Комбинаторика:

Пермутация: Всяка нередена n-торка, получена с елементите на A, като всеки елемент на А участва точно веднъж. В множеството А не се допускат повтарящи се елементи.

Пермутация с повторения: вместо множество А разглеждаме мултимножество (т.е в А е възможно да участват повтарящи се елементи)

Вариация с повторение (на n елемента от к-ти клас): к-елементен мултисписък, образуван от елементи на А( не непременно различни).

Вариация без повторение (на n елемента от к-ти клас) : к-елементен списък (неповтарящи се) елелемнти от А

Комбинация без повторение(на n елемента от к-ти клас): к-елементно подмножество на А

Комбинация s повторение(на n елемента от к-ти клас): к-елементно мултимножество, съдържащо елементи на А

**Finding subsequences**

var inputArray = inputLine.Split(new char[] {' '}, StringSplitOptions.RemoveEmptyEntries)

.Select(int.Parse).ToList();

var result = inputArray.GroupWhile((x, y) => y - x >= 1)

.ToList();

public static IEnumerable<IEnumerable<T>> GroupWhile<T>(this IEnumerable<T> seq, Func<T, T, bool> condition)

{

T prev = seq.First();

List<T> list = new List<T>() { prev };

foreach (T item in seq.Skip(1))

{

if (condition(prev, item) == false)

{

yield return list;

list = new List<T>();

}

list.Add(item);

prev = item;

}

yield return list;

}

inputArray.GroupWhile((x, y) => y - x >= 1) !!!!

y – x >= 1 -> Това е условието на сортирането.

Ако е „== 0“ тогава ще намери всички подмасиви от еднакви елементи.

Ако е „== 1“ тогава ще намери всички подмасиви от нарастващи с 1 елементи

Ако е „>= 1“ тогава ще намери всички подмасиви от нарастващи елементи

Ако е „== -1“ тогава ще намери всички подмасиви от намаляващи с 1 елементи

Ако е „<= -1“ тогава ще намери всички подмасиви от намаляващи елементи

y.Equals(x) => за подмасиви от еднакви стрингове

**Shifting left and right arrays**

var shiftsCount = shifts % inputList.Count;

if (shiftsCount < 0)

{

Console.WriteLine("Invalid input parameters.");

return;

}

if (!right)

{

var elementsToMove = inputList

.Take(shiftsCount)

.ToArray();

inputList.AddRange(elementsToMove);

inputList.RemoveRange(0, shiftsCount);

}

else

{

var elementsToMove = inputList.

Skip(inputList.Count - shiftsCount)

.Take(shiftsCount)

.ToArray();

inputList.InsertRange(0, elementsToMove);

inputList.RemoveRange(inputList.Count - shiftsCount, shiftsCount);

}

**Traverse arrays indexes left or right until count or turns**

private static void GetIndex(int offset)

{

offset = offset % numsArray.Length;

if (offset < 0)

{

offset += numsArray.Length;

}

currentIndex = (currentIndex + offset) % numsArray.Length;

}