Лабораторная работа №6

Алгоритмы сортировки

Разработка простых алгоритмов сортировки одномерных и многомерных массивов. Разработка, отладка и выполнение программы с использованием простых алгоритмов сортировки.

Краткие теоретические сведения:

Прежде чем приступить к изучению и разработке простых алгоритмов сортировки, необходимо изучить методы сравнения быстродействия различных алгоритмов.

Как определить сложность массива? Это можно сделать, проанализировав вложенные циклы. Каждый вложенный цикл умножает сложность алгоритма на n. Например, если у нас есть алгоритм вида:

```
std::cin >> n;
for(int i = 0; i < n; i++)
{
	for(int j = 0; j < n; j++)
		{ //do something; }
}
```

То его сложность будет n * n = n^2, т.е. он работает за O(n^2)

При этом важно понимать, что если алгоритм состоит из нескольких независимых частей, то имеет значение только компонент
с наибольшей асимптотической сложностью (в первом столбце таблицы сложность алгоритмов расставлена по возрастанию)

Пример:

```
std::cin >> n;
for(int i = 0; i < n; i++)
{
	for(int j = 0; j < n; j++)
		{ //do something; }
}
for(int i = 0; i < n; i++)
{
	//do something;
}
```

Этот алгоритм состоит из двух частей: первый цикл работает за O(n^2), второй цикл - за O(n). Суммарная сложность будет O(n^2 + n). Но n^2 асимптотически сложнее, чем n, т.е. квадратичная функция стремится к бесконечности быстрее, чем линейная. При n = 500, первый цикл, работающий за O(n^2) выполнит 250 000 операций, а второй - всего 500. И при росте размера входных данных разница будет заметна всё сильнее.

Но из произведения сложностей, очевидно, удалить меньшую компоненту нельзя: O(n*logn) нельзя сократить до O(n)

Сравнение сложности

Оценка	N=5	N=10	N=25	N=100	N=500
$O(1)$	1	1	1	1	1
$O(logN)$	3	4	5	7	9
$O(N)$	5	10	25	100	500
$O(N \times logN)$	15	40	125	700	4500
$O(N^2)$	25	100	625	10 000	250 000
$O(N^3)$	125	1 000	15 625	1 000 000	125 000 000
$O(2^N)$	32	1 024	33 554 432	очень много	очень много
$O(N!)$	120	3 628 800	очень много	очень много	очень много

Виды простых сортировок:

1) Сортировка пузырьком

```
const int size = 4;
   int arr[size] = {7, 2, 5, 1};
   for (int i = 0; i < size - 1; i++) {
      for (int j = 0; j < size - i - 1; j++) {
         if (arr[j] > arr[j + 1]) {
            int temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
      }
   }
```

Время выполнения нашего алгоритма не зависит от вида входящего массива, в худшем, среднем и лучшем (когда массив изначально отсортирован) он отработает за O(n^2), поскольку в нём нет механизмов проверки на отсортированность.

2) Сортировка вставками

```
const int size = 4;
   int arr[size] = { 7, 2, 5, 1 };
   for (int i = 1; i < size; i++) {
      int current = arr[i];
      for (int j = i - 1; j >= -1; j--) {
         if (j >= 0 && arr[j] > current) {
```

```
            arr[j + 1] = arr[j];
        }
        else {
            arr[j+1] = current;
            break;
    } } }
```

Всего алгоритм должен пройтись по всему массиву (n-1 операция) и каждый элемент массива сравнить с несколькими элементами отсортированной части. В худшем случае (для этого алгоритма это ситуация, когда элементы расположены в обратном порядке, т.е. когда массив {4, 3, 2, 1} нужно отсортировать по возрастанию) каждый элемент массива придётся сравнить со всеми элементами из уже отсортированной части. Тогда сложность алгоритма составит O(n^2 / 2), т.е. O(n^2). В среднем случае сложность будет такой же - O(n^2). А в лучшем случае, когда массив уже отсортирован, сложность составит всего O(n), т.е. алгоритм отработает за линейное время, поскольку нужное положение каждого элемента в отсортированной части будет определяться за O(1).

3) Сортировка выбором

```
const int size = 10;
    int arr[size] = { 8, 5, 2, 6, 9, 3, 1, 4, 0, 7 };
    for (int i = 0; i < size - 1; i++) {
        int min = arr[i];
        int min_pos = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < min) {
                min = arr[j];
                min_pos = j;
            }
        }
        if (min_pos != i) {
            int temp = arr[i];
            arr[i] = arr[min_pos];
            arr[min_pos] = temp;
        }
    }
```

На каждой итерации алгоритма ставятся в нужное положение сразу два элемента: минимальный из просмотренных и максимальный
из просмотренных, т.е. всего необходимо совершить size / 2 итераций по массиву.

Улучшенная сортировка пузырьком для многомерных массивов:

```
#include <iostream>
```

```
int main() {
    const int row = 4;
    const int column = 4;
    int size = row * column;
    int arr[row][column] = { {7, 2, 5, 1}, {3, 16, 6, 13}, {12, 4, 15, 10}, {9, 14, 8, 11}
};
    for (int i = 0; i < size / 2; i++) {
        int min = arr[i / row][i % column];
        int min_pos = i;
          for (int j = i; j < size - i - 1; j++) {
          //ищем минимальный элемент
                    if (arr[j / row][j % column] < min)
                    {
                        min = arr[j / row][j % column];
                        min_pos = j;
                    }
                    if (arr[j / row][j % column] > arr[(j + 1) / row][(j + 1) % column]) {
                        int temp = arr[j / row][j % column];
                        arr[j / row][j % column] = arr[(j + 1) / row][(j + 1) % column];
                        arr[(j + 1) / row][(j + 1) % column] = temp;
                   // поскольку элементы обменялись местами, необходимо
                   проверить, не
                                // является ли меньший из них текущим минимумом
                                if (arr[j / row][j % column] < min)
                                {
                                    min = arr[j / row][j % column];
                                    min_pos = j;
                                }
                            }
                        }

        if (min_pos != i)
                {
                    int temp = arr[i / row][i % column];
                    arr[i / row][i % column] = arr[min_pos / row][min_pos % column];
                    arr[min_pos / row][min_pos % column] = temp;
                }
            }
            //вывод отсортированного массива
            for (int i = 0; i < row; i++)
            {
```

```
            for (int j = 0; j < column; j++)
                std::cout << arr[i][j] << ' ';
            std::cout << '\n';
    }
}
```

Задание 1

На вход подаётся массив целых чисел. Реализуйте сортировку пузырьком в обратном порядке.

Задание 2

На вход подаётся массив дробных чисел. Первую половину массива отсортировать сортировкой вставками, вторую половину сортировку выбором.

Задание 3

На вход подается двумерный динамической массив размером n*k, где n,k<=10^3. Каждый элемент массива по модулю меньше либо равен 2^31. Требуется отсортировать порядок строк по возрастанию, где сумма элементов строки увеличивается, т.е. сумма элементов в первой строке минимальна, а в последней-максимальна.