Київський національний університет імені Тараса Шевченка

ФРЕКС

Звіт до лабораторної роботи №5
з теми «Рефакторинг»
курсу ІПЗ

Виконав студент ІІІ курсу

КІ, група СА

Кравченко В’ячеслав

2018

CODE SMELLS

Іноді коментарі достатньо змістовні – єдине позитивне

Деякі методи мали назви не у тому ж стилі, що й інші, тому відформатував у 1 стиль

Іноді переплутані (виправлено)

```
public int HeapSize
{
    get { return _heapSize; }
    private set
    {
        if (value > _heapCapacity) throw new ArgumentException();
        if (value == _heapCapacity) _heapCapacity++;
        _heapSize = value;//встановлюємо висоту
    }
}
public int Height { get { return _heapHeight; } }
public bool Empty//перевірка, чи не порожня купа
{ get { return _heapSize == 0; } }

public int Min()//максимальний елемент купи
{
    return (int)(!Empty ? Array[0] : 0);
}

public int Max()//мінімальний елемент в купі
{
    if (Empty) return 0;
    int max = (int)Array[_heapSize - _heapHeight - 1];
    for (int i = _heapSize - _heapHeight - 1; i < _heapSize; ++i)
    {
        if (Array[i] > max)
```

Дублювання коду + враховуючи особливості методу – неможливо потрапити в цю гілку

```
/*цей пошук аналогічний до попереднього */
if (Empty)
{
    return 0;
}
```

Гілки, в які я так і не зміг потрапити після 20 тестів через перекриття іншою умовою та\або складністю/неправильністю обрахунку(умови)

```
if (valueToSearch == Array[position])//якщо елемент знайдено - повертаємо його позицію
    return position;
if (Array[position] == null || valueToSearch >= Array[position])
{
    if (position == (_heapHeight + 1) * (_heapHeight + 2) / 2 - 1)
    {
        break;
    }

    if (position == _heapHeight * (_heapHeight + 1) / 2 - 1 && _heapSize != (_heapHeight + 1) * (_heapHeigl
    {
        break;
    }

    if (position + h + 2 < _heapSize)
    {
        position += h + 2;
        ++h;
    }
    else
```

Аналогічна проблема

```
        else if (valueToSearch < Array[position])
        {
            if (position == (h + 1) * (h + 2) / 2 - 1)
            {
                break;
            }
            position -= h;
            --h;
        }
    }
    return -1;//інакше - повертаємо -1
}
```

