Selection Sort

Алгоритм сортировки выбором, идея которого заключается в поиске минимального значения для каждой итерации и перестановки первого элемента текущей итерации с найденным минимальным элементом.

Selection Sort

```
void selection_sort(vector<int> & v) {
    for(int i = 0; i < v.size()-1; ++i)     // перемещаемся по массиву вправо
    {
        int min = i;                        // считаем все что слева i отсортированным
        for(int j = i+1; j < v.size(); ++j) // ищем наименьший в правой части массива
        {
            if(v[j] < v[min])
                min = j;                    // если нашли - запоминаем минимальный
        }

        if(v[j] < v[i])
            swap(v[min], v[i]);             // меняем местами минимальный и текущий
    }
}
```

godbolt.org

Selection Sort

Сравнений: (N-1)+(N-2)+…1+0 ~$N^2/2$

Перестановок: N

Сложность: O(N^2)

Время работы алгоритма не зависит от порядка расположения исходных данных. В общем случае – квадратичное, даже если исходный массив отсортирован.

Плюсы: Минимальное количество перестановок.

Минусы: Очень высокая вычислительная сложность O(N^2)

Insertion Sort

Алгоритм сортировки вставками, идея которого заключается в замене *i-го* элемента на каждой итерации с каждым бОльшим элементом слева от него

Insertion Sort

```
void insertion_sort(vector<int> & v) {
    for(int i = 0; i < v.size(); ++i)        // перемещаемся по массиву вправо
    {
        for(int j = i-1; j < 0; --j)         // находим место для вставки текущего
        {                                    // элемента в отсортированной левой части
            if(v[j] < v[j-1])
                swap(v[j], v[j-1]);          // если нашли элемент меньше текущего - меняем
        }
    }
}
```

godbolt.org

Insertion Sort

Сравнений: *~1/4* N^2

Перестановок: *~1/4* N^2

Сложность: $O(N^2)$

Время работы алгоритма зависит от состояния массива.

- В полностью отсортированном массиве алгоритм выполняет N-1 сравнение и 0 перестановок.

- В реверсивно-отсортированном массиве алгоритм выполняет *~1/2* N^2 сравнений и *~1/2* N^2 перестановок

Плюсы: Эффективность на малых наборах данных (1-~100 элементов).

Минусы: Очень высокая вычислительная сложность $O(N^2)$

Bubble Sort

Алгоритм «пузырьковой» сортировки, идея которого заключается в повторяющихся проходах по массиву, сравнивая за каждый проход попарно все элементы и выполняя обмен между ними, в случае если порядок не верный.

7	10	5	3	8	4	2	9	6	Swap (6,9)
7	10	5	3	8	4	2	6	9	Swap (2,4)
7	10	5	3	8	2	4	6	9	Swap (2,8)
			...						
2	7	10	5	3	8	4	6	9	Конец 1-го прохода
2	3	7	10	5	4	8	6	9	Конец 2-го прохода
			...						

Bubble Sort

```
void bubble_sort(vector<int> & v) {
    for (int i = 0; i < v.size()-1; ++i) {
        for (int j = 0; j < v.size() - 1; ++j) {
            /* если встречаем элемент больше чем следующий.
             * то меняем их местами
             **/
            if (v[j] > v[j + 1])
                swap(v[j], v[j+1]);
        }
    }
}
```

godbolt.org

Bubble Sort

Сравнений: *(N-1) * N*

Перестановок: *(N-1)*N/2*

Сложность: $O(N^2)$

Время работы алгоритма зависит от состояния массива.

- В полностью отсортированном массиве алгоритм выполняет *N*(N-1)* сравнение и 0 перестановок.

- В реверсивно-отсортированном массиве алгоритм выполняет *N*(N-1)* сравнений и *(N-1)*N/2* перестановок

Плюсы:

Минусы: Очень высокая вычислительная сложность $O(N^2)$ вне зависимости от расположения данных