

```
1 // AP/Honors Java & Data Structures Name -
2 // Ch 18 Sorting Algorithms Period -
3
4
5 // ***** SELECTION *****
6
7 public static void selectionSort (int[] a)
8 {
9     for (int i = 0; i < a.length - 1; i++)
10    {
11        int minIndex = i;
12        int min = a[minIndex];
13
14
15        for ( int j = i + 1; j < a.length; j ++)
16        {
17
18            if ( a[j] < min)
19            {
20                minIndex = j;
21                min = a[minIndex];
22            }
23
24        }
25
26        int T = a[minIndex]; // swap
27        a [minIndex] = a[i];
28        a [i] = T;
29    }
30
31 }
32
33 // ***** INSERTION *****
34
35 public static void insertionSort (int[] a)
36 {
37
38     for (int i = 1; i < a.length; i ++)
39     {
40         int value = a[i];
41         int j = 0;
42
43         // could use a boolean flag variable
44         for ( j = i - 1; j >= 0 && a[j] > value; j -- )
45         {
46             a [j + 1] = a[j]; // shift
47         }
48         a [j + 1] = value; // insert
49     }
50
51 }
52
53 // ***** BUBBLE *****
54
55 public static void bubbleSort (int[] a)
56 {
57     boolean swapMade = false;
58
59     for (int i = 0; i < a.length; i ++)
60     {
61         swapMade = false;
62
63         for (int j = a.length -1; j > i; j--)
64         {
65             if (a[j-1] > a[j])
66             {
67                 int T = a[j-1]; // swap
```

```
68         a [j-1] = a[j];
69         a [j] = T;
70         swapMade = true;
71     }
72 }
73
74     if (!swapMade) break;
75
76 }
77
78 }
79
80 }
81
82 // ***** MERGE *****
83
84 public static void sort(int a[])
85 {
86     if (a.length > 1)
87         mergeSort (a, 0, a.length);
88 }
89
90 public static void mergeSort (int a[], int start, int segLength)
91 {
92     // Divide the array in half and sort and merge the halves
93     int half = segLength / 2;
94
95     // Sort each of the two parts of the arrays
96     if (half > 1)
97     {
98         mergeSort (a, start, half);
99     }
100
101     if (segLength - half > 1)
102     {
103         mergeSort (a, start + half, segLength - half);
104     }
105
106     // Copy the two parts of the arrays & merge
107     int t1[] = new int[half];
108     int t2[] = new int[segLength - half];
109     int i = start;
110
111     for (int j = 0; j < half; j++)
112     {
113         t1 [j] = a[i++];
114     }
115
116     for ( int j = 0; j < segLength - half; j++)
117     {
118         t2 [j] = a[i++];
119     }
120
121     merge (a, start, t1, t2 );
122 }
123
124 public static void merge (int a[], int start, int t1[], int t2[])
125 {
126     int index = start;
127     int end = start + t1.length + t2.length;
128     int t1index = 0;
129     int t2index = 0;
130
131     while (t1index < t1.length && t2index < t2.length)
132     {
133         if (t1[t1index] <= t2[t2index])
134             a [index++] = t1[t1index++];
```

```
135         else
136             a [index++] = t2[t2index++];
137     }
138
139     while (t1index < t1.length )
140     {
141         a [index++] = t1[t1index++];
142     }
143
144     while (t2index < t2.length )
145     {
146         a [index++] = t2[t2index++];
147     }
148
149 }
150
151 // ***** QUICK *****
152
153 public static void quickSort (int a[], int lo0, int hi0)
154 {
155     int lo = lo0;
156     int hi = hi0;
157     int mid = 0;
158
159     if ( hi0 > lo0)
160     {
161
162         // Arbitrarily establishing partition element as the midpoint of
163         // the array.
164
165         mid = a[(lo0 + hi0) / 2];
166
167         // loop through the array until indices cross
168         while (lo <= hi)
169         {
170             // find the first element that is greater than or equal to
171             // the partition element starting from the left Index.
172
173             while ((lo < hi0) && (a[lo] < mid ))
174             {
175                 ++lo;
176             }
177
178             // find an element that is smaller than or equal to
179             // the partition element starting from the right Index.
180
181             while ((hi > lo0) && (a[hi] > mid))
182             {
183                 --hi;
184             }
185
186             // if the indexes have not crossed, swap
187             if( lo <= hi )
188             {
189                 swap(a, lo, hi);
190
191                 ++lo;
192                 --hi;
193             }
194         }
195     }
196
197     // If the right index has not reached the left side of array
198     // must now sort the left partition.
199
200     if(lo0 < hi)
201     {
```

```
202         quickSort ( a, lo0, hi );
203     }
204
205     // If the left index has not reached the right side of array
206     // must now sort the right partition.
207
208     if (lo < hi0)
209     {
210         quickSort (a, lo, hi0);
211     }
212
213 }
214 }
215
216 private static void swap(int a[], int i, int j)
217 {
218     int T = 0;
219     T = a[i];
220     a [i] = a[j];
221     a [j] = T;
222 }
223
224 // ***** RADIX *****
225
226 public static void radixSort (int[] a, int max)
227 {
228     int bins [][] = new int [10][100];
229
230     int binCounters [] = new int [10];
231
232     for (int k = 1; k <= 4; ++k)
233     {
234         for (int j = 0; j < max; ++j)
235         {
236             addToBin (bins, binCounters, a [j], digit (a[j], k));
237         }
238
239         collectBins (a, bins, binCounters );
240         initializeCounters (binCounters );
241     }
242
243 }
244
245
246 public static int digit (int number, int k)
247 {
248     for (int i = 1; i < k; ++i)
249     {
250         number = number / 10;
251     }
252
253     return number % 10;
254 }
255
256
257 public static void initializeCounters (int[] binCounters )
258 {
259     for (int i = 0; i < 10; ++i)
260     {
261         binCounters [i] = 0;
262     }
263 }
264
265 public static void addToBin (int[][] bins, int[] binCounters,
266     int number, int place)
267 {
268     binCounters [place] = binCounters [place] + 1;
```

```
269     bins [place][binCounters [place]] = number;
270 }
271
272 public static void collectBins (int list [], int bins [][], int
binCounters [])
273 {
274     int place = 0;
275
276     for (int i = 0; i < 10; ++i)
277     {
278
279         for (int j = 1; j <= binCounters [i]; ++j)
280         {
281             list [place] = bins [i][j];
282             ++place;
283         }
284     }
285 }
286
287
288 // ***** HEAP *****
289
290 public static void heapSort (int[] a)
291 {
292     int n = a.length;
293
294     for (int v = n/2 - 1; v >= 0; v--)
295     {
296         downheap (a, v, n);
297     }
298
299     while (n > 1)
300     {
301         n --;
302         swap (a, 0, n);
303         downheap (a, 0, n);
304     }
305 }
306
307
308 public static void downheap (int[] a, int v, int n)
309 {
310     int w = 2 * v + 1;
311
312     while (w < n)
313     {
314
315         if (w + 1 < n)
316         {
317
318             if (a[w + 1] > a[w])
319             {
320                 w ++;
321             }
322
323         }
324
325         if (a[v] >= a[w])
326         {
327             return;
328         }
329
330         swap (a, v, w);
331         v = w;
332         w = 2 * v + 1;
333     }
334 }
```

```
335     }
336
337     public static void swap(int[] a, int i, int j)
338     {
339         int t = a[i];
340         a[i] = a[j];
341         a[j] = t;
342     }
343
```