

```
1 using System;
2 using System.CodeDom.Compiler;
3 using System.Collections.Generic;
4 using System.Data.SqlClient;
5 using System.Runtime.Remoting.Messaging;
6 using System.Security;
7
8 namespace Unit4
9 {
10     class QueueUtils
11     {
12         public static Queue<T> CreateQueueFromArray<T>(T[] arr)
13         {
14             Queue<T> s = new Queue<T>();
15
16             for (int i = 0; i < arr.Length; i++)
17             {
18                 s.Insert(arr[i]);
19             }
20
21             return s;
22         }
23         public static void SpilledOn<T>(Queue<T> dest, Queue<T> src)
24         {
25             while (!src.IsEmpty())
26             {
27                 T temp = src.Remove();
28                 dest.Insert(temp);
29             }
30         }
31
32         public static Queue<T> Clone<T>(Queue<T> s)
33         {
34             Queue<T> t = new Queue<T>();
35             Queue<T> s2 = new Queue<T>();
36
37             SpilledOn(t, s);
38
39             while(!t.IsEmpty())
40             {
41                 T temp = t.Remove();
42                 s.Insert(temp);
43                 s2.Insert(temp);
44             }
45
46             return s2;
47         }
48         public static int GetSize<T>(Queue<T> q)
49         {
```

```
50         int count = 0;
51
52         Queue<T> temp = Clone(q);
53         while (!temp.IsEmpty())
54         {
55             temp.Remove();
56             count++;
57         }
58
59         return count;
60     }
61
62     public static int GetSum(Queue<int> q)
63     {
64         int sum = 0;
65
66         Queue<int> temp = Clone(q);
67         while (!temp.IsEmpty())
68         {
69
70             sum+=temp.Remove();
71         }
72
73         return sum;
74     }
75
76     public static bool IsExist<T>(Queue<T> q, T e)
77     {
78         Queue<T> temp = Clone(q);
79
80         while (!temp.IsEmpty())
81         {
82             if (EqualityComparer<T>.Default.Equals(temp.Remove(), e))
83                 return true;
84         }
85         return false;
86     }
87
88     public static void LastToFirst<T>(Queue<T> q)
89     {
90         Queue<T> temp = new Queue<T>();
91
92         int l = GetSize(q) - 1;
93
94         for (int i = 0; i < l; i++)
95         {
96             temp.Insert(q.Remove());
97         }
98     }
```

```
99         while(!temp.IsEmpty())
100         {
101             q.Insert(temp.Remove());
102         }
103     }
104
105     public static bool IsSorted(Queue<int> q)
106     {
107         Queue<int> temp = Clone(q);
108
109         bool sorted = true;
110         int last = q.Head();
111         while(!temp.IsEmpty() && sorted)
112         {
113             int curr = temp.Remove();
114             sorted = last <= curr;
115             last = curr;
116         }
117
118         return sorted;
119     }
120
121     public static void InsertToSorted(Queue<int> q, int val)
122     {
123         Queue<int> temp = new Queue<int>();
124
125         bool found = false;
126
127         while (!q.IsEmpty())
128         {
129             int curr = q.Remove();
130             if (val < curr && !found)
131             {
132                 temp.Insert(val);
133                 temp.Insert(curr);
134                 found = true;
135             }
136             else
137             {
138                 temp.Insert(curr);
139             }
140         }
141
142     }
143
144     if (!found)
145     {
146         temp.Insert(val);
147     }
```

```
148
149         SpilledOn(q, temp);
150     }
151
152     public static int FindMin(Queue<int> q)
153     {
154         Queue<int> temp = Clone(q);
155
156         int min = int.MaxValue;
157
158         while(!temp.IsEmpty())
159         {
160             int val = temp.Remove();
161             if (val < min)
162                 min = val;
163         }
164
165         return min;
166     }
167
168     public static int FindMax(Queue<int> q)
169     {
170         Queue<int> temp = Clone(q);
171
172         int max = int.MinValue;
173
174         while (!temp.IsEmpty())
175         {
176             int val = temp.Remove();
177             if (val > max)
178                 max = val;
179         }
180
181         return max;
182     }
183
184     public static void RemoveMin(Queue<int> q)
185     {
186         Queue<int> temp = Clone(q);
187
188         int min_index = 0;
189         int min = int.MaxValue;
190         int cnt = 0;
191
192         while (!temp.IsEmpty())
193         {
194             int val = temp.Remove();
195             if (val < min)
196             {
```

```
197         min = val;
198         min_index = cnt;
199     }
200     cnt++;
201 }
202
203 SpilledOn(temp, q);
204
205 int l = GetSize(temp);
206
207 for (int i = 0; i < l; i++)
208 {
209     int val = temp.Remove();
210     if (i != min_index)
211     {
212         q.Insert(val);
213     }
214 }
215
216 }
217
218 public static void RemoveMax(Queue<int> q)
219 {
220     Queue<int> temp = Clone(q);
221
222     int max_index = 0;
223     int max = int.MinValue;
224     int cnt = 0;