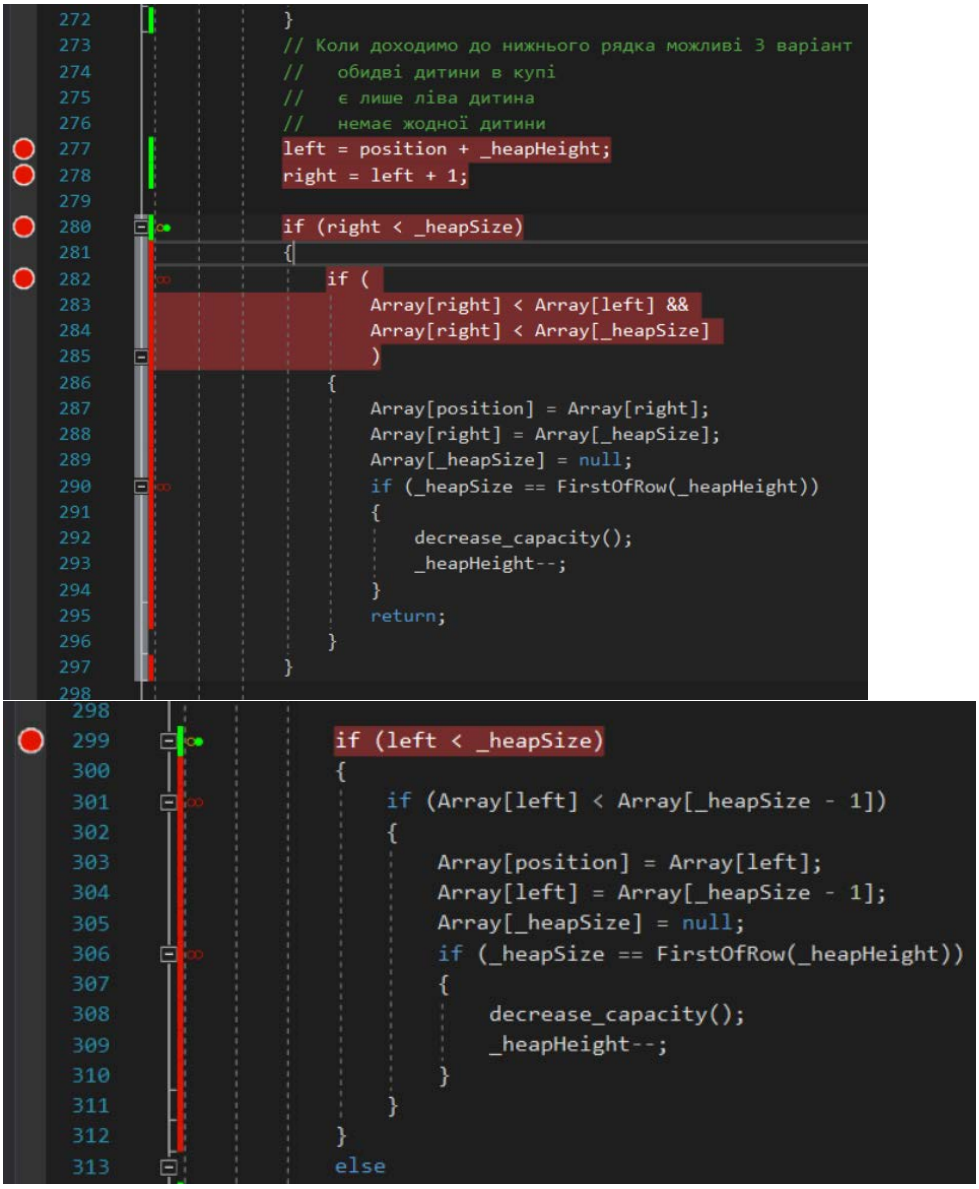
Аналогічно через логіку методу та інші умови сюди нереально потрапити

```
213         if (valueToInsert < Array[0])//перевіряємо чи є елемент найменшим в купі
214         {
215             Array[position] = Array[0];
216             Array[0] = valueToInsert;
217         }
218         else//або просто вставляємо на останню позицію
219         {
220             Array[position] = valueToInsert;
221         }
```

Перша умова перекриває другу і неможливо потрапити в 2 (такі методи прибрано)

```
226         if (!Find(valueToDlete)) return;//перевіряємо чи є елемент для видалення в масиві
227         int position = Count(valueToDlete);
228         //якщо купа пуста немає сенсу видаляти перший елемент
229         if (Empty)
230         {
231             return;
232         }
```

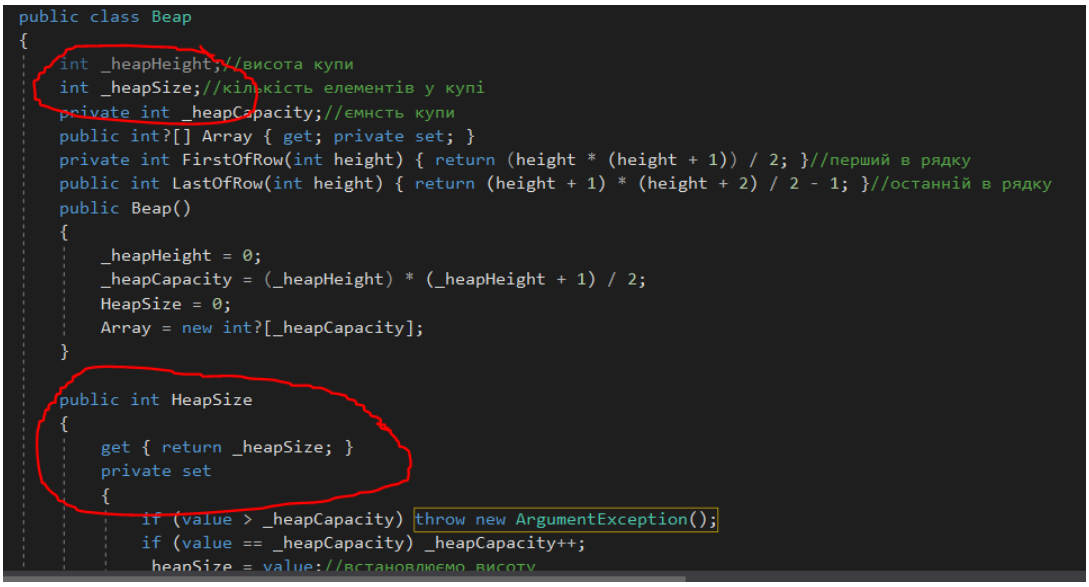
Неправильний обрахунок в умові → надлишкові умови, без яких все і так працює. Або ж недостатньо коментарів\прозорості коду для розуміння



```
272             }
273             // Коли доходимо до нижнього рядка можливі 3 варіант
274             //   обидві дитини в купі
275             //   є лише ліва дитина
276             //   немає жодної дитини
277             left = position + _heapHeight;
278             right = left + 1;
279
280             if (right < _heapSize)
281             {
282                 if (
283                     Array[right] < Array[left] &&
284                     Array[right] < Array[_heapSize]
285                     )
286                 {
287                     Array[position] = Array[right];
288                     Array[right] = Array[_heapSize];
289                     Array[_heapSize] = null;
290                     if (_heapSize == FirstOfRow(_heapHeight))
291                     {
292                         decrease_capacity();
293                         _heapHeight--;
294                     }
295                     return;
296                 }
297             }
298
```

```
298
299             if (left < _heapSize)
300             {
301                 if (Array[left] < Array[_heapSize - 1])
302                 {
303                     Array[position] = Array[left];
304                     Array[left] = Array[_heapSize - 1];
305                     Array[_heapSize] = null;
306                     if (_heapSize == FirstOfRow(_heapHeight))
307                     {
308                         decrease_capacity();
309                         _heapHeight--;
310                     }
311                 }
312             }
313             else
```

