Study the Introspective Sort, which is a default sorting algorithm implemented in the Array. Sort method. You must address the following questions:

1. What is special about the Introspective Sort?

   A： Introspective Sort combines QuickSort, HeapSort, and Insertion Sort. It starts with QuickSort, switches to HeapSort if recursion depth exceeds O(log n), and uses Insertion Sort for small subarrays for efficiency.

2. Does this method result in an unstable sorting; that is, if two elements are seen equal, might their order be not preserved after the data collection is sorted? Or does it perform a stable sorting, which preserves the order of elements that are seen equal?

   A： QuickSort and HeapSort are both unstable, so Introsort does not preserve the order of equal elements.

3. What is the time complexity of this sorting algorithm?

   A：
   Best: O(nlogn)
   Average: O(nlogn)
   Worst-case: O(nlogn)

In Task 1.1P on the Vector<T> class, you have completed/been provided with a number of methods (and properties), such as Count, Capacity, Add, IndexOf, Insert, Clear, Contains, Remove, and RemoveAt.Answer the following questions:

1. What is the time complexity of each of these operations?

A:
Count/Capacity: O(1)

Add: O(1)

IndexOf/Contains: O(n).

Insert/Remove/RemoveAt: O(n)

Clear: O(1)

2.Does your implementation match the complexity of the equivalent operations provided by the Microsoft .NET Framework for its List<T> collection?

A: Yes they are matches, because both use dynamic arrays with identical amortized complexities.

Answer and explain whether the following statements are right or wrong.

3.a. A $\theta(n^3)$ algorithm **always** takes longer to run than a $\theta(\log n)$ algorithm.

3.b. The best-case time complexity of the Bubble Sort algorithm is always $O(n)$.

3.c. The worst-case time complexity of the Insertion Sort algorithm is always $O(n^2)$.

3.d. The Selection Sort is an in-place sorting algorithm.

3.a: It's wrong, it only true for large n, not always.
3.b: It's right sorted input requires O(n) comparisons.
3.c: It's right, reversed input causes O(n^2) swaps.
3.d: It's right because it uses O(1) extra space.

3.e. The worst-case space complexity of the Insertion Sort algorithm is $O(1)$.

3.f. $2 + 2 = O(1)$

3.g. $n^3 + 10^6 n = O(n^3)$

3.h. $nlog\, n = O(n)$

3.i. $log\, n = o(n)$

3.j. $log\, n + 2^{100} = \Theta(log\, n)$

3.k. $n\, log\, n = \Omega(n)$

3.l. $\mathrm{n} + 2^n = O(n)$

3.m. $n + \frac{100n}{\log n} = o(n)$

3.n. $\mathrm{n}! = O(n)$

3.e: It's right because the worst case will be Sorts in place.
3.f: It's right because the time will be constant, and there is no other condition.
3.g: It's wrong because 10^6grows much faster, with n than n does and cannot be reduced to O(n).
3. h. It's wrong because n log(n) grows faster.
3.i. It's right because log n is getting smaller and smaller.

3.j. It's right because log n will eventually exceed any constant term as long as n is large enough.
3.k. It's right because (n log n) grows at least as fast as n.
3.l. It's wrong because it will be more relay on 2^n.
3.m. It's wrong because both n have the same increase rate.
3.n. It's wrong because(n!) grows faster than any polynomial.

Give your best- and worst-case asymptotic runtime analysis to the following code snippets.

4.1. Let flag be a random Boolean variable, whose value is either true or false.

```
int count = 0;
for (int i = 0; i < n; i++)
{
   if (flag)
   {
       for (j = i+1; i < n; j++)
       {
           count = count + i + j;
       }
   }
}
```

A: Best-case: O(n) (if flag is false).

Worst-case: O(n^2) (if flag is true).

4.2. Let random() be a function producing a real random number in the range [0,1).

```
int count = 0;
for (int i = 0; i < n; i++)
{
   int num = random();
   if( num < 0.01 )
   {
       count = count + 1;
   }
}
int num = count;
for (int j = 0; j < num; j++)
{
   count = count + j;
}
```

A: Best-case: O(n).

Worst-case: O(n).

4.3. Let Compare(x, y) be a method with the time complexity of $\Theta(1)$.

```
for (int i = 0; i < n; i++)
{
   for (int j = 0; j < n-i-1; j++)
   {
       if (a[j] > a[j+1])
       {
           Compare(a[j], a[j + 1]);
       }
   }
}
```

A: Best-case: O(n^2).

Worst-case: O(n^2).

Similar to bubble sort