Алгоритмы сортировки массива: классификация. Примеры алгоритмов сортировки за полиномиальное время.

Алгоритмы сортировки массива

Сортировка — это переупорядочивание заданного массива или списка элементов в определённом порядке, по возрастанию, по убыванию или в любом другом порядке, заданном пользователем, например, сортировка строк по длине. Алгоритмы сортировки широко используются в поиске, базах данных, стратегиях «разделяй и властвуй» и структурах данных.

Основные области применения:
-Организация больших наборов данных для упрощения обработки и печати
-Быстрый доступ к k-му наименьшему или наибольшему элементу
-Возможность бинарного поиска для быстрого поиска в отсортированных данных
-Решение сложных задач как в программном обеспечении, так и в разработке алгоритмов

Сортировка предусмотрена в библиотечных реализациях большинства языков программирования. Эти функции сортировки, как правило, являются функциями общего назначения, обладающими гибкостью задания ожидаемого порядка сортировки (по возрастанию, по убыванию или по определённому ключу в случае объектов).

Алгоритмы также могут различаться в зависимости от требований к выходным данным. Например, сортировка может быть устойчивой (сохранение исходного порядка одинаковых элементов) или нестабильной.

Классификация алгоритмов сортировки массива

1. По потребности в дополнительной памяти: ***«Not In-place» и «In-place» алгоритмы.***

У термина «in-place» есть несколько определений. *Одно строгое определение:*

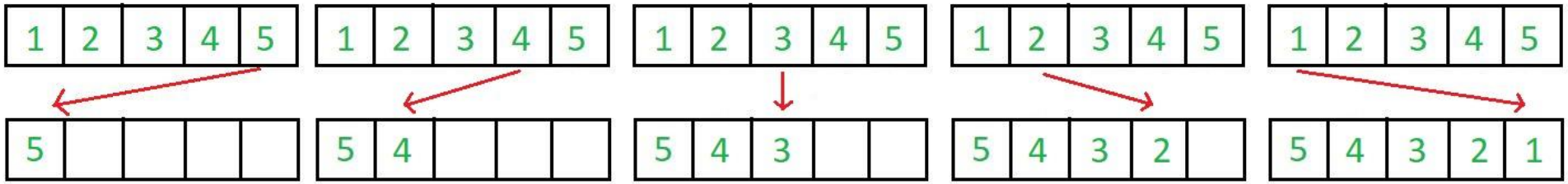
Алгоритм «in-place» — это алгоритм, которому не требуется дополнительная память и который создаёт выходные данные в той же памяти, что и входные данные, преобразуя их «на месте». Однако допускается небольшое постоянное дополнительное пространство для переменных.

Более широкое определение:

«in-place» означает, что алгоритм не использует дополнительную память для обработки входных данных, но может потребовать небольшое, хотя и непостоянное, дополнительное пространство для своей работы. Обычно это пространство составляет O(log n), хотя иногда допускается любое значение в O(n) (меньшее, чем линейное).

Классификация алгоритмов сортировки массива

«Not In-place» - алгоритм.



Для этого алгоритма требуется O(n) дополнительного места.

Сортировка по цепочкам (Strand Sort).

«In-place» - алгоритм.

Для обмена элементами требуется O(1) дополнительного места.

Пузырьковая сортировка (Bubble sort), сортировка выбором (Selection Sort), сортировка вставками (Insertion Sort)

Классификация алгоритмов сортировки массива

2. По сфере применения: Внутренняя и внешняя сортировки.

Внутренняя сортировка — это когда все данные (оперирует массивами) помещаются в оперативную или внутреннюю память. При внутренней сортировке задача не может принимать входные данные, превышающие размер выделенной памяти.

Внешняя сортировка — это термин, обозначающий класс алгоритмов сортировки, способных обрабатывать большие объёмы данных (оперирует запоминающими устройствами большого объёма). Внешняя сортировка необходима, когда сортируемые данные не помещаются в основную память вычислительного устройства (обычно ОЗУ) и должны находиться в более медленной внешней памяти (обычно на жёстком диске). Сортировка не с произвольным доступом, а последовательным (упорядочение файлов), то есть в данный момент «виден» только один элемент.