Поле непотрібне, коли є властивість

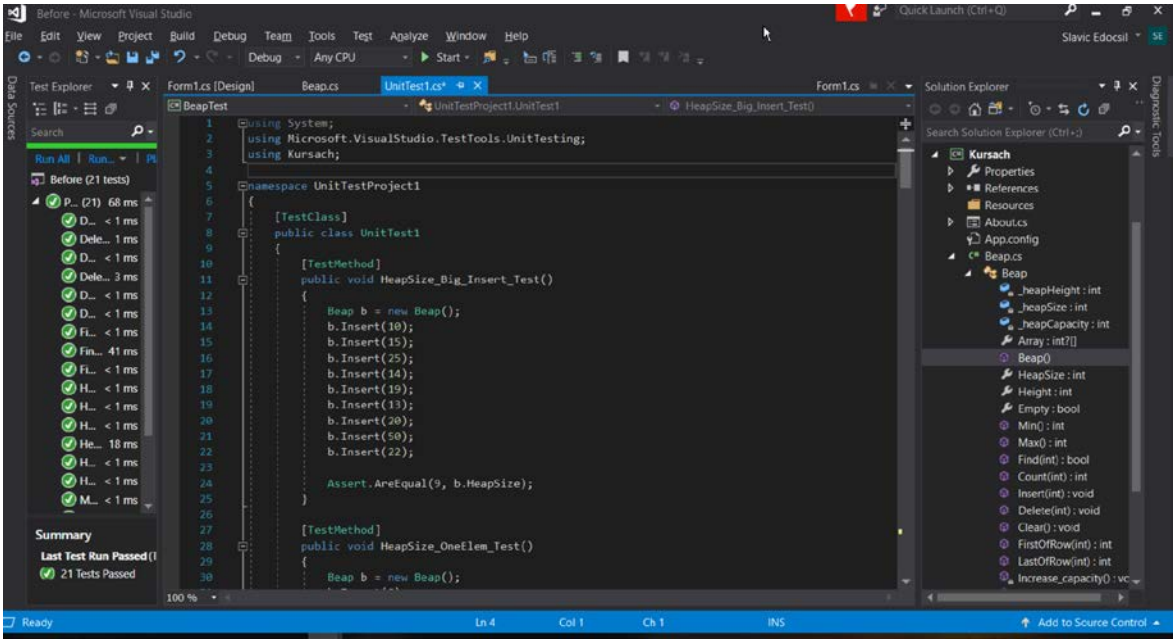


```
public class Beap
{
    int _heapHeight;//висота купи
    int _heapSize;//кількість елементів у купі
    private int _heapCapacity;//ємність купи
    public int?[] Array { get; private set; }
    private int FirstOfRow(int height) { return (height * (height + 1)) / 2; }//перший в рядку
    public int LastOfRow(int height) { return (height + 1) * (height + 2) / 2 - 1; }//останній в рядку
    public Beap()
    {
        _heapHeight = 0;
        _heapCapacity = (_heapHeight) * (_heapHeight + 1) / 2;
        HeapSize = 0;
        Array = new int?[_heapCapacity];
    }

    public int HeapSize
    {
        get { return _heapSize; }
        private set
        {
            if (value > _heapCapacity) throw new ArgumentException();
            if (value == _heapCapacity) _heapCapacity++;
            _heapSize = value;//встановлюємо висоту
```

