```
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         {
```

```
            int count = 0;

            Queue<T> temp = Clone(q);
            while (!temp.IsEmpty())
            {
                temp.Remove();
                count++;
            }

            return count;
        }

        public static int GetSum(Queue<int> q)
        {
            int sum = 0;

            Queue<int> temp = Clone(q);
            while (!temp.IsEmpty())
            {

                sum+=temp.Remove();
            }

            return sum;
        }

        public static bool IsExist<T>(Queue<T> q, T e)
        {
            Queue<T> temp = Clone(q);

            while (!temp.IsEmpty())
            {
                if (EqualityComparer<T>.Default.Equals(temp.Remove(), e))
                    return true;
            }
            return false;
        }

        public static void LastToFirst<T>(Queue<T> q)
        {
            Queue<T> temp = new Queue<T>();

            int l = GetSize(q) - 1;

            for (int i = 0; i < l; i++)
            {
                temp.Insert(q.Remove());
            }

```

```
            while(!temp.IsEmpty())
            {
                q.Insert(temp.Remove());
            }
        }

        public static bool IsSorted(Queue<int> q)
        {
            Queue<int> temp = Clone(q);

            bool sorted = true;
            int last = q.Head();
            while(!temp.IsEmpty() && sorted)
            {
                int curr = temp.Remove();
                sorted = last <= curr;
                last = curr;
            }

            return sorted;
        }

        public static void InsertToSorted(Queue<int> q,  int val)
        {
            Queue<int> temp = new Queue<int>();

            bool found = false;

            while (!q.IsEmpty())
            {
                int curr = q.Remove();
                if (val < curr && !found)
                {
                    temp.Insert(val);
                    temp.Insert(curr);
                    found = true;
                }
                else
                {
                    temp.Insert(curr);

                }

            }

            if (!found)
            {
                temp.Insert(val);
            }
```

```

            SpilledOn(q, temp);
        }

        public static int FindMin(Queue<int> q)
        {
            Queue<int> temp = Clone(q);

            int min = int.MaxValue;

            while(!temp.IsEmpty())
            {
                int val = temp.Remove();
                if (val < min)
                    min = val;
            }

            return min;
        }

        public static int FindMax(Queue<int> q)
        {
            Queue<int> temp = Clone(q);

            int max = int.MinValue;

            while (!temp.IsEmpty())
            {
                int val = temp.Remove();
                if (val > max)
                    max = val;
            }

            return max;
        }

        public static void RemoveMin(Queue<int> q)
        {
            Queue<int> temp = Clone(q);

            int min_index = 0;
            int min = int.MaxValue;
            int cnt = 0;

            while (!temp.IsEmpty())
            {
                int val = temp.Remove();
                if (val < min)
                {
```

```
                    min = val;
                    min_index = cnt;
                }
                cnt++;
            }

            SpilledOn(temp, q);

            int l = GetSize(temp);

            for (int i = 0; i < l; i++)
            {
                int val = temp.Remove();
                if (i != min_index)
                {
                    q.Insert(val);
                }
            }

        }

        public static void RemoveMax(Queue<int> q)
        {
            Queue<int> temp = Clone(q);

            int max_index = 0;
            int max = int.MinValue;
            int cnt = 0;

            while (!temp.IsEmpty())
            {
                int val = temp.Remove();
                if (val > max)
                {
                    max = val;
                    max_index = cnt;
                }
                cnt++;
            }

            SpilledOn(temp, q);

            int l = GetSize(temp);

            for (int i = 0; i < l; i++)
            {
                int val = temp.Remove();
                if (i != max_index)
                {
```

```
                    q.Insert(val);
                }
            }
        }

        public static void SortQueue(Queue<int> q)
        {
            int l = GetSize(q);
            Queue<int> temp = new Queue<int>();

            for (int i = 0; i < l; i++)
            {
                int val = FindMin(q);
                temp.Insert(val);
                RemoveMin(q);
            }

            SpilledOn(q, temp);
        }

        public static void Reverse<T>(Queue<T> q)
        {
            int l = GetSize(q);

            Queue<T> new_q = new Queue<T>();
            Queue<T> save_q = new Queue<T>();
            SpilledOn(save_q, q);

            for (int i = 0; i < l; i++)
            {
                Queue<T> temp = Clone(save_q);
                for (int j = 0; j < l - i - 1; j++)
                    temp.Remove();
                q.Insert(temp.Remove());
            }
        }

        public static void RemoveDuplicates(Queue<int> q)
        {
            Queue<int> temp = new Queue<int>();
            SpilledOn(temp, q);

            while (!temp.IsEmpty())
            {
                int val = temp.Remove();
                if (!IsExist(q, val))
                    q.Insert(val);
            }
        }
```

```
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
```

```
        public static int BigNumber(Queue<Queue<int>> q)
        {
            int max = int.MinValue;

            Queue<Queue<int>> clone = Clone(q);

            while (!clone.IsEmpty())
            {
                int val = ToNumber(clone.Remove());
                if (val > max)
                    max = val;
            }

            return max;
        }

        // ---
        public static void ConnectQueues<T>(Queue<T> q1, Queue<T> q2)
        {
            Queue<T> temp = Clone(q2);

            while (!temp.IsEmpty())
                q1.Insert(temp.Remove());
        }
        public static Queue<int> DoublesToPali(Queue<int> qd)
        {
            if (GetSize(qd) == 2)
                return qd;

            Queue<int> res = new Queue<int>();
            res.Insert(qd.Remove());
            int val = qd.Remove();
            ConnectQueues(res, DoublesToPali(qd));
            res.Insert(val);

            return res;
        }

        // ---
        public static bool IsIdentical(Queue<int> q1, Queue<int> q2)
        {
            Queue<int> copy1 = Clone(q1);
            Queue<int> copy2 = Clone(q2);

            bool identical = true;

            while(identical && !copy1.IsEmpty() && !copy2.IsEmpty())
            {

```

```
                identical = copy1.Remove() == copy2.Remove();

                if ((!copy1.IsEmpty() && copy2.IsEmpty()) ||
                  (copy1.IsEmpty() && !copy2.IsEmpty()))
                    identical = false;
            }

            return identical;
        }

        public static bool IsSimilar(Queue<int> q1, Queue<int> q2)
        {
            int size = GetSize(q1);

            bool similar = false;

            for (int i = 0; i < size && !similar; i++)
            {
                similar = IsIdentical(q1, q2);
                LastToFirst(q1);
            }

            return similar;

        }

        // -- Bagrut 2023
        public static bool TwoSum(Queue<int> q, int x)
        {
            bool found = false;
            Queue<int> copy = Clone(q);

            while (!found && !copy.IsEmpty())
            {
                Queue<int> temp = Clone(copy);
                int head = temp.Remove();

                while(!temp.IsEmpty() && !found)
                {
                    found = head + temp.Remove() == x;
                }

                copy.Remove();
            }

            return found;


        }
```

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