Примеры внешней сортировки:

1. Сортировка слиянием (Merge sort)
2. Ленточная сортировка (Tape sort)
3. Полифазная сортировка (Polyphase sort)

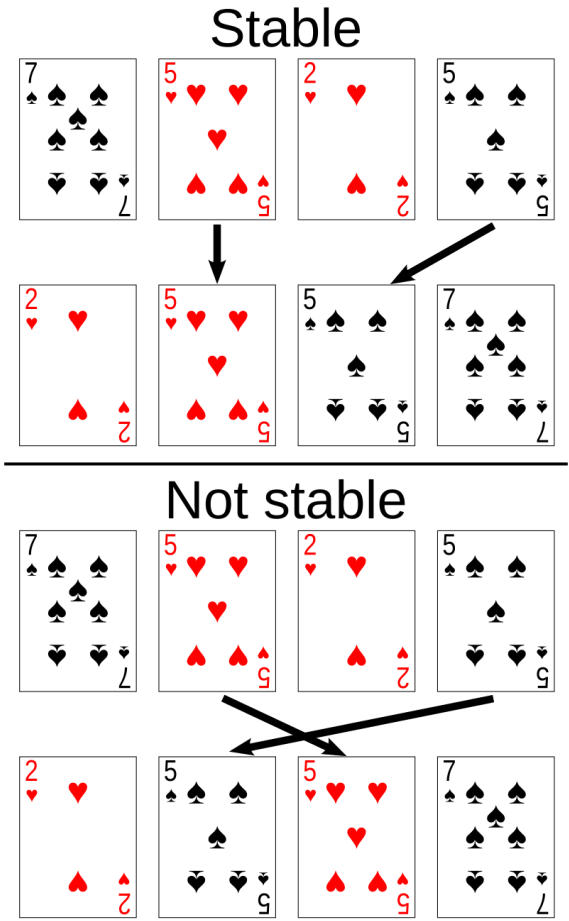
Классификация алгоритмов сортировки массива

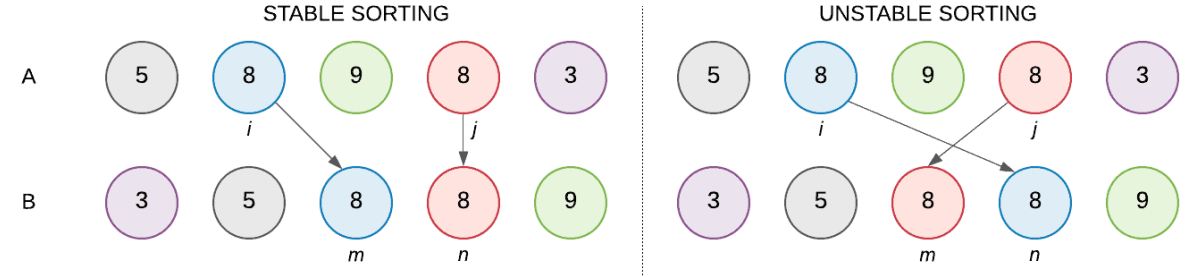
3. По устойчивости: Устойчивые и неустойчивые алгоритмы.

Устойчивая (стабильная) сортировка — сортировка, которая не меняет относительный порядок сортируемых элементов, имеющих одинаковые ключи, по которым происходит сортировка. Устойчивость важна, когда есть пары «ключ-значение» с возможными дублирующимися ключами (например, имена людей в качестве ключей и их данные в качестве значений). И необходимо отсортировать эти объекты по ключам.

Устойчивые алгоритмы: Пузырьковая сортировка (Bubble Sort), сортировка вставками (Insertion Sort), сортировка подсчётом (Count Sort).

Неустойчивые алгоритмы: Сортировка выбором (Selection Sort), быстрая сортировка (Quick Sort), пирамидальная сортировка (Heap Sort).



