```
 1  using System.Xml.Linq;
 2
 3  namespace DAMLib
 4  {
 5      public class OrderedSet<T>
 6      {
 7          private Item[] _orderedSet;
 8
 9          private class Item
10          {
11              public T element;
12              public int hash;
13
14              public Item()
15              {
16
17              }
18              public Item(T element, int hash)
19              {
20                  this.element = element;
21                  this.hash = hash;
22              }
23          }
24
25
26          public int Count
27          {
28              get
29              {
30                  if (_orderedSet == null)
31                      return 0;
32                  return _orderedSet.Length;
33              }
34          }
35          public bool IsEmpty => _orderedSet.Length == 0;
36
37
38          public OrderedSet()
39          {
40              _orderedSet = Array.Empty<Item>();
41          }
42
43          // Funcion publica que añade un elemento al Set. Despues lo     ↵
              ordena.
44          public void Add(T element)
45          {
46              if (element == null)
47                  return;
48
49              if (Contains(element))
50                  return;
51              else
52                  AddElement(element);
```

```
53
54              SortSet();
55          }
56
57          // Funcion que añade un elemento al Set.
58          private void AddElement(T element)
59          {
60              if (element == null)
61                  return;
62
63              int newLength = _orderedSet.Length + 1;
64              Item[] newItemArray = new Item[newLength];
65
66              int hash = element.GetHashCode();
67              Item newItem = new Item(element, hash);
68
69              for (int i = 0; i < newLength - 1; i++)
70              {
71                  newItemArray[i] = _orderedSet[i];
72              }
73              newItemArray[newLength - 1] = newItem;
74
75              _orderedSet = newItemArray;
76          }
77
78          // Funcion que elimina el item que se encuentra en la posicion ↵
              del index.
79          public void RemoveAt(int index)
80          {
81              if (index < 0 || index >= _orderedSet.Length)
82                  return;
83
84              int newLength = _orderedSet.Length - 1;
85              Item[] newItemArray = new Item[newLength];
86
87
88              for (int i = 0; i < index; i++)
89              {
90                  newItemArray[i] = _orderedSet[i];
91              }
92
93              for (int i = index; i < newLength; i++)
94              {
95                  newItemArray[i] = _orderedSet[i + 1];
96              }
97
98              _orderedSet = newItemArray;
99          }
100
101         // Funcion que ordena el Set de menor a mayor.
102         public void SortSet()
103         {
104             int count = _orderedSet.Length;
```

```
105             Item aux;
106
107             for (int i = 0; i < count - 1; i++)
108             {
109                 for (int j = i + 1; j < count; j++)
110                 {
111                     if (_orderedSet[i].hash > _orderedSet[j].hash)
112                     {
113                         aux = _orderedSet[i];
114                         _orderedSet[i] = _orderedSet[j];
115                         _orderedSet[j] = aux;
116                     }
117                 }
118             }
119         }
120
121         // Funcion que realiza una busqueda binaria de un elemento
              segun el Hash.
122         public T BinarySearch(T element)
123         {
124             if (element == null)
125                 return default(T);
126
127             int hash = element.GetHashCode();
128             int superiorIndex = _orderedSet.Length.GetHashCode();
129             int inferiorIndex = _orderedSet[0].GetHashCode();
130
131
132             while(superiorIndex > inferiorIndex)
133             {
134                 int searchIndex = superiorIndex / inferiorIndex;
135
136                 if (element.Equals(_orderedSet[searchIndex].element))
137                     return _orderedSet[searchIndex].element;
138
139                 if(hash > searchIndex)
140                 {
141                     inferiorIndex = searchIndex + 1;
142                 }
143                 else
144                 {
145                     superiorIndex = searchIndex - 1;
146                 }
147             }
148
149             return default(T);
150         }
151
152         // Funcion que evalua si el Set contiene un elemento.
153         public bool Contains(T element)
154         {
155             return IndexOf(element) >= 0;
156         }
```

```
157
158         // Funcion que devuelve el indice de un elemento dentro del Set.
159         public int IndexOf(T element)
160         {
161             if (element == null)
162                 return -1;
163
164             int hash = element.GetHashCode();
165
166             for (int i = 0; i < _orderedSet.Length; i++)
167             {
168                 Item item = _orderedSet[i];
169                 if (hash == item.hash && item.element.Equals(element))
170                 {
171                     return i;
172                 }
173             }
174             return -1;
175         }
176
177         // Funcion que elimina todos los elementos del Set.
178         public void Clear()
179         {
180             _orderedSet = new Item[0];
181         }
182
183         // Funcion que sobreescribe el metodo para recoger el Hash de un elemento.
184         public override int GetHashCode()
185         {
186             return 133 * 533 * 224 * _orderedSet.GetHashCode();
187         }
188
189         public override string ToString()
190         {
191             string result = "";
192             int count = 0;
193
194             foreach(Item i in _orderedSet)
195             {
196                 result += $"El elemento numero {count} de la coleccion es: {i.element}.\n";
197                 count++;
198             }
199             return result;
200         }
201     }
202 }
203
```