9. heti feladatok

1. órai feladat Készítsen egy **OrderedItemsHandler** osztályt! Az osztály adattagként olyan elemek tömbjét tárolja el, amelyek megvalósítják az **IComparable** interfészt. A tömböt az osztály konstruktorán keresztül lehessen inicializálni, és a konstruálást követően ne lehessen az osztályon kívülről módosítani.
Az osztály publikus metódusai valósítsák meg a következő funkcionalitásokat:

1. Lehessen megvizsgálni, hogy rendezett-e a tömb a `bool IsOrdered(bool isAscending = true)` metódus használatával. Az `isAscending` „flag" értékétől függően növekvő, illetve csökkenő rendezettséget vizsgáljon a metódus.
2. Lehessen rendezni a tömböt a `void Sort(SortingMethod sortingMethod, bool isAscending = true)` metódus használatával. A `SortingMethod` egy olyan *enum* típus legyen, amely három értékkel rendelkezik: `Selection`[1], `Bubble`[2] és `Insertion`[3]. Ha a rendezéseket csak növekvő módon rendezett tömbökre kívánja implementálni, akkor érdemes készíteni egy privát `void Reverse()` metódust, amely képes megfordítani a tömbbeli elemek sorrendjét.
3. Bináris kereséssel lehessen eldönteni, hogy egy adott érték benne van-e a tömbben és ha igen, akkor adja vissza a keresett elemet. Ez a funkcionalitás is működjön növekvő, illetve csökkenő módon rendezett tömbök esetén is. Ha a tömb nem rendezett az elvárt módon, akkor a metódus dobjon el egy **NotOrderedItems** kivétel objektumot, ami tartalmazza a tömböt is.
4. A bináris keresést implementálja iteratív és rekurzív módon is!
5. Növekvően rendezett tömbök esetén lehessen megkeresni a legkisebb olyan indexet a tömbben, amely indexnél található elem értéke *nem kisebb* egy keresett értéknél. Ha a tömb nem növekvő módon rendezett, akkor dobjon el egy **NotOrderedItems** kivételt.
6. Növekvően rendezett tömbök esetén lehessen megkeresni a legkisebb olyan indexet a tömbben, amely indexnél található elem értéke *nagyobb* egy keresett értéknél. Ha a tömb nem növekvő módon rendezett, akkor dobjon el egy **NotOrderedItems** kivételt.
7. Növekvően rendezett tömbben lehessen meghatározni azon elemek darabszámát, amelyek egy adott értékkel egyenlőek.
8. Növekvően rendezett tömbben lehessen meghatározni azon elemek darabszámát, amelyek egy megadott tartományba esnek.
9. Növekvően rendezett tömbből lehessen kinyerni az összes olyan elemet, amelyek egy megadott tartományba esnek.

Minden publikus metódushoz készítsen unit teszteket, amelyekkel az összes lehetséges határeset le tudja fedni.

Készítsen egy **PhoneBookItem** osztályt, mely publikus auto-property-k segítségével eltárolja emberek nevét és telefonszámát. Az osztály valósítsa meg az **IComparable** interfészt. A `int CompareTo(object other)` metódust implementálja úgy, hogy az osztály elemeit lehessen név szerint rendezni. A metódus `other` paramétere fogadjon

[1] Minimumkiválasztásos rendezés
[2] Javított buborékrendezés
[3] Javított beillesztéses rendezés

PhoneBookItem és **string** típusú bemenetet is, egyéb esetben dobjon el **ArgumentException**-t. Összehasonlítás menete:

- ha `other` `null`, akkor **ArgumentNullException**
- ha `other` nem **PhoneBookItem** vagy **string**, akkor **ArgumentException**
- ha `other` **PhoneBookItem** típusú, akkor a `Name` tulajdonsága alapján hasonlítsa össze az adott példányban tárolt `Name` értékkel
- ha `other` **string** típusú, akkor az értéke alapján hasonlítsa össze az adott példányban tárolt `Name` értékkel

Írja felül az **Equals** metódust is oly módon, hogy a paraméterként átadott másik objektumot **PhoneBookItem** ként kezeli és párban az összes paraméterének az egyezését vizsgálja meg. Ebben a metódusban is valósítsa meg a szükséges típus vizsgálatokat.