Дублювання коду замість винесення в окремий метод

```
else
{
    ++position;
    if (position >= _heapSize)
    {
        position -= h;
        --h;
    }
}
```

UNIT TESTS

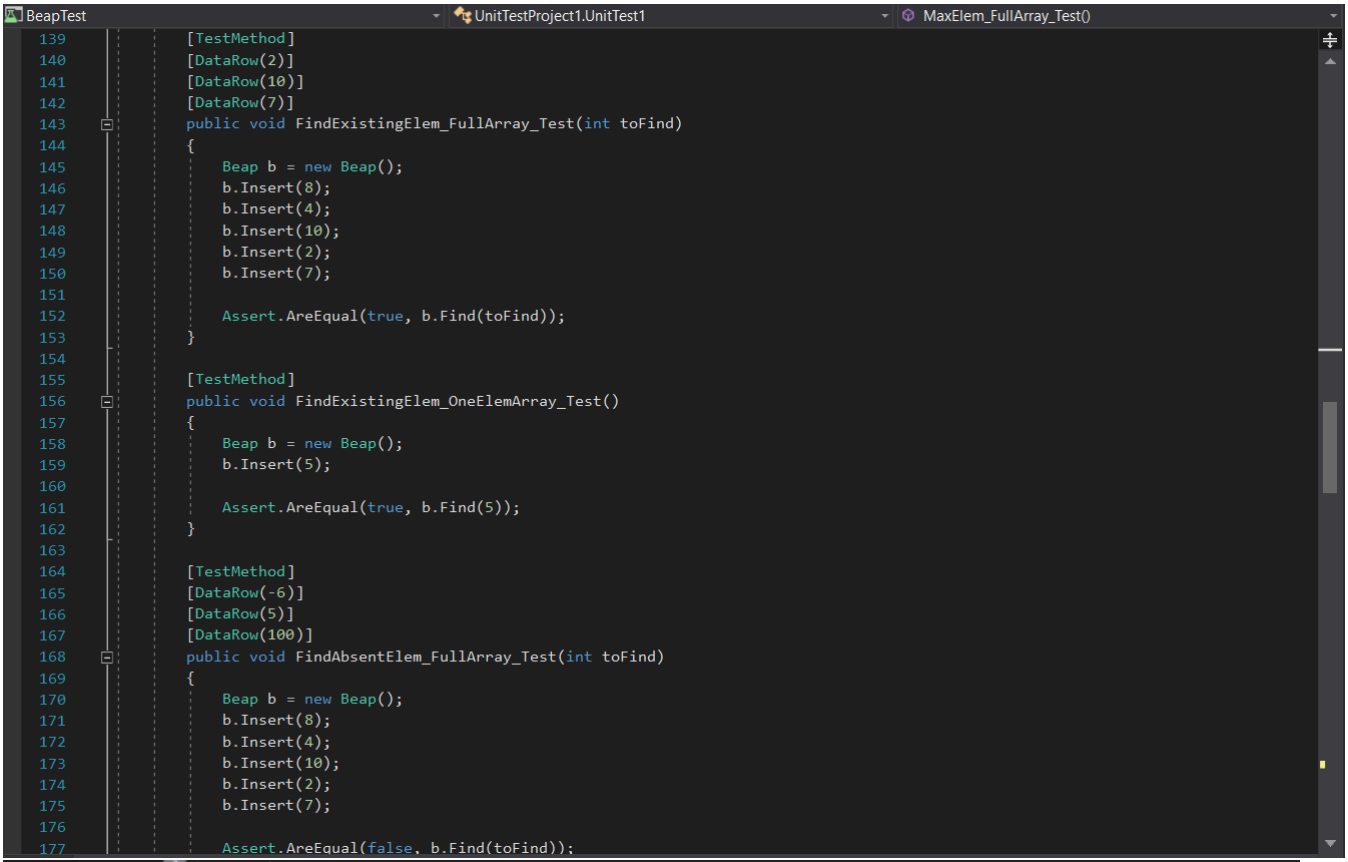



```
[TestMethod]
public void HeapSize_OneElem_Test()
{
    Beap b = new Beap();
    b.Insert(8);

    Assert.AreEqual(1, b.HeapSize);
}

[TestMethod]
public void HeapSize_EmptyArray_Test()
{
    Beap b = new Beap();

    Assert.AreEqual(0, b.HeapSize);
}

[TestMethod]
public void HeapSize_AfterClear_Test()
{
    Beap b = new Beap();
    b.Insert(10);
    b.Insert(4);
    b.Insert(2);
    b.Insert(8);

    b.Clear();

    Assert.AreEqual(0, b.HeapSize);
```

```
58          [TestMethod]
59          public void HeapHeight_FullArray_Test()
60          {
61              Beap b = new Beap();
62              b.Insert(8);
63              b.Insert(4);
64              b.Insert(10);
65              b.Insert(2);
66
67              Assert.AreEqual(2, b.Height);
68          }
69
70          [TestMethod]
71          public void HeapHeight_OneElemArray_Test()
72          {
73              Beap b = new Beap();
74              b.Insert(8);
75
76              Assert.AreEqual(0, b.Height);
77          }
78
79          [TestMethod]
80          public void MinElem_FullArray_Test()
81          {
82              Beap b = new Beap();
83              b.Insert(8);
84              b.Insert(4);
85              b.Insert(10);
86              b.Insert(2);
87              b.Insert(7);
88
89              Assert.AreEqual(2, b.Min());
90          }
```

```
92          [TestMethod]
93          public void MinElem_OneElemArray_Test()
94          {
95              Beap b = new Beap();
96              b.Insert(8);
97
98              Assert.AreEqual(8, b.Min());
99          }
100
101         [TestMethod]
102         public void MinElem_EmptyArray_Test()
103         {
104             Beap b = new Beap();
105
106             Assert.AreEqual(0, b.Min());
107         }
108
109         [TestMethod]
110         public void MaxElem_FullArray_Test()
111         {
112             Beap b = new Beap();
113             b.Insert(8);
114             b.Insert(4);
115             b.Insert(10);
116             b.Insert(2);
117             b.Insert(7);
118
119             Assert.AreEqual(10, b.Max());
120         }
```

```
122         [TestMethod]
123         public void MaxElem_OneElemArray_Test()
124         {
125             Beap b = new Beap();
126             b.Insert(9999);
127
128             Assert.AreEqual(9999, b.Max());
129         }
130
131         [TestMethod]
132         public void MaxElem_EmptyArray_Test()
133         {
134             Beap b = new Beap();
135
136             Assert.AreEqual(0, b.Max());
