**Лабораторная работа № 28-29**

**«Оценка сложности и оформление алгоритмов выбора из массива»**

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

**Краткие теоретические и учебно-методические материалы   
по теме лабораторной работы**

**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** 94 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | 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. В консоли наша сортировка методом выбора будет выглядеть вот так:

**Задания для лабораторной работы:**

Написать программу, которая сортирует выбором из массива по возрастанию

using System;

class Program

{

static void Main()

{

int[] array = { 64, 25, 12, 22, 11 };

Console.WriteLine("Исходный массив:");

PrintArray(array);

SelectionSortAscending(array);

Console.WriteLine("Отсортированный массив (по возрастанию):");

PrintArray(array);

}

static void SelectionSortAscending(int[] arr)

{

int n = arr.Length;

for (int i = 0; i < n - 1; i++)

{

int minIndex = i;

for (int j = i + 1; j < n; j++)

{

if (arr[j] < arr[minIndex])

{

minIndex = j;

}

}

if (minIndex != i)

{

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

}

static void PrintArray(int[] arr)

{

foreach (var item in arr)

{

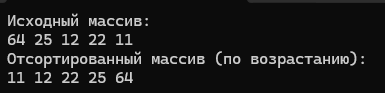
Console.Write(item + " ");

}

Console.WriteLine();

}

}



Написать программу, которая сортирует выбором из массива по убыванию

class Program

{

static void Main()

{

int[] array = { 64, 25, 12, 22, 11 };

Console.WriteLine("Исходный массив:");

PrintArray(array);

SelectionSortDescending(array);

Console.WriteLine("Отсортированный массив (по убыванию):");

PrintArray(array);

}

static void SelectionSortDescending(int[] arr)

{

int n = arr.Length;

for (int i = 0; i < n - 1; i++)

{

int maxIndex = i;

for (int j = i + 1; j < n; j++)

{

if (arr[j] > arr[maxIndex])

{

maxIndex = j;

}

}

if (maxIndex != i)

{

int temp = arr[i];

arr[i] = arr[maxIndex];

arr[maxIndex] = temp;

}

}

}

static void PrintArray(int[] arr)

{

foreach (var item in arr)

{

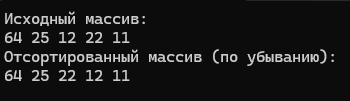
Console.Write(item + " ");

}

Console.WriteLine();

}

}



Контрольные вопросы:

1. **Что такое алгоритм выбора?**

Алгоритм выбора — это алгоритм для нахождения k-го по величине элемента в массиве (такой элемент называется k-й порядковой статистикой).

1. **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

1. **В чём смысл сортировка выбором :: Selection sort?**

Найденный максимум меняем местами с последним элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает последний элемент, куда мы переставили найденный максимум).

1. **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.

**Как выглядит блок схема сортировки выбором?**

**[Начало]**

**|**

**v**

**[Установить i = 0]**

**|**

**v**

**[Проверить i < n - 1?] -- Нет --> [Конец]**

**|**

**Да**

**|**

**v**

**[Установить minIndex = i]**

**|**

**v**

**[Установить j = i + 1]**

**|**

**v**

**[Проверить j < n?] -- Нет --> [Обмен значениями]**

**|**

**Да**

**|**

**v**

**[Проверить arr[j] < arr[minIndex]?]**

**|**

**Да Нет**

**| |**

**v v**

**[Установить minIndex = j]**

**|**

**v**

**[Увеличить j на 1]**

**|**

**v**

**[Вернуться к проверке j < n?]**

**Вывод:** изучил оценку сложности и оформление алгоритмов выбора из массива.