)
        {
            SAS[i] = originalData[i].SAS;
            SASArr[i] = SAS[i];
        }
        Mergesort(SAS, tempSAS, 0, SAS.length - 1);
        IndexArr();
        System.out.println("----------------------------------------------------------Merge Sort: SAS ----------------------------------------------");
        printArray(SASIndexArr);

```

```
64          // SumRankSort
65          int[] sortedIndexFinal = sumRankSort(IndexArr, mrIndexArr, SASIndexArr);
66          System.out.println("----------------------------------------------------SumRankSort Sort
    :-------------------------------------------------");
67          printArray(sortedIndexFinal);
68
69          SaveNewData(sortedIndexFinal);
70      }
71
72      // Methods
73      public void loadData(String path) throws IOException
74      {
75          String line1 = "";
76          int i = 0;
77          try
78          {
79              BufferedReader br = new BufferedReader(new FileReader(path));
80              System.out.println(br.readLine());
81              while((line1 = br.readLine()) != null)
82              {
83                  DataItem item = new DataItem();
84                  String[] ar = line1.split(",");
85
86                  item.smiles = ar[0];
87                  //System.out.print(item.smiles);
88                  item.fragments = ar[1];
89                  //System.out.print(", " + item.fragments);
90                  item.n_fragments = Integer.parseInt(ar[2]);
91                  //System.out.print(", " + item.n_fragments);
92                  item.C = Integer.parseInt(ar[3]);
93                  //System.out.print(", " + item.C);
94                  item.F = Integer.parseInt(ar[4]);
95                  //System.out.print(", " + item.F);
96                  item.N = Integer.parseInt(ar[5]);
97                  //System.out.print(", " + item.N);
98                  item.O = Integer.parseInt(ar[6]);
99                  //System.out.print(", " + item.O);
100                 item.Other = Integer.parseInt(ar[7]);
101                 //System.out.print(", " + item.Other);
102                 item.SINGLE = Integer.parseInt(ar[8]);
103                 //System.out.print(", " + item.SINGLE);
104                 item.DOUBLE = Integer.parseInt(ar[9]);
105                 //System.out.print(", " + item.DOUBLE);
106                 item.TRIPLE = Integer.parseInt(ar[10]);
107                 //System.out.print(", " + item.TRIPLE);
108                 item.Tri = Integer.parseInt(ar[11]);
109                // System.out.print(", " + item.Tri);
110                 item.Quad = Integer.parseInt(ar[12]);
111                 //System.out.print(", " + item.Quad);
112                 item.Pent = Integer.parseInt(ar[13]);
113                 //System.out.print(", " + item.Pent);
114                 item.Hex = Integer.parseInt(ar[14]);
115                 //System.out.print(", " + item.Hex);
116                 item.logP = Float.parseFloat(ar[15]);
117                 //System.out.print(", " + item.logP);
118                 item.mr = Float.parseFloat(ar[16]);
119                 //System.out.print(", " + item.mr);
120                 item.qed = Float.parseFloat(ar[17]);
121                 //System.out.print(", " + item.qed);
122                 item.SAS = Float.parseFloat(ar[18]);
123                 //System.out.print(", " + item.SAS);
124
125                 originalData[i] = item;
126                 //System.out.println(originalData[i].smiles + " , " + originalData[i].fragments
     + " , " + originalData[i].n_fragments + " , " + originalData[i].C + " , " + originalData[i].F
     + " , " + originalData[i].N + " , " + originalData[i].O + " , " + originalData[i].Other + " , " +
    originalData[i].SINGLE + " , " + originalData[i].DOUBLE + " , " + originalData[i].TRIPLE + " , " +
```