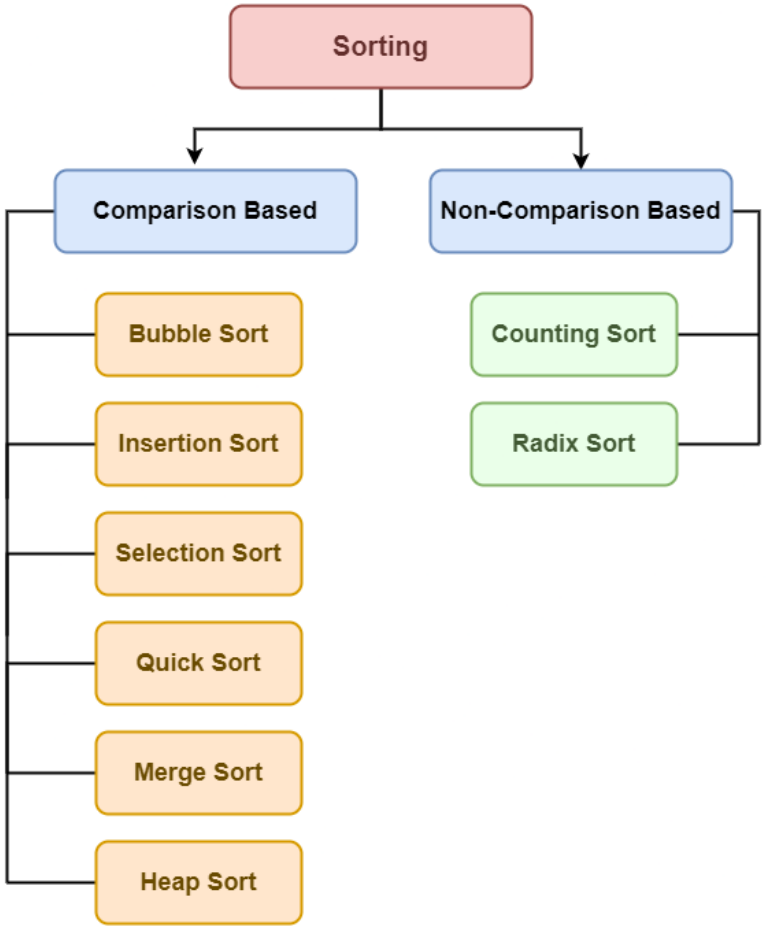


Классификация алгоритмов сортировки массива

4. По использованию операции сравнения.

В структурах данных используются различные алгоритмы сортировки. Алгоритмы сортировки можно разделить на два основных типа:

1. **Основанные на сравнении:** в алгоритме сортировки, основанном на сравнении, элементы сравниваются.
2. **Не основанные на сравнении:** в алгоритме сортировки, не основанном на сравнении, элементы не сравниваются.



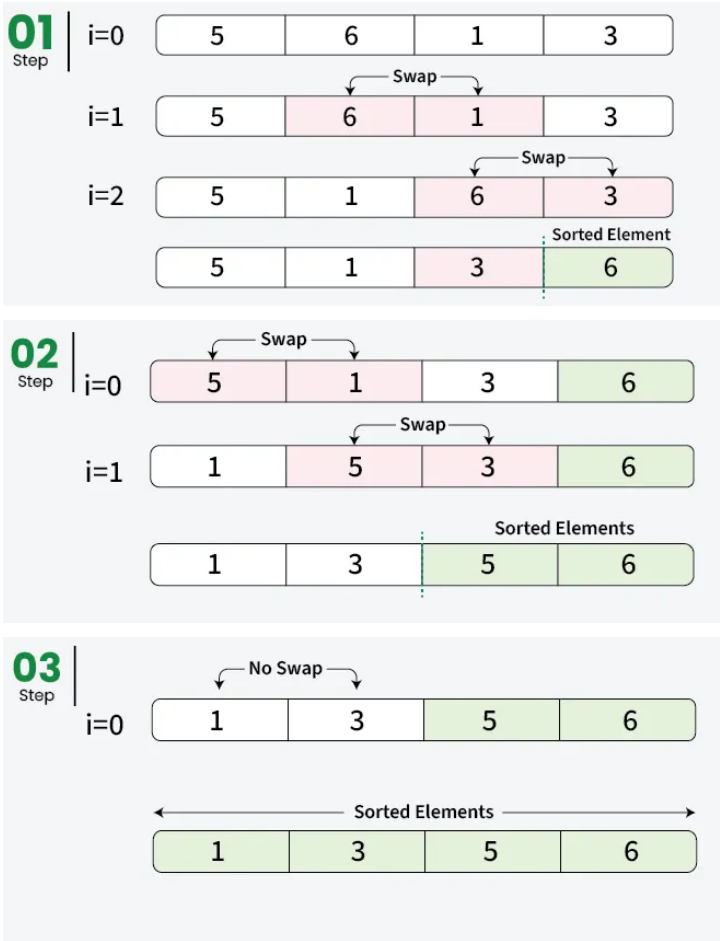
Пузырьковая сортировка (Bubble Sort)

Это простой алгоритм сортировки, который многократно меняет местами соседние элементы, если они расположены в неправильном порядке. Он выполняет несколько проходов по массиву, и в каждом проходе наибольший неотсортированный элемент перемещается на свою правильную позицию в конце.

