1 // TV Show Object  
 2   
 3 public class TVShows  
 4 {  
 5   
 6 // TVShow Properties   
 7   
 8 String name;  
 9 int yearPremired;  
10 int numberOfSeasons;  
11 int numberOfEpisodes;  
12 String network;  
13 String genre;  
14 String maleLead;  
15 String femaleLead;  
16   
17 // TV Show Constructor  
18   
19 TVShows(String name, int yearPremired, int numberOfSeasons, int numberOfEpisodes, String network, String genre, String maleLead, String femaleLead)  
20 {  
21 this.name=name;  
22 this.yearPremired=yearPremired;  
23 this.numberOfSeasons=numberOfSeasons;  
24 this.numberOfEpisodes=numberOfEpisodes;  
25 this.network=network;  
26 this.genre=genre;  
27 this.maleLead=maleLead;  
28 this.femaleLead=femaleLead;  
29 }  
30 }

1 import java.io.BufferedReader;   
 2 import java.io.FileReader;   
 3 import java.io.IOException;  
 4 import java.io.\*;   
 5 import java.util.\*;   
 6   
 7 public class TVShowTester   
 8 {   
 9   
 10 // Use an insertion sort, completed correctly, in ascending order by TV show name  
 11   
 12 static void insertionSortAscending(TVShows array[])  
 13 {  
 14 int n = array.length;   
 15 for (int j = 1; j < n; j++) {   
 16 TVShows key = array[j];   
 17 int i = j-1;   
 18 while ( (i > -1) && ( array[i].name.compareTo(key.name)>0) ) {   
 19 array [i+1] = array [i];   
 20 i--;   
 21 }  
 22 array[i+1] = key;  
 23 }  
 24   
 25 for(int i=0;i<n;i++)print(array[i]);  
 26 }  
 27   
 28 // Use an insertion sort, completed correctly, in descending order by number of seasons  
 29   
 30 static void insertionSortDescending(TVShows array[])  
 31 {  
 32 int n = array.length;   
 33 for (int j = 1; j < n; j++) {   
 34 TVShows key = array[j];   
 35 int i = j-1;   
 36 while ( (i > -1) && ( array[i].name.compareTo(key.name)<0) ) {   
 37 array [i+1] = array [i];   
 38 i--;   
 39 }  
 40 array[i+1] = key;  
 41 }   
 42 for(int i=0;i<n;i++)print(array[i]);  
 43 }  
 44   
 45 // Use a selection sort, completed correctly, in ascending order by year the show premiered  
 46   
 47 static void selectionSortAscending(TVShows arr[])  
 48 {  
 49 for (int i = 0; i < arr.length - 1; i++)   
 50 {   
 51 int index = i;   
 52 for (int j = i + 1; j < arr.length; j++){   
 53 if (arr[j].name.compareTo(arr[index].name) < 0){   
 54 index = j;   
 55 }   
 56 }   
 57 TVShows smallerNumber = arr[index];   
 58 arr[index] = arr[i];   
 59 arr[i] = smallerNumber;   
 60 }   
 61 for(int i=0;i<arr.length;i++)print(arr[i]);  
 62 }  
 63   
 64 // Use a selection sort, completed correctly, in descending order by genre  
 65   
 66 static void selectionSortDescending(TVShows arr[])  
 67 {  
 68 for (int i = 0; i < arr.length - 1; i++)   
 69 {   
 70 int index = i;   
 71 for (int j = i + 1; j < arr.length; j++){   
 72 if (arr[j].name.compareTo(arr[index].name) > 0){   
 73 index = j;   
 74 }   
 75 }   
 76 TVShows smallerNumber = arr[index];   
 77 arr[index] = arr[i];   
 78 arr[i] = smallerNumber;   
 79 }   
 80 for(int i=0;i<arr.length;i++)print(arr[i]);  
 81 }  
 82   
 83 // Use a merge sort, completed correctly, in ascending order by number of episodes  
 84   
 85 static void mergeAsscending(TVShows arr[], int l, int m, int r)  
 86 {  
 87 int n1 = m - l + 1;  
 88 int n2 = r - m;  
 89   
 90 TVShows L[] = new TVShows[n1];  
 91 TVShows R[] = new TVShows[n2];  
 92   
 93 for (int i = 0; i < n1; ++i)  
 94 L[i] = arr[l + i];  
 95 for (int j = 0; j < n2; ++j)  
 96 R[j] = arr[m + 1 + j];  
 97   
 98 int i = 0, j = 0;  
 99   
100 int k = l;  
101 while (i < n1 && j < n2) {  
102 if (L[i].numberOfEpisodes <= R[j].numberOfEpisodes) {  
103 arr[k] = L[i];  
104 i++;  
105 }  
106 else {  
107 arr[k] = R[j];  
108 j++;  
109 }  
110 k++;  
111 }  
112   
113 while (i < n1) {  
114 arr[k] = L[i];  
115 i++;  
116 k++;  
117 }  
118   
119 while (j < n2) {  
120 arr[k] = R[j];  
121 j++;  
122 k++;  
123 }  
124 }  
125   
126   
127 // Recursive merge  
128   
129 static void mergeSortAscending(TVShows arr[], int l, int r)  
130 {  
131 if (l < r) {  
132 int m =l+ (r-l)/2;  
133 mergeSortAscending(arr, l, m);  
134 mergeSortAscending(arr, m + 1, r);  
135 mergeAsscending(arr, l, m, r);  
136 }  
