Лабораторная работа № 28-29

«Оценка сложности и оформление алгоритмов выбора из массива»

Цель работы:
изучить оценку сложности и оформление алгоритмов выбора из массива.

Теория

Алгоритм выбора — это алгоритм для нахождения k-го по величине элемента в массиве (такой элемент называется k-й порядковой статистикой). Частными случаями этого алгоритма являются нахождение минимального элемента, максимального элемента и медианы. Существует алгоритм, который гарантированно решает задачу выбора k-го по величине элемента за O(n).

В чём идея сортировок выбором?

1. В неотсортированном подмассиве ищется локальный максимум (минимум).

2. Найденный максимум (минимум) меняется местами с последним (первым) элементом в подмассиве.

3. Если в массиве остались неотсортированные подмассивы

Виды сортировок выбором:
Сортировка выбором :: Selection sort

Просто и незатейливо — проходим по массиву в поисках максимального элемента. Найденный максимум меняем местами с последним элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает последний элемент, куда мы переставили найденный максимум). К этой неотсортированной части применяем те же действия — находим максимум и ставим его на последнее место в неотсортированной части
массива. И так продолжаем до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента.

```
def selection(data):
for i, e in enumerate(data):
mn = min(range(i, len(data)), key=data.__getitem__)
data[i], data[mn] = data[mn], e
return data
```

Двухсторонняя сортировка выбором :: Double selection sort

Похожая идея используется в шейкерной сортировке, которая является вариантом пузырьковой сортировки. Проходя по неотсортированной части массива, мы кроме максимума также попутно находим и минимум. Минимум ставим на первое место, максимум на последнее. Таким образом, неотсортированная часть при каждой итерации уменьшается сразу на два элемента.

На первый взгляд кажется, что это ускоряет алгоритм в 2 раза — после каждого прохода неотсортированный подмассив уменьшается не с одной, а сразу с двух сторон. Но при этом в 2 раза увеличилось количество сравнений, а число свопов осталось неизменным. Двойной выбор лишь незначительно увеличивает скорость алгоритма, а на некоторых языках даже почему-то работает медленнее.

Пример:

Сортировка методом выбора проходит следующим образом.
К примеру, у нас имеется массив 3 9 1 4 0
Проводим первую итерацию. Ищем минимальный элемент массива. Для этого берём число на первой позиции, условно обозначаем его как минимальное и сравниваем с остальными. Если найдётся число, которое окажется меньше, чем нынешнее, то такое число обозначается минимальным, а затем также сравнивается с последующими. Таким образом, после прохода по всему массиву, мы найдём элемент с самым маленьким числовым значением.

Затем меняем местами найденное минимальное число с элементом на нулевой позиции в массиве и “выкидываем” эту нулевую позицию из процесса сортировки.

Жирным шрифтом будет обозначаться нынешний минимальный элемент.

3 9 1 4 0

Нынешний минимальный элемент сравнивается с 9. Он меньше, чем 9, поэтому сравнивается со следующим элементом. 1 меньше, чем 3, следовательно, теперь 1 – это наш нынешний минимальный элемент.

3 9 **1** 4 0

Продолжаем сравнение. 1 меньше 4, но 0 меньше 1, значит теперь он считается минимальным элементом.

3 9 1 4 **0**

Мы прошли по массиву и нашли самый маленький его элемент.

Теперь мы меняем местами этот элемент с элементом на нулевой позиции и больше не сравниваем его ни с чем (так как это бессмысленно и только занимает лишнее время), уменьшая количество шагов в итерации на 1.

0 9 1 4 3

Вторая итерация. Принимаем первый элемент среди оставшихся неупорядоченных чисел за минимальный и сравниваем его с остальными.

0 9 1 4 3

Единица меньше 9, теперь она минимальная.

0 9 1 4 3

Единица сравнивается с 4 и с 3, но все эти числа больше неё, следовательно, это и есть наш следующий минимальный элемент.

Мы меняем его местами с элементом на первой позиции массива и опять сокращаем количество шагов в итерации на 1.

0 1 9 4 3

Третья итерация. Таким же образом ищем следующее минимальное число.

0 1 9 4 3

Четвёрка меньше девятки.

0 1 9 4 3

Тройка меньше четвёрки. Мы обошли весь массив, значит 3 и является минимальными значением. Меняем его местами с элементном на второй позиции массива.

0 1 3 4 9

Четвёртая итерация. Сравниваем оставшиеся числа. Нынешнее минимальное число 4 меньше, чем 9, значит оно остаётся на своём месте, а так как у нас после него остаётся последнее число 9 в массиве, то оно, соответственно, является максимальным из всего массива. Сортировка окончена.

Таким образом у нас получается упорядоченный методом выбора массив:

0 1 3 4 9

В программе для начала создаём функцию сортировки.

```
static int[] ViborSort(int[] mas)
{
    for (int i = 0; i < mas.Length - 1; i++)
    {
        //поиск минимального числа
        int min=i;
        for (int j = i + 1; j < mas.Length; j++)
        {
            if (mas[j] < mas[min])
            {
                min = j;
            }
        }
```

```
        //обмен элементов
        int temp = mas[min];
        mas[min] = mas[i];
        mas[i] = temp;
    }
    return mas;
}
```

В строках 7-14 мы ищем минимальный элемент массива. Происходит это так же, как описано в примере.