-После первого прохода максимальный элемент перемещается в конец (на свою правильную позицию). Аналогично, после второго прохода второй по величине элемент перемещается на предпоследнюю позицию.

-В каждом проходе мы обрабатываем только те элементы, которые ещё не были перемещены в правильную позицию. После k проходов наибольшие k элементов должны быть перемещены на последние k позиций.

-В проходе мы рассматриваем оставшиеся элементы, сравниваем все смежные элементы и меняем местами, если больший элемент находится перед меньшим. Продолжая так, мы получаем наибольший элемент (из оставшихся) на своей правильной позиции.



Преимущества:

- Пузырьковая сортировка проста в понимании и реализации.
- «in-place» алгоритм - не требует дополнительного объема памяти.
- Стабильный алгоритм сортировки, что означает, что элементы с одинаковым значением ключа сохраняют свой относительный порядок в отсортированном выводе.

Недостатки:

- Временная сложность пузырьковой сортировки составляет $O(n^2)$.
- Пузырьковая сортировка имеет ограниченное применение на практике.

Временная сложность: $O(n^2)$

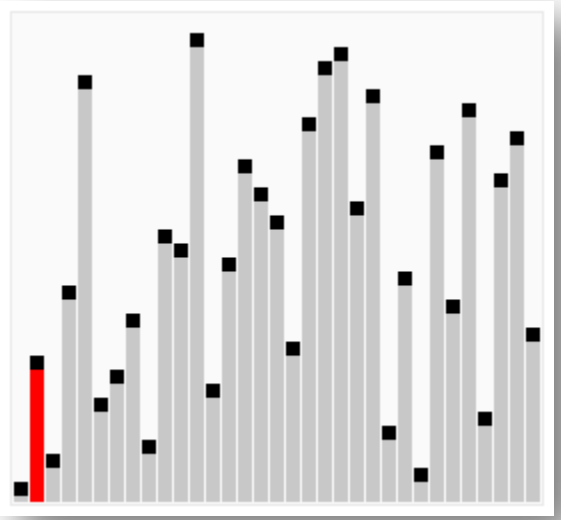
Пространственная сложность: O(1)

```cpp
#include <iostream>
#include <vector>
using namespace std;

void printVector(const vector<int>& arr) {
    for (int num : arr) cout << " " << num;
}

void bubbleSort(vector<int>& arr) {
    int n = arr.size();
    bool swapped; // Флаг для отслеживания обменов

    for (int i = 0; i < n - 1; i++) { // Внешний цикл
        swapped = false; // Сбрасываем флаг обменов перед каждым проходом

        for (int j = 0; j < n - i - 1; j++) { // Внутренний цикл
            if (arr[j] > arr[j + 1]) { // Сравниваем соседние элементы
                swap(arr[j], arr[j + 1]); // Меняем местами, если порядок нарушен
                swapped = true; // Устанавливаем флаг, что был обмен
            }
        }
        // Если ни один из двух элементов не был поменян, то прерываем
        if (!swapped)
            break;
    }
}

int main() {
    setlocale(LC_ALL, "Russian");
    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };
    cout << "Unsorted array: "; printVector(arr);
    bubbleSort(arr);
    cout << "\n\nBubble sort:    "; printVector(arr); cout << "\n";
}
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array:  64 34 25 12 22 11 90

Bubble sort:     11 12 22 25 34 64 90
```

Сортировка перемешиванием (Cocktail Shaker Sort)

Коктейльная сортировка (Cocktail Shaker Sort) — является разновидностью алгоритма пузырьковой сортировки. Как и алгоритм пузырьковой сортировки, коктейльная сортировка сортирует массив элементов, многократно меняя местами соседние элементы, если они расположены в неправильном порядке. Однако коктейльная сортировка также движется в обратном направлении после каждого прохода по массиву, что в некоторых случаях делает её более эффективной. Коктейльная сортировка поочередно обходит заданный массив в обоих направлениях.