137 for(int i=0;i<arr.length;i++)print(arr[i]);  
138 }  
139   
140 // Use a merge sort, completed correctly, in descending order by network  
141   
142   
143 static void mergeDescending(TVShows arr[], int l, int m, int r)  
144 {  
145 int n1 = m - l + 1;  
146 int n2 = r - m;  
147   
148 TVShows L[] = new TVShows[n1];  
149 TVShows R[] = new TVShows[n2];  
150   
151 for (int i = 0; i < n1; ++i)  
152 L[i] = arr[l + i];  
153 for (int j = 0; j < n2; ++j)  
154 R[j] = arr[m + 1 + j];  
155   
156 int i = 0, j = 0;  
157   
158 int k = l;  
159 while (i < n1 && j < n2) {  
160 if (L[i].numberOfEpisodes >= R[j].numberOfEpisodes) {  
161 arr[k] = L[i];  
162 i++;  
163 }  
164 else {  
165 arr[k] = R[j];  
166 j++;  
167 }  
168 k++;  
169 }  
170   
171 while (i < n1) {  
172 arr[k] = L[i];  
173 i++;  
174 k++;  
175 }  
176   
177 while (j < n2) {  
178 arr[k] = R[j];  
179 j++;  
180 k++;  
181 }  
182   
183 }  
184   
185 static void mergeSortDescending(TVShows arr[], int l, int r)  
186 {  
187 if (l < r) {  
188 int m =l+ (r-l)/2;  
189 mergeSortDescending(arr, l, m);  
190 mergeSortDescending(arr, m + 1, r);  
191 mergeDescending(arr, l, m, r);  
192 }  
193 for(int i=0;i<arr.length;i++)print(arr[i]);  
194 }  
195   
196 static void print(TVShows te)  
197 {  
198 System.out.println(te.name);  
199 System.out.println(te.yearPremired);  
200 System.out.println(te.numberOfSeasons);  
201 System.out.println(te.numberOfEpisodes);  
202 System.out.println(te.network);  
203 System.out.println(te.genre);  
204 System.out.println(te.maleLead);  
205 System.out.println(te.femaleLead);  
206 }  
207   
208 // Use a sequential search, completed correctly, to find all shows in the array that lasted for 2 years  
209   
210 static void sequentialSearchYearPremired(TVShows arr[])  
211 {  
212 int n=arr.length;  
213   
214 for(int i=0;i<n;i++)  
215 {  
216 if(arr[i].yearPremired>=2)  
217 {  
218 print(arr[i]);  
219 }  
220 }  
221 }  
222   
223 // Use a sequential search, completed correctly to find all shows in the array that had more than 100 episodes  
224   
225 static void sequentialSearchNumberOfEpisodes(TVShows arr[])  
226 {  
227 int n=arr.length;  
228   
229 for(int i=0;i<n;i++)  
230 {  
231 if(arr[i].numberOfEpisodes>=100)  
232 {  
233 print(arr[i]);  
234 }  
235 }  
236 }  
237   
238 // Use a binary search, completed correctly, to find and count all shows in the array that aired on CBS.  
239   
240 static void binarySearchNetwork(TVShows arr[])  
241 {  
242 int n=arr.length;  
243   
244 int l=0, r=n-1;  
245   
246 int count=0;  
247 int end=0, start=0;  
248   
249 while(l<=r)  
250 {  
251 int m=(l+r)/2;  
252   
253 if(arr[m].network.compareTo("CBS")>=0)  
254 {  
255 start=m;  
256 r=m-1;  
257 }  
258 else  
259 {  
260 end=m;  
261 l=m+1;  
262 }  
263 }  
264   
265 while(l<=r)  
266 {  
267 int m=(l+r)/2;  
268   
269 if(arr[m].network.compareTo("CBS")>0)  
270 {  
271 r=m-1;  
272 }  
273 else  
274 {  
275 end=m;  
276 l=m+1;  
277 }  
278 }  
279   
280 count=end-start+1;  
281   
282 System.out.println(count);  
283 }  
284   
285 // Use a binary search, completed correctly, to find and count all shows in the array in the fantasy category  
286   
287 static void binarySearchCategoryFantasy(TVShows arr[])  
288 {  
289 int n=arr.length;  
290   
291 int l=0, r=n-1;  
292   
293 int count=0;  
294 int end=0, start=0;  
295   
296 while(l<=r)  
297 {  
298 int m=(l+r)/2;  
299   
300 if(arr[m].genre.compareTo("CBS")>=0)  
301 {  
302 start=m;  
303 r=m-1;  
304 }  
305 else  
306 {  
307 end=m;  
308 l=m+1;  
309 }  
310 }  
311   
312 while(l<=r)  
313 {  
314 int m=(l+r)/2;  
315   
316 if(arr[m].genre.compareTo("fantasy")>0)  
317 {  
318 r=m-1;  
319 }  
320 else  
321 {  
322 end=m;  
323 l=m+1;  
324 }  
325 }  
326   
327 count=end-start+1;  
328   
329 System.out.println(count);  
330 }  
331   
332 // Use a binary search, completed correctly, to find and count all shows in the array in the drama category  
333   
334 static void binarySearchCategoryDrama(TVShows arr[])  
335 {  
336 int n=arr.length;  
337   
338 int l=0, r=n-1;  
339   
340 int count=0;  
341 int end=0, start=0;  
342   
343 while(l<=r)  