```
126 originalData[i].Tri + " , " + originalData[i].Quad + " , " + originalData[i].Pent + " , " +
    originalData[i].Hex + " , " + originalData[i].logP + " , " + originalData[i].mr + " , " + originalData
    [i].qed + " , " + originalData[i].SAS);
127                 i++;
128 
129                 //break;
130             }
131         }
132         catch (FileNotFoundException e)
133         {
134             e.printStackTrace();
135         }
136     }
137 
138     public int [] heapSort(float [] HeapLogp)
139     {
140         for(int i = 0; i < HeapLogp.length; i++)
141         {
142             HeapLogp[i] = originalData[i].logP;
143             //System.out.println(HeapLogp[i]);
144             //System.out.println(IndexArr[i]);
145         }
146 
147         // Build heap
148         for (int i = HeapLogp.length; i >= 0; i--)
149         {
150             heapify(HeapLogp, HeapLogp.length, i);
151         }
152 
153         // seperate the largest element by moving it to the end of the array
154         for (int i = HeapLogp.length - 1; i > 0; i--)
155         {
156             // Move current root to end
157             float temp = HeapLogp[0];
158             HeapLogp[0] = HeapLogp[i];
159             HeapLogp[i] = temp;
160 
161             int temp2 = IndexArr[0];
162             IndexArr[0] = IndexArr[i];
163             IndexArr[i] = temp2;
164 
165             // call the max heap on the reduced heap
166             heapify(HeapLogp, i, 0);
167         }
168         return IndexArr;
169     }
170 
171     // Heapifies the array by recursively labelling which of the 3: root, left, right - is the largest
172     // and then swapping the largest with the root
173     public void heapify(float HeapLogp[], int n, int i)
174     {
175         int largest = i; // root is initially labelled as the largest, unless the children are larger
176         int l = 2 * i + 1; // left child
177         int r = 2 * i + 2; // right chile
178         // If the left child is larger than the root
179         if (l < n && HeapLogp[l] > HeapLogp[largest])
180             largest = l;
181 
182         // If the right child is larger than largest so far : could be the root or left child
183         if (r < n && HeapLogp[r] > HeapLogp[largest])
184             largest = r;
185 
186         // If the largest is not the root anymore
187         if (largest != i)
188         {
189             float swap = HeapLogp[i];
190             HeapLogp[i] = HeapLogp[largest];
```

```
191             HeapLogp[largest] = swap;
192 
193             int swap2 = IndexArr[i];
194             IndexArr[i] = IndexArr[largest];
195             IndexArr[largest] = swap2;
196 
197             // Recursively heapify the array
198             heapify(HeapLogp, n, largest);
199         }
200     }
201 
202     /* The main function that implements QuickSort
203        float arr[] - is the Array to be sorted
204        low - the first index (0)
205        high - the last index (arr.length - 1)
206   */
207     public void QuickSort(float[] arr, int low, int high)
208     {
209         if (low < high)
210         {
211             // pivot is partitioning the left and right side apart
212             int pivot = partition(arr, low, high);
213 
214             // Separately sort elements on left and right side of the pivot
215             QuickSort(arr, low, pivot - 1);
216             QuickSort(arr, pivot + 1, high);
217         }
218     }
219 
220     /*
221     * Takes the last element as a pivot value
222     * Then places it in its sorted position
223     * partitions the left - smaller than the pivot and right - larger than the pivot*/
224     public int partition(float[] arr, int low, int high)
225     {
226         // pivot
227         float pivot = arr[high];
228         int i = (low - 1);
229 
230         for(int j = low; j <= high - 1; j++)
231         {
232             // If current element is smaller
233             // than the pivot
234             if (arr[j] < pivot)
235             {
236                 // Increment index of
237                 // smaller element
238                 i++;
239                 swap(arr, i, j);
240             }
241         }
242         swap(arr, i + 1, high);
243         return (i + 1);
244     }
245 
246     // swaps two elements in a float array
247     public void swap(float[] arr, int i, int j)
248     {
249         float temp = arr[i];
250         arr[i] = arr[j];
251         arr[j] = temp;
252 
253         // swaps the indicies as well
254         int temp2 = mrIndexArr[i];
255         mrIndexArr[i] = mrIndexArr[j];
256         mrIndexArr[j] = temp2;
257     }
```