137         }
```




```
139     [TestMethod]
140     [DataRow(2)]
141     [DataRow(10)]
142     [DataRow(7)]
143     public void FindExistingElem_FullArray_Test(int toFind)
144     {
145         Beap b = new Beap();
146         b.Insert(8);
147         b.Insert(4);
148         b.Insert(10);
149         b.Insert(2);
150         b.Insert(7);
151
152         Assert.AreEqual(true, b.Find(toFind));
153     }
154
155     [TestMethod]
156     public void FindExistingElem_OneElemArray_Test()
157     {
158         Beap b = new Beap();
159         b.Insert(5);
160
161         Assert.AreEqual(true, b.Find(5));
162     }
163
164     [TestMethod]
165     [DataRow(-6)]
166     [DataRow(5)]
167     [DataRow(100)]
168     public void FindAbsentElem_FullArray_Test(int toFind)
169     {
170         Beap b = new Beap();
171         b.Insert(8);
172         b.Insert(4);
173         b.Insert(10);
174         b.Insert(2);
175         b.Insert(7);
176
177         Assert.AreEqual(false, b.Find(toFind));
```

```
180         [TestMethod]
181         [DataRow(2)]
182         [DataRow(10)]
183         [DataRow(7)]
184         public void DeletingElem_FullArray_Test(int toDelete)
185         {
186             Beap b = new Beap();
187             b.Insert(8);
188             b.Insert(4);
189             b.Insert(10);
190             b.Insert(2);
191             b.Insert(7);
192
193             b.Delete(toDelete);
194
195             Assert.AreEqual(4, b.HeapSize);
196         }
197
198         [TestMethod]
199         public void DeletingElem_OneElemArray_Test()
200         {
201             Beap b = new Beap();
202             b.Insert(5);
203
204             b.Delete(5);
205
206             Assert.AreEqual(0, b.HeapSize);
207         }
```

```
209         [TestMethod]
210         public void DeletingElem_EmptyArray_Test()
211         {
212             Beap b = new Beap();
213
214             b.Delete(5);
215
216             Assert.AreEqual(0, b.HeapSize);
217         }
218
219         [TestMethod]
220         [DataRow(10, 50, 25, 13)]// for branch  "else if (Array[right] < Array[_heapSize])"
221         [DataRow(25, 50, 20, 13)]
222         public void DeletingElem_ForBranches_Test(int a, int q, int c, int d)
223         {
224             Beap b = new Beap();
225             b.Insert(10);
226             b.Insert(15);
227             b.Insert(14);
228             b.Insert(25);
229             b.Insert(13);
230             b.Insert(19);
231             b.Insert(20);
232             b.Insert(50);
233             b.Insert(22);
234
235             b.Delete(a);
236             b.Delete(q);
237             b.Delete(c);
238             b.Delete(d);
239
240             Assert.AreEqual(5, b.HeapSize);
241         }
```