Временная сложность: O(n²)

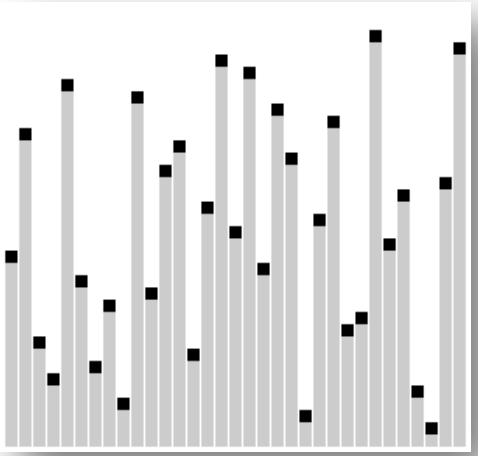
Пространственная сложность: O(1)

```cpp
void cocktail_sort(vector<int>& arr) {
    int n = arr.size();
    bool swapped = true;
    int start = 0;   int end = n - 1;
    while (swapped) {
        // Bubble Sort - слева направо
        swapped = false;
        for (int i = start; i < end; i++) {
            if (arr[i] > arr[i + 1]) {
                swap(arr[i], arr[i + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
        end--;
        // Bubble Sort - справа налево
        swapped = false;
        for (int i = end - 1; i >= start; i--) {
            if (arr[i] > arr[i + 1]) {
                swap(arr[i], arr[i + 1]);
                swapped = true;
            }
        }
        start++;
    }
}
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array:  52 12 49 31 7 26

Cocktail sort:   7 12 26 31 49 52
```

Сортировка расчёской (Comb Sort)

Сортировка расчёской (Comb Sort) — это, по сути, улучшение пузырьковой сортировки. Основная идея — устранить маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком. Пузырьковая сортировка всегда сравнивает соседние значения. Поэтому все инверсии удаляются одна за другой. Сортировка гребенкой улучшает пузырьковую сортировку, используя зазор размером больше. Зазор начинается с большого значения и уменьшается с каждой итерацией, пока не достигнет значения 1. Таким образом, сортировка гребенкой удаляет более одного значения инверсии за один обмен и работает лучше пузырьковой сортировки.



Временная сложность: O(n²)

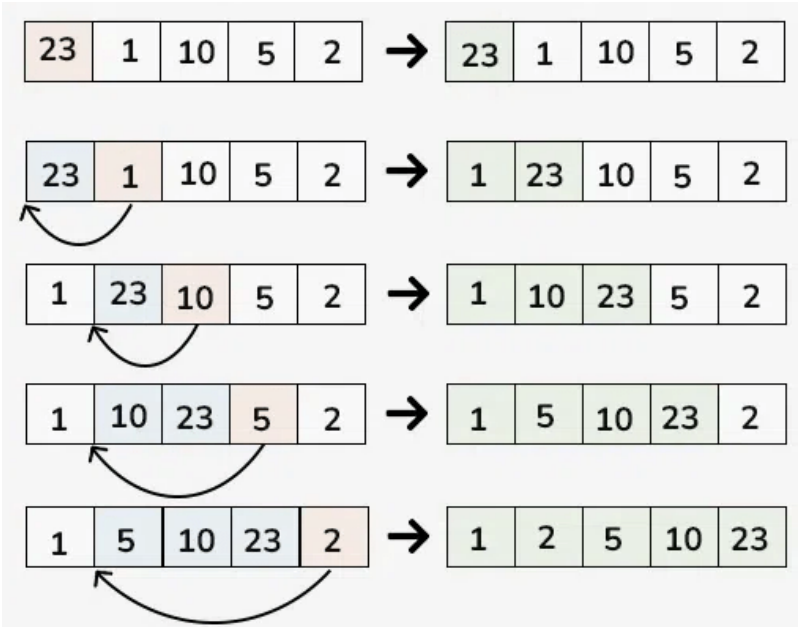
Пространственная сложность: O(1)

```
int getNextGap(int gap) // Функция для вычисления следующего промежутка (gap)
{
    gap = (gap * 10) / 13; // Уменьшаем промежуток с коэффициентом сжатия 1.3
    if (gap < 1) return 1; // Промежуток будет не меньше 1
    return gap;
}

void combSort(int a[], int n)
{
    int gap = n; // Инициализируем начальный промежуток равным размеру массива
    bool swapped = true; // Флаг, были ли обмены элементов на предыдущей итерации
    // Продолжаем цикл, пока промежуток не станет равным 1 И не будет обменов
    while (gap != 1 || swapped == true)
    {
        gap = getNextGap(gap); // Вычисляем следующий промежуток
        swapped = false; // Сбрасываем флаг обменов
        // Сравниваем элементы на расстоянии gap друг от друга
        for (int i = 0; i < n - gap; i++)
        {
            // Если элементы находятся в неправильном порядке, меняем их местами
            if (a[i] > a[i + gap])
            {
                swap(a[i], a[i + gap]);
                swapped = true; // Устанавливаем флаг, что был произведен обмен
            }
        }
    }
}

int main()
{
    int arr[] = { 8, 4, 1, 56, 3, -44, 23, -6, 28, 0 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Unsorted array: "; printArray(arr, n);  cout << "\n\n";
    combSort(arr, n);
    cout << "Comb sort:      "; printArray(arr, n);  cout << "\n";
    return 0;
}
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array: 8 4 1 56 3 -44 23 -6 28 0

Comb sort:      -44 -6 0 1 3 4 8 23 28 56
```