В строках 16-18 мы меняем местами элементы. В переменную temp записываем минимальный элемент массива. Затем на позицию минимального элемента записываем элемент, на чьё место он должен встать (на первой итерации – нулевая позиция, на второй итерации – первая, и так далее; это контролируется условием в строке 4), а затем, на эту самую позицию, которую мы освободили, вставляем значение из переменной temp.

В главную функцию main мы веорнём уже упорядоченный массив (строка 20).

Теперь рассмотрим, что у нас будет в функции main.

```
static void Main(string[] args)
{
    Console.WriteLine("Ведите количество чисел для сортировки.");
    int N = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Введите числа для сортировки:");
    int[] mas = new int[N];
    for (int i = 0; i < mas.Length; i++)
    {
        mas[i] = Convert.ToInt32(Console.ReadLine());
    }
    ViborSort(mas);
    Console.WriteLine("Отсортированный массив:");
    for (int i = 0; i < mas.Length; i++)
    {
        Console.WriteLine( mas[i]);
    }
    Console.ReadLine();
}
```

В строках 3-10 мы просим пользователя ввести количество элементов массива и сами эти значения, и считываем их.

Затем вызываем функцию ViborSort(mas), куда передаём заполненный массив. Там он упорядочивается и возвращается. Нам остаётся только вывести его в в консоль, что мы и делаем в строках 12-17.

Ход работы:

1. Написать программу, которая сортирует массив по возрастанию.
2. Написать программу, которая сортирует массив по убыванию

```
static void Main(string[] args)
{
    onChoice();

    void onChoice()
    {

        int[] sortingArr = createArr();
        Console.WriteLine("\nВыбирете действие:");
        Console.WriteLine("0: Сортировка по возрастанию");
        Console.WriteLine("1: Сортировка по убыванию");
        Console.WriteLine("2: выход");
        Console.Write("> ");
        try
        {
            int choice = Convert.ToInt32(Console.ReadLine());


            switch (choice)
            {
                case 0:
                    sortArr(sortingArr, true);
                    break;
                case 1:
                    sortArr(sortingArr, false);
                    break;
                case 2:
                    System.Environment.Exit(0);
                    break;
                default:
                    Console.WriteLine(); onChoice();
                    break;
            }
        }
        catch
        {
            Console.WriteLine("\nВведите корректное значение\n");
        }
    }

    int[] sortArr(int[] arr, bool increase)
    {
```

```
        Array.Sort(arr);
        if (!increase) Array.Reverse(arr);
        showArr(arr);
        return arr;
    }

    int[] createArr()
    {
        Random rnd = new Random();

        Console.Write("\nВведите кол-во эллементов массива ");
        Console.Write("> ");
        int[] array = new int[Convert.ToInt32(Console.ReadLine())];

        for (int i = 0; i < array.Length; i++)
        {
            array[i] = rnd.Next(-10, 10);
        }

        showArr(array);

        return array;
    }

    void showArr(int[] arr)
    {
        for (int i = 0; i < arr.Length; i++)
        {
            Console.Write(arr[i] + " ");
        }
    }

    while (true)
    {
        onChoice();
    }
}
```

```
Введите кол-во эллементов массива > 5
4 9 -5 2 9
Выбирете действие:
0: Сортировка по возрастанию
1: Сортировка по убыванию
2: выход
> 1
9 9 4 2 -5
Введите кол-во эллементов массива > 6
-3 -6 -6 4 7 1
Выбирете действие:
0: Сортировка по возрастанию
1: Сортировка по убыванию
2: выход
> 0
-6 -6 -3 1 4 7
Введите кол-во эллементов массива >
```

Контрольные вопросы:

1. Что такое алгоритм выбора?
   алгоритм для нахождения k-го по величине элемента в массиве (такой элемент называется k-й порядковой статистикой).
2. Идея сортировок выбором?

- В неотсортированном подмассиве ищется локальный максимум (минимум).
- Найденный максимум (минимум) меняется местами с последним (первым) элементом в подмассиве.
- Если в массиве остались неотсортированные подмассивы

3. В чём смысл сортировка выбором :: Selection sort?
   Просто и незатейливо — проходим по массиву в поисках максимального элемента. Найденный максимум меняем местами с последним элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает последний элемент, куда мы переставили найденный максимум). К этой неотсортированной части применяем те же действия — находим максимум и ставим его на последнее место в неотсортированной части массива. И так продолжаем до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента.
4. В чём смысл Двухсторонняя сортировка выбором :: Double selection sort?
   Похожая идея используется в шейкерной сортировке, которая является вариантом пузырьковой сортировки. Проходя по неотсортированной части массива, мы кроме максимума также попутно находим и минимум. Минимум ставим на первое место, максимум на последнее. Таким образом, неотсортированная часть при каждой итерации уменьшается сразу на два элемента.