どうぐばこ

Array.BinarySearch

計算量は $O(\log_2 N)$
配列の二分探索は標準ライブラリにある。

```
public static int BinarySearch (Array array, object value);
public static int BinarySearch (Array array, int index, int length, object value);
```

1 次元のソート済配列のなかから特定の要素を検索する。戻り値は、value が存在する場合、指定した array における指定した value のインデックス。それ以外の場合は負の数値。

bisects

Array.BinarySearch の負の戻り値を用いれば bisect_right のようなこともできるようだが、なんか記述がめんどくさくなりそうなので、用意することにした（snippets/Bisect.cpp ）。

```
    public static int BisectRight<T>(T[] a, T x, int lo=0, int hi=-1) where T:IComparable
    public static int BisectLeft<T>(T[] a, T x, int lo=0, int hi=-1) where T:IComparable
```

配列中に、検索対象と等しい値がひとつ以上あったときに、BisectRight はそれらの一番右のものより、さらに右隣りのインデックスを返す。BisectLeft は、それらの一番左のもののインデックスを返す。
配列中に、検索対象と等しい値がなかった場合には、検索対象よりも大きな要素のうち最小のもののインデックスを返す（Right/Left 共通）

```
    public static void Main()
    {
        int[] a = {1, 2, 3, 3, 4, 4, 5};
        AssertEquals(4, Bisect.BisectRight(a, 3));
        AssertEquals(2, Bisect.BisectLeft(a, 3));

        double[] b = {1.0, 2.0, 3.14, 4.28, 5.01};
        AssertEquals(2, Bisect.BisectRight(b, 3.0));
        AssertEquals(2, Bisect.BisectLeft(b, 3.0));
    }
```

Priority Queue

小さいものからでてくるものと、大きいものから出てくるものとを、コンストラクタ引数で指定できる。`void Push(T x)`, `T Pop()` 以外にも、`int Count()`, `T Peep()` のメソッドがある。

```
public class Program
{
    public static void Main()
    {
        PriorityQueue<int> Q0 = new PriorityQueue<int>(0);
        Q0.Push(1);
        Q0.Push(3);
        Q0.Push(2);
        Q0.Push(1);

        AssertEquals(1, Q0.Pop());
        AssertEquals(1, Q0.Pop());
        AssertEquals(2, Q0.Pop());
        AssertEquals(3, Q0.Pop());

        PriorityQueue<int> Q = new PriorityQueue<int>(1);
        Q.Push(1);
        Q.Push(3);
        Q.Push(2);
        Q.Push(1);

        AssertEquals(3, Q.Pop());
        AssertEquals(2, Q.Pop());
        AssertEquals(1, Q.Pop());
        AssertEquals(1, Q.Pop());
    }
}
```