Сортировка вставкой (Insertion Sort)

Сортировка вставкой (Insertion Sort) — это простой алгоритм сортировки, который итеративно вставляет каждый элемент неотсортированного списка на его правильную позицию в отсортированной части списка. Это похоже на сортировку игральных карт в руках. Вы делите карты на две группы: отсортированные и неотсортированные. Затем вы выбираете карту из неотсортированной группы и кладете ее на нужное место в отсортированной группе.



-Начнем со второго элемента массива, поскольку первый элемент считается отсортированным.
-Сравним второй элемент с первым, если второй элемент меньше, поменяйте их местами.
-Перейдём к третьему элементу, сравните его с первыми двумя и поместите на правильную позицию.
-Повторяем, пока весь массив не будет отсортирован.

Преимущества:

- «in-place» алгоритм - не требует дополнительного объема памяти.
- Стабильный алгоритм сортировки
- Адаптивный. Количество инверсий прямо пропорционально количеству перестановок. Например, для отсортированного массива перестановка не производится, и занимает всего O(n) времени.

Недостатки:

- Неэффективно для больших списков.
- В большинстве случаев не так эффективно, как другие алгоритмы сортировки (например, сортировка слиянием, быстрая сортировка).

Временная сложность: $O(n^2)$

Пространственная сложность: O(1)

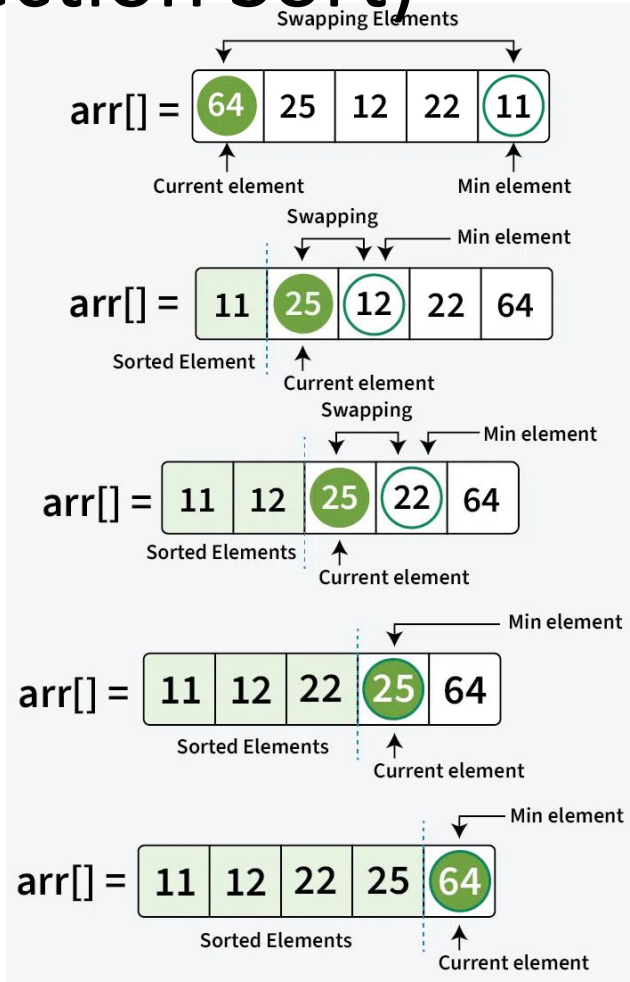
```cpp
#include <iostream>
#include <vector>
using namespace std;

void printVector(const vector<int>& arr) {
    for (int num : arr) cout << " " << num;
}

void insertionsort(vector<int>& arr, int n) {
    // Начинаем с индекса 1, так как первый элемент считается отсортиров
    for (int i = 1; i < n; ++i) {
        int key = arr[i]; // Текущий элемент, который нужно вставить в о
        int j = i - 1; // Индекс последнего элемента в отсортированной ч

        // Сдвигаем элементы большие чем key на одну позицию вправо
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j]; // Сдвигаем элемент вправо
            j = j - 1; // Переходим к следующему элементу слева
        }
        arr[j + 1] = key; // Вставляем key на правильную позицию в отсор
    }
}

int main() {
    setlocale(LC_ALL, "Russian");
    vector<int> arr = { 24, 1, 13, 5, 36, 4, 18, 9, 22};
    int n = arr.size();
    cout << "Unsorted array: "; printVector(arr);
    insertionsort(arr, n);
    cout << "\n\nInsertion Sort: "; printVector(arr);
    return 0;
}
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array:  24 1 13 5 36 4 18 9 22

Insertion Sort:  1 4 5 9 13 18 22 24 36
```