A főprogramban készítsen egy „telefonkönyv" tömböt, amellyel példányosítson egy **OrderedItemsHandler** objektumot. Ellenőrizze le, hogy az implementált funkcionalitások helyesen működnek-e a telefonkönyv elemeivel.

2. gyakorló feladat Készítsen teljesítménymérési lehetőséget a hagyományos Lineáris keresés, Lineáris keresés rendezett tömbben, valamint a Bináris keresés algoritmusaira vonatkozóan. A teljesítmény mérés alatt az eltelt időt, az összehasonlítások számát kell elvégezni. Az egyes algoritmusok futási idejének mérésére használja a `System.Diagnostics` névtér alatt található **Stopwatch** osztályt.

Stopwatch osztály használatára példa

```
Stopwatch sw = new Stopwatch();

// Stopper indítása
sw.Start();

// Mérendő kód meghívása itt...

// Stopper megállítása
sw.Stop();

// Eltelt idő kikérése Timestamp formában TimeSpan time = sw.Elapsed;
```

- Hozzon létre egy **SearchBenchmarkResult** osztályt az alábbi paraméterek szerint
  - Legyen egy egész típusú auto property-je `Steps` névvel.
  - Legyen egy **TimeSpan** típusú auto property-je `Elapsed` névvel.
  - Legyen egy egész típusú auto property-je `SearchResultIndex` névvel.
  - Legyen felüldefiniálva a `ToString` metódusa úgy, hogy az értékek magyarázó szöveggel együtt egymás mellé legyenek fűzve.
- Hozzon létre egy **SearchStepCounter** osztályt az alábbi paraméterek szerint
  - Legyen egy egész típusú auto property-je `Steps` névvel.
  - Legyen egy `void Increase()` eljárása, mely eggyel növeli a `Steps` értékét.
- Hozzon létre egy **SearchAlgorithmBenchmark** osztályt az alábbi paraméterek szerint
  - Legyen egy **Stopwatch** típusú adattagja.

  - A stopper adattag a konstruktorban inicializálódjon.
  - Legyen egy `SearchBenchmarkResult PerformBenchmark(Func<int[], int, SearchStepCounter, int> search, int[] array, int value)` metódusa, mely elvégzi a teljesítmény mérést és az eredményét adja vissza visszatérési értékként. Minden egyes lefutáshoz példányosít egy **SearchBenchmarkResult** és egy **SearchStepCounter** osztálypéldányokat. Mivel a stopper példányszinten ugyan az, emiatt célszerű minden metódus hívás során a mérés előtt az esetleges korábbi méréseket kiüríteni a `sw.Reset()` hívással, ahol az `sw` a stopper adattagot jelöli (tetszőlegesen elnevezhető).

- A **Program** osztályban az alábbiakra lesz szüksége:
  - Készítsen egy `static int[] GenerateRandomElements(int n)` metódust, amely létrehoz egy `n` elemű egészekből álló tömböt és véletlen szerű értékekkel fel is tölti őket. A véletlen számok generálásához használja a **Random** osztály egy példányán az `r.Next()` hívást, ahol `r` a random osztály példányát hivatkozza (tetszőlegesen elnevezhető).
  - Implementáljon egy tetszőleges rendező algoritmust.
  - Példányosítson egy **SearchAlgorithmBenchmark** osztálypéldányt.
  - Generáljon egy 100000 elemű egészeket tartalmazó tömböt a `GenerateRandomElements()` függvénnyel.
  - Válasszon ki véletlenszerűen egy értéket az előzőleg generált egészek tömbjéből. Segítség: generáljon egy számot a tömb indextartományán belül.
  - Implementálja egy-egy különálló statikus metódusban a Lineáris keresés, Lineáris keresés rendezett tömbben, valamint a Bináris keresés algoritmusait.
  - Hajtsa végre a teljesítmény mérést a hagyományos lineáris kereséssel a még rendezetlen tömbön.
  - Rendezze a tömböt, majd mérje le a Lineáris keresés rendezett tömbön és a Bináris keresés implementációját.
  - Az eredményeket jelenítse meg a konzolon az alábbi módon.

```
Algorithm: Linear search
Search result idx: 16981. Steps: 16981. Elapsed time: 00:00:00.0007537.
Algorithm: Linear search in ordered array
Search result idx: 12124. Steps: 12124. Elapsed time: 00:00:00.0003888.
Algorithm: Linear search
Search result idx: 12124. Steps: 15. Elapsed time: 00:00:00.0003334.
```