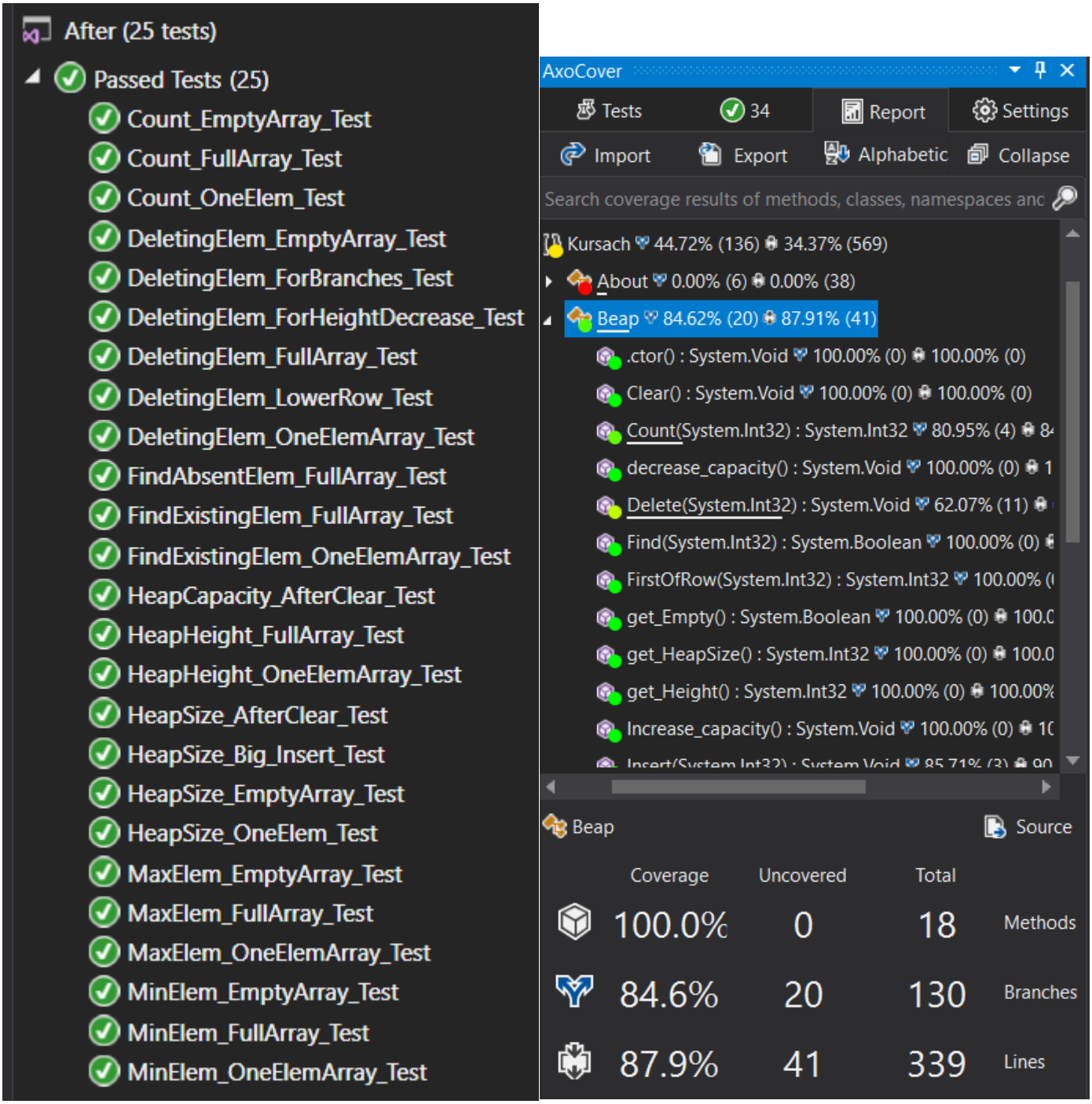
```
[TestMethod]
[ExpectedException(typeof(Exception), "BAD: no exception was generated!")]
public void MaxElem_EmptyArray_Test()
{
    Beap b = new Beap();

    b.Max();
}
```

```
243         [TestMethod]
244         [DataRow(25)]
245         public void DeletingElem_ForHeightDecrease_Test(int a)
246         {
247             Beap b = new Beap();
248             b.Insert(10);
249             b.Insert(15);
250             b.Insert(14);
251             b.Insert(25);
252             b.Insert(13);
253             b.Insert(19);
254             b.Insert(20);
255 
256             b.Delete(a);
257 
258             Assert.AreEqual(6, b.HeapSize);
259         }
260 
261         [TestMethod]
262         [DataRow(0)]
263         [DataRow(1)]
264         [DataRow(-1)]
265         [DataRow(2)]
266         public void DeletingElem_LowerRow_Test(int a)
267         {
268             Beap b = new Beap();
269             b.Insert(0);
270             b.Insert(-1);
271             b.Insert(1);
272             b.Insert(2);
273 
274             b.Delete(a);
275 
276             Assert.AreEqual(3, b.HeapSize);
277         }
```

