Практическая работа № 4

Теоретическая часть

Сортировка массива методом Хоара

Быстрая сортировка, **сортировка Хоара** (англ. *quicksort*), часто называемая ***qsort*** (по имени в стандартной библиотеки Си) — алгоритм сортировки, разработанный английским информатиком Тони Хоаром во время его работы в МГУ в 1960 году. Один из самых быстрых известных универсальных алгоритмов сортировки массивов.

Алгоритм состоит из трёх шагов:

1. Выбрать элемент из массива. Назовём его опорным.
2. *Разбиение*: перераспределение элементов в массиве таким образом, что элементы, меньшие опорного, помещаются перед ним, а большие или равные — после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

Работа с двумерными массивами:

Объявление двумерного массивами

```
int[,] nums1;
int[,] nums2 = new int[2, 3];
int[,] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums4 = new int[,] { { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums5 = new [,]{ { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Перебор двумерного массивами

```
for (int i = 0; i < rows; i++)
{
    for (int j = 0; j < columns; j++)
    {
        Console.Write($"{numbers[i, j]} \t");
    }
    Console.WriteLine();
}
```

Практическая часть

1. Реализация сортировки массива

1.1 Для начала создайте консольное приложение, далее объявите целочисленный массив (количество элементов массива вводит сам пользователь), массив можно заполнить либо через ручной ввод данных, либо через рандом. Пример заполнения массива случайными числами представлен ниже:

```
// Размер массива
int size = 10;

// Создание массива
int[] array = new int[size];

// Создание генератора случайных чисел
Random random = new Random();

// Заполнение массива случайными числами в диапазоне от 0 до 100
for (int i = 0; i < array.Length; i++)
{
    array[i] = random.Next(0, 101); // Случайные числа от 0 до 100
}
```

1.2 Объявите метод **QuickSort**, который будет принимать в качестве параметров сам массив, а также значения индексов первого элемента массива и последнего (назовите из left и right, тип данных int).

1.3 Далее следует проверка, если индекс первого элемента меньше индекса последнего элемента, то выполняем следующее: разделяем массив и получаем индекс опорного элемента.

```
int pivotIndex = Partition(array, left, right);
```

1.4 Выполняем рекурсию нашего метода **QuickSort,** то есть вызываем метод внутри самого себя. При задании аргументов учтите, аргументы **array** и **left** оставляем прежними, а вот вместо аргумента **right** прописываем индекс опорного элемента (переменная **pivotIndex**), уменьшая его на 1.

1.5 Снова прописываем рекурсию метода **QuickSort,** только тут уже меняем аргумент **left**, и вместо него прописываем увеличенный на 1 индекс опорного элемента.

1.6 Для пояснения: **весь код метода QuickSort прописывается внутри условия из пункта 1.3, конструкции else/ if else здесь не предусмотрены.**

1.7 На рисунке пункта 1.3 был вызов метода **Partition,** но объявления самого метода не было, необходимо его прописать. Метод **Partition** возвращает **int,** в качестве аргументов он принимает все то же самое, что и метод **QuickSort**

1.8 Далее необходимо задать вспомогательные переменные внутри метода **Partition:**

```
int pivot = array[left];
int i = left;
int j = right;
```

1.9 Прописываем цикл **while,** который будет идти до тех пор, пока индекс левого элемента массива не станет больше, чем индекс правого элемента массива (i > j)

1.10 Далее ищем элемент слева, который больше или равен опорному элементу массива.

```
while (array[i] <= pivot && i < right)
{
    i++;
}
```

1.11 Теперь ищем элемент справа, который меньше или равен опорному элементу массива

```
while (array[j] > pivot)
{
    j--;
}
```

1.12 Если значение переменной **i** меньше значения переменной **j**, то меняем местами значения элементов массива под индексами **i** и **j** (**array[i]** и **array[j]**). Цикл, который мы объявили в пункте 1.9, заканчивается.

1.13 Меняем местами опорный элемент массива и тот, индекс которого равен **j** (**array[left]** и **array[j]**).

1.14 Метод Partition возвращает значение переменной **j**.

1.15 Выведите отсортированный массив на экран.

2. Решение задач.

2.1 Имеются две ёмкости: кубическая с ребром A, цилиндрическая с высотой H и радиусом основания R. Определить можно ли заполнить жидкостью объёма M первую ёмкость, вторую, обе.

2.2 Дан одномерный массив числовых значений, насчитывающий N элементов. Поменять местами первую и вторую половины массива.

2.3 Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Найти наибольшее значение среди средних значений для каждой строки матрицы.