Сортировка выбором (Selection Sort)

Сортировка выбором (Selection Sort) — это алгоритм сортировки, основанный на сравнении, который многократно выбирает наименьший (или наибольший) элемент из неотсортированной части массива и меняет его местами с первым неотсортированным элементом. Этот процесс продолжается до тех пор, пока массив не будет полностью отсортирован.

1. Сначала мы находим наименьший элемент и меняем его местами с первым элементом. Таким образом, наименьший элемент оказывается на нужной позиции.
2. Затем мы находим наименьший среди оставшихся элементов (или второй по величине) и меняем его местами со вторым элементом.
3. Мы продолжаем это делать, пока все элементы не будут перемещены в нужные позиции.



Преимущества:

- Простота понимания и реализации.
- in-place» алгоритм - не требует дополнительного объема памяти.
- Требует меньше перестановок (или записей в память) по сравнению со многими другими стандартными алгоритмами. Только циклическая сортировка (cycle sort) превосходит её по объёму записи в память.

Недостатки:

- Временная сложность
- Не сохраняет относительный порядок равных элементов, т.е. неустойчивый.

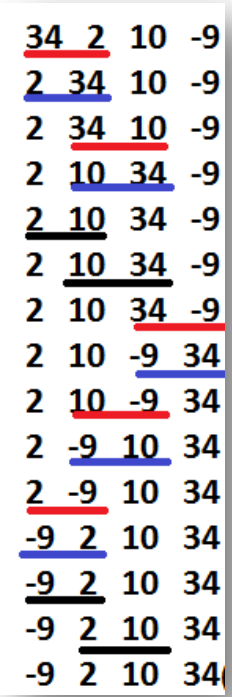
Временная сложность: O(n²)

Пространственная сложность: O(1)

```cpp
#include <iostream>
#include <vector>
using namespace std;

void printVector(const vector<int>& arr) {
    for (int num : arr) cout << " " << num;
}

void selectionSort(vector<int>& arr) {
    int n = arr.size();
    // Проходим по всем элементам массива, кроме последнего
    for (int i = 0; i < n - 1; ++i) {
        int min_idx = i; // Предполагаем, что текущий элемент является минимальным

        // Проходим по неотсортированной части массива для поиска реального минимума
        for (int j = i + 1; j < n; ++j) {

            // Если находим элемент меньше текущего минимума
            if (arr[j] < arr[min_idx]) {
                min_idx = j; // Обновляем индекс минимального элемента
            }
        }
        // Перемещаем найденный минимальный элемент на его правильную позицию
        swap(arr[i], arr[min_idx]);
    }
}

int main() {
    setlocale(LC_ALL, "Russian");
    vector<int> arr = { 56, 8, 12, 22, 11, 43, 3, 23, 51, 18 };
    cout << "Unsorted array: "; printVector(arr);
    selectionSort(arr);
    cout << "\n\nSelection sort: "; printVector(arr); cout << "\n";
    return 0;
}
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array:  56 8 12 22 11 43 3 23 51 18

Selection sort:  3 8 11 12 18 22 23 43 51 56
```

Гномья сортировка(Gnome Sort)



Сортирует следующим образом:
-Он смотрит на соседний и предыдущий горшок; если они расположены в правильном порядке, он делает шаг вперёд, в противном случае он меняет их местами и делает шаг назад.
-Если предыдущего горшка нет (он в начале ряда горшков), он делает шаг вперёд; если рядом нет горшка (он в конце ряда горшков), он закончил.