Завдяки продуманості тестів було досягнуто майже повного покриття коду, розгалуджень та всіх методів.

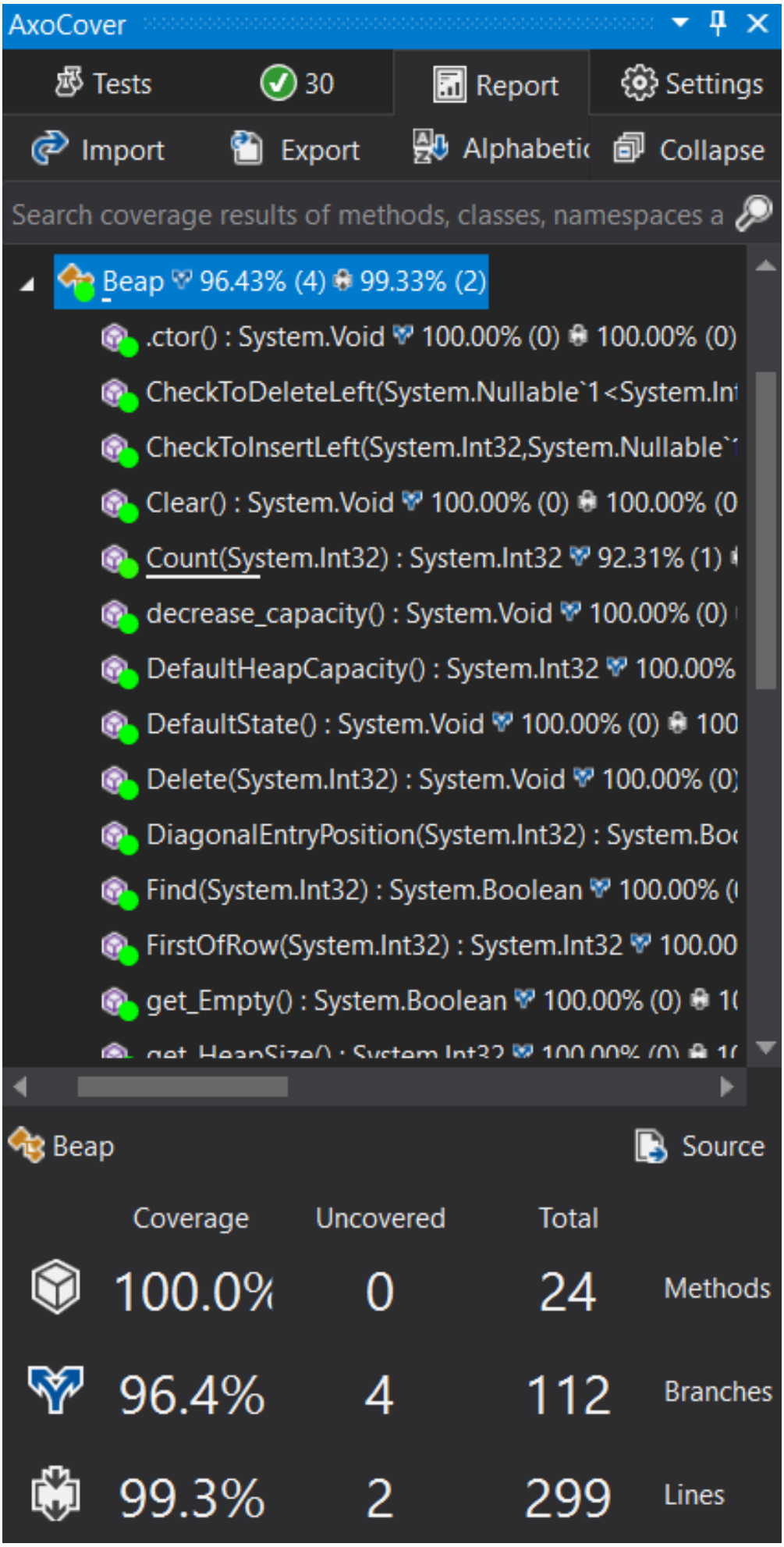
Назви тестових методів відображають логіку перевірки.