```
258
259
260     public void Merge(float [] A, float [] tmp , int leftPos , int rightPos , int rightEnd)
261     {
262         int leftEnd = rightPos - 1;
263         int tmpPos = leftPos;
264         int numElements = rightEnd - leftPos + 1;
265         while(leftPos <= leftEnd && rightPos <= rightEnd)
266         {
267             if(A[leftPos] <= A[rightPos])
268             {
269                 //tmp[tmpPos] = SASIndexArr[leftPos];
270                 tmp[tmpPos] = A[leftPos];
271                 tmpPos++;
272                 leftPos++;
273             }
274             else
275             {
276                 //tmp[tmpPos] = SASIndexArr[rightPos];
277                 tmp[tmpPos] = A[rightPos];
278                 tmpPos++;
279                 rightPos++;
280             }
281         }
282         while(leftPos <= leftEnd)
283         {
284             //tmp[tmpPos] =  SASIndexArr[leftPos];
285             tmp[tmpPos] = A[leftPos];
286             tmpPos++;
287             leftPos++;
288         }
289         while(rightPos <= rightEnd)
290         {
291             //tmp[tmpPos] =  SASIndexArr[rightPos];
292             tmp[tmpPos] = A[rightPos];
293             tmpPos++;
294             rightPos++;
295         }
296         for(int i = 0; i < numElements ; i++, rightEnd--)
297         {
298             A[rightEnd] = tmp[rightEnd];
299             //SASIndexArr[rightEnd] = (int) tmp[rightEnd];
300             //System.out.println(tmp[rightEnd]);
301         }
302     }
303
304     public void Mergesort(float [] A, float [] tmp, int lower, int upper)
305     {
306         if(lower < upper)
307         {
308             int mid = (lower+upper)/2; //int division
309             Mergesort(A, tmp, lower, mid);
310             Mergesort(A, tmp, mid+1, upper);
311             Merge(A, tmp, lower, mid+1, upper);
312         }
313     }
314
315     /*
316     Takes the 3 1-d sorted index arrays and adds all the indicies together into each index of another
    array
317     sorts the new array using heap sort
318     returns as an int [] array     */
319     public int [] sumRankSort(int [] logpArr, int [] mrArr, int [] SASArr)
320     {
321         float [] sumrank = new float[logpArr.length];
322         for(int i = 0; i < logpArr.length; i++)
323         {
```

```
324             sumrank[i] = logpArr[i] + mrArr[i] + SASArr[i];
325             //System.out.println(sumrank[i]);
326         }
327         int [] sumranksort = heapSort(sumrank);
328         return sumranksort;
329     }
330 
331     /* Prints an int [] array - used to print out the index arrays */
332     public void printArray(int arr[])
333     {
334         for (int i = 0; i < arr.length; ++i)
335         {
336             System.out.print(arr[i] + " , ");
337         }
338         System.out.println();
339         System.out.println();
340     }
341 
342     /*
343     Used for the merge sort algorithm to effectively sort the array based on indicies
344     takes the original un-sorted array
345     compares value stored in 1st index to each index of the sorted array
346     when values match, then inserts the index value into the sorted array
347      */
348     public void IndexArr()
349     {
350         for(int i = 0; i < SASArr.length; i++)
351         {
352             for(int j = 0; j < SAS.length; j++)
353             {
354                 if(SASArr[i] == SAS[j])
355                 {
356                     SASIndexArr[j] = i;
357                 }
358             }
359         }
360     }
361 
362     /*
363     Uses PrintWriter to create a CSV file of the object "finalSortedData"
364      */
365     public void SaveNewData(int [] sortedIndexFinal)
366     {
367         try {
368             PrintWriter pw = new PrintWriter(new File("C:\\Users\\Admin\\Desktop\\COSC 2P03\\
    ASSIGNMENTS\\ASSIGNMENT#3\\COSC2P03_Assignment3\\ZINCSubet_sorted.csv"));
369             StringBuilder sb = new StringBuilder();
370             for (int i = 0; i < sortedIndexFinal.length; i++)
371             {
372                 DataItem item2 = new DataItem();
373                 item2.sumRank = sortedIndexFinal[i];
374                 finalSortedData[i] = item2;
375                 sb.append(finalSortedData[i].sumRank);
376                 sb.append("\n");
377             }
378             pw.write(sb.toString());
379             pw.close();
380         }
381         catch (Exception e)
382         {
383             e.printStackTrace();
384         }
385     }
386 
387 
388     public static void main(String[] args) throws IOException
389     {
```

```
390         DrugDesign d = new DrugDesign();
391     }
392 }
393
```