•Подчёркнутые элементы — это рассматриваемая пара.
•«Красным» отмечены пары, которые нужно поменять местами.
•Результат обмена окрашен в синий цвет.

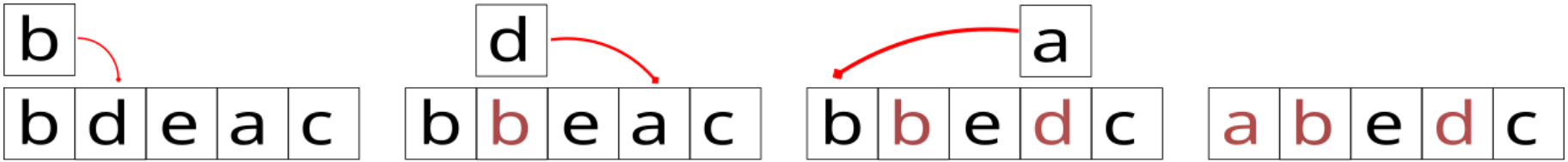
Временная сложность: O(n²)
Пространственная сложность: O(1)

```cpp
#include <iostream>
using namespace std;

void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)  cout << arr[i] << " ";
}


void gnomeSort(int arr[], int n)
{
    int index = 0; // Начинаем с первого элемента
    // Продолжаем, пока не дойдем до конца массива
    while (index < n) {
        // Если находимся в начале массива, переходим к следующему элементу
        if (index == 0)
            index++;
        // Если текущий элемент больше или равен предыдущему, идем вперед
        if (arr[index] >= arr[index - 1])
            index++;
        else {
            // Если текущий элемент меньше предыдущего, меняем их местами
            swap(arr[index], arr[index - 1]);
            // Возвращаемся на шаг назад для проверки предыдущих элементов
            index--;
        }
    }
    return;
}

int main()
{
    int arr[] = { 6, 21, 10, -9, 19, 4, -11 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Unsorted array: "; printArray(arr, n);  cout << "\n\n";
    gnomeSort(arr, n);
    cout << "Gnome sort:     "; printArray(arr, n);  cout << "\n";
    return (0);
}
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array: 6 21 10 -9 19 4 -11

Gnome sort:     -11 -9 4 6 10 19 21
```

Циклическая сортировка (Cycle sort)

Циклическая сортировка (Cycle sort) — это нестабильный алгоритм сортировки на месте, особенно полезный при сортировке массивов, содержащих элементы с узким диапазоном значений. Она основана на идее о том, что перестановку, которую нужно отсортировать, можно разделить на циклы, которые можно поворачивать по отдельности, чтобы получить отсортированный результат. Затем алгоритм выполняет серию перестановок, чтобы поместить каждый элемент на правильную позицию внутри цикла, пока все циклы не будут завершены и массив не будет отсортирован. Каждое значение либо записывается ноль раз, если оно уже находится на правильном месте, либо записывается один раз на правильное место. Это соответствует минимальному количеству перезаписей, необходимых для завершения сортировки на месте.



Преимущества:

-«in-place» алгоритм - не требуется дополнительная память.

- Минимальное количество записей в память.

-Сортировка по циклу полезна, когда массив хранится в EEPROM или FLASH.

Недостатки:

-Используется редко.

- Временная сложность $O(n^2)$ для всех случаев.

-Нестабильный алгоритм сортировки.

Применение:

-Этот алгоритм сортировки лучше всего подходит для ситуаций, когда операции записи или подкачки памяти требуют больших затрат.

Временная сложность: $O(n^2)$

Пространственная сложность: O(1)

```cpp
void cycleSort(int arr[], int n)
{
    int writes = 0; // Счетчик количества операций записи в память

    // Проходим по всем элементам массива, кроме последнего
    for (int cycle_start = 0; cycle_start <= n - 2; cycle_start++) {
        