344 {  
345 int m=(l+r)/2;  
346   
347 if(arr[m].genre.compareTo("drama")>=0)  
348 {  
349 start=m;  
350 r=m-1;  
351 }  
352 else  
353 {  
354 end=m;  
355 l=m+1;  
356 }  
357 }  
358   
359 while(l<=r)  
360 {  
361 int m=(l+r)/2;  
362   
363 if(arr[m].genre.compareTo("drama")>0)  
364 {  
365 r=m-1;  
366 }  
367 else  
368 {  
369 end=m;  
370 l=m+1;  
371 }  
372 }  
373   
374 count=end-start+1;  
375   
376 System.out.println(count);  
377 }  
378   
379 // Test Method  
380   
381 public static void main(String[] args)   
382 {   
383 TVShows tv\_shows1[]=new TVShows[100];  
384   
385 String line = "";   
386 String splitBy = ",";   
387 int cnt=0;  
388 try   
389 {   
390 BufferedReader br = new BufferedReader(new FileReader("tv\_shows.csv"));   
391 br.readLine();   
392 while ((line = br.readLine()) != null)  
393 {   
394 String[] data = line.split(splitBy);// use comma as separator  
395   
396 // System.out.println(line);  
397   
398 tv\_shows1[cnt++]=new TVShows(data[0],Integer.parseInt(data[1]),Integer.parseInt(data[2]),Integer.parseInt(data[3]),data[4],data[5],data[6],data[7]);  
399 }   
400 }   
401 catch (IOException e)   
402 {   
403 e.printStackTrace();   
404 }  
405   
406 TVShows tv\_shows[]=new TVShows[cnt];  
407   
408 for(int i=0;i<cnt;i++)tv\_shows[i]=tv\_shows1[i];  
409   
410 System.out.println("Insert Sort Result 1");  
411 insertionSortAscending(tv\_shows);  
412 System.out.println("Insert Sort Result 2");  
413 insertionSortDescending(tv\_shows);  
414 System.out.println("Selection Sort Result 1");  
415 selectionSortAscending(tv\_shows);  
416 System.out.println("Selection Sort Result 2");  
417 selectionSortDescending(tv\_shows);  
418 System.out.println("Merge Sort Results 1");  
419 mergeSortAscending(tv\_shows,0,tv\_shows.length-1);  
420 System.out.println("Merge Sort Results 2");  
421 mergeSortDescending(tv\_shows,0,tv\_shows.length-1);  
422 System.out.println("Sequential Search Results 1");  
423 sequentialSearchYearPremired(tv\_shows);  
424 System.out.println("Sequential Search Results 2");  
425 sequentialSearchNumberOfEpisodes(tv\_shows);  
426 System.out.println("Binary Search Results 1");  
427 binarySearchNetwork(tv\_shows);  
428 System.out.println("Binary Search Results 2");  
429 binarySearchCategoryFantasy(tv\_shows);  
430 System.out.println("Binary Search Result 3");  
431 binarySearchCategoryDrama(tv\_shows);  
432 }   
433 }

The merge sort algorithm divides an array into two equal halves then the arrays are combined together and sorted at the very end. Recursively, the algorithm keeps on dividing the array in half until it is no longer possible to do so. For example, if the array is empty or has a single element remaining, the division will stop. The base case is what stops the recursion. The merge sort is invoked when the arrays are merge combining them into make a larger array. Essentially the array is reassembled.

The insertion algorithm takes the first array element and gives the programmer the option to set it aside or place it in a new array. The subsequent element in the array is inserted to either the left OR the right of the original first array element. It also takes the following elements and looks for a value different and inserts a new value. This continues until the entire array is sorted.

The selection sort algorithm manages two arrays inside one larger array. The array is sorted by searching for the minimum value and puts it at the beginning of the array. Each iteration of the selection sort algorithm, the element with the minimum value is moved to the sorted array.

Merge Sort is preferred for huge data sets. It happens to compare all the elements present in the array hence is not much helpful for small datasets, whereas insertion Sort is preferred for fewer elements. It becomes fast when data is already sorted or nearly sorted because it skips the sorted values.

Insertion sort builds the final array by moving one element at a time. Selection sort, in contrast, searches for the smallest array element and moves it to the correct location. Both algorithms have their own individual use cases for efficiency. Insertion is considered more efficient.