Для кращої перевірки було використано у деяких випадках параметризовані тести та тест на перевірку вкидання винятків (Exception).



Тут зображено покриття до рефакторингу

Та після рефакторингу :



Перелік виконаних рефакторингів

- Деякі поля винесені у властивості

```
public int Height { get; private set; }//висота купи
public int HeapCapacity { get; private set; }//ємність купи
```

- Виправлено та уточнене деякі коментарі
- Назви методів змінних у єдиному стилі
- Видалено неробочі розгалдуження (про які було сказано у 1 пункті звіту)
- Замість дублювання коду винесено в методи:

```
//часто використовувана величина купи
public int DefaultHeapCapacity() { return (Height) * (Height + 1) / 2; }

//дефолтний стан класу
public void DefaultState()
{
    Height = 0;
    HeapCapacity = DefaultHeapCapacity();
    HeapSize = 0;
    Array = new int?[] {null};
}
```

- Викидання винятку замість повертання 0, що було неправильно, бо 0 міг бути просто максимальним елементом в купі + так не було зрозуміло, як обробляти такий return

```
public int Max()//максимальний елемент в купі
{
    if (Empty) throw new Exception("use of empty array!");
    int max = (int)Array[HeapSize - Height - 1];
    for (int i = HeapSize - Height - 1; i < HeapSize; ++i)
    {
        if (Array[i] > max)
        {
            max = (int)Array[i];
        }
    }
    return max;
}
```

- Замість того, щоб щоразу перевіряти вручну, що означає якась складна умова – вони були винесені у методи

```
//перевірка, чи дійшли до діагоналі
public bool DiagonalEntryPosition(int position)
{
    return position == (Height + 1) * (Height + 2) / 2 - 1;
}
//перевірка чи дійшли до правого крайнього елемента
public bool RightEdgePosition(int position)
{
    return position == (Height * (Height + 1) / 2 - 1)
        &&
           HeapSize != ((Height + 1) * (Height + 2) / 2);
}
//перевірка та рух вниз і вправо
public bool NeedToGoDownAndRight(int position, int height)
{
    return position == ((height + 1) * (height + 2) / 2 - 1);
}
```

```
//чи можна рухатися по діагоналі
public bool CanMoveDiagonal(int? position, int? valueToSearch)
{
    return position == null || position < valueToSearch;
}
```

```
//перевірка на вставку зліва
public bool CheckToInsertLeft(int value, int? leftParent, int? rightParent)
{
   return value < leftParent && leftParent > rightParent;
}
```

- От як це тепер виглядає у коді:

```
if (CanMoveDiagonal(Array[position], valueToSearch))//відбувається просування по діагоналі
{
    if (RightEdgePosition(position))//якщо дійшли до крайього правого елемента - пошук завершено
    {
        return false;
    }
    if (DiagonalEntryPosition(position))//якщо дійшли до діагоналі - пошук завершено
    {
        return false;
    }
    if (position >= HeapSize)
    {
        position -= h;
        h--;
    }
```

ВИСНОВОК

У ході виконання я ознайомився з Юніт Тестами, їх складовими та реалізацією у Visual Studio.

Також отримав досвід з рефакторингу та перевірки відносно робочого та, певно, не найгіршого коду, який можна було написати для виконання поставленої задачі.

Уцілому, працювати, тестувати та рефакторити такий функціонально повний, але не без огріхів, код було корисно для мене як розробника. Я повністю відчув сенс інженерії програмного забезпечення та важливість писати гарний, приємний, ефективний код, який не має «запахів», та який буде не соромно передати у спадок іншим розробникам