225
226     while (!temp.IsEmpty())
227     {
228         int val = temp.Remove();
229         if (val > max)
230         {
231             max = val;
232             max_index = cnt;
233         }
234         cnt++;
235     }
236
237     SpilledOn(temp, q);
238
239     int l = GetSize(temp);
240
241     for (int i = 0; i < l; i++)
242     {
243         int val = temp.Remove();
244         if (i != max_index)
245         {
```

```
246         q.Insert(val);
247     }
248 }
249
250
251 public static void SortQueue(Queue<int> q)
252 {
253     int l = GetSize(q);
254     Queue<int> temp = new Queue<int>();
255
256     for (int i = 0; i < l; i++)
257     {
258         int val = FindMin(q);
259         temp.Insert(val);
260         RemoveMin(q);
261     }
262
263     SpilledOn(q, temp);
264 }
265
266 public static void Reverse<T>(Queue<T> q)
267 {
268     int l = GetSize(q);
269
270     Queue<T> new_q = new Queue<T>();
271     Queue<T> save_q = new Queue<T>();
272     SpilledOn(save_q, q);
273
274     for (int i = 0; i < l; i++)
275     {
276         Queue<T> temp = Clone(save_q);
277         for (int j = 0; j < l - i - 1; j++)
278             temp.Remove();
279         q.Insert(temp.Remove());
280     }
281 }
282
283 public static void RemoveDuplicates(Queue<int> q)
284 {
285     Queue<int> temp = new Queue<int>();
286     SpilledOn(temp, q);
287
288     while (!temp.IsEmpty())
289     {
290         int val = temp.Remove();
291         if (!IsExist(q, val))
292             q.Insert(val);
293     }
294 }
```

```
295
296     public static int Count(Queue<int> q, int n)
297     {
298         Queue<int> temp = Clone(q);
299
300         int cnt = 0;
301
302         while (!temp.IsEmpty())
303         {
304             if (temp.Remove() == n)
305                 cnt++;
306         }
307
308         return cnt;
309     }
310
311     public static void RemoveSpec(Queue<int> q, int val)
312     {
313         Queue<int> temp = new Queue<int>();
314         SpilledOn(temp, q);
315
316         while (!temp.IsEmpty())
317         {
318             int curr = temp.Remove();
319             if (curr != val)
320                 q.Insert(curr);
321         }
322     }
323
324     public static void InsertAtPos<T>(Queue<T> q, T e, int n)
325     {
326     }
327
328     // --- Bagrut Exercices ---
329     public static int ToNumber(Queue<int> q)
330     {
331         int num = 0;
332
333         while (!q.IsEmpty())
334         {
335             int val = q.Remove();
336
337             num *= 10;
338             num += val;
339         }
340
341         return num;
342     }
343
```

```
344     public static int BigNumber(Queue<Queue<int>> q)
345     {
346         int max = int.MinValue;
347
348         Queue<Queue<int>> clone = Clone(q);
349
350         while (!clone.IsEmpty())
351         {
352             int val = ToNumber(clone.Remove());
353             if (val > max)
354                 max = val;
355         }
356
357         return max;
358     }
359
360     // ---
361     public static void ConnectQueues<T>(Queue<T> q1, Queue<T> q2)
362     {
363         Queue<T> temp = Clone(q2);
364
365         while (!temp.IsEmpty())
366             q1.Insert(temp.Remove());
367     }
368     public static Queue<int> DoublesToPali(Queue<int> qd)
369     {
370         if (GetSize(qd) == 2)
371             return qd;
372
373         Queue<int> res = new Queue<int>();
374         res.Insert(qd.Remove());
375         int val = qd.Remove();
376         ConnectQueues(res, DoublesToPali(qd));
377         res.Insert(val);
378
379         return res;
380     }
381
382     // ---
383     public static bool IsIdentical(Queue<int> q1, Queue<int> q2)
384     {
385         Queue<int> copy1 = Clone(q1);
386         Queue<int> copy2 = Clone(q2);
387
388         bool identical = true;
389
390         while(identical && !copy1.IsEmpty() && !copy2.IsEmpty())
391         {
392
```



```
393         identical = copy1.Remove() == copy2.Remove();
394
395         if ((!copy1.IsEmpty() && copy2.IsEmpty()) ||
396             (copy1.IsEmpty() && !copy2.IsEmpty()))
397             identical = false;
398     }
399     return identical;
400 }
401
402 public static bool IsSimilar(Queue<int> q1, Queue<int> q2)
403 {
404     int size = GetSize(q1);
405
406     bool similar = false;
407
408     for (int i = 0; i < size && !similar; i++)
409     {
410         similar = IsIdentical(q1, q2);
411         LastToFirst(q1);
412     }
413
414     return similar;
415 }
416
417
418 // -- Bagrut 2023
419 public static bool TwoSum(Queue<int> q, int x)
420 {
421     bool found = false;
422     Queue<int> copy = Clone(q);
423
424     while (!found && !copy.IsEmpty())
425     {
426         Queue<int> temp = Clone(copy);
427         int head = temp.Remove();
428
429         while(!temp.IsEmpty() && !found)
430         {
431             found = head + temp.Remove() == x;
432         }
433
434         copy.Remove();
435     }
436
437     return found;
438
439 }
440 }
```

```
441
442     }
443 }
444
445
446
447
448
449
450
451
452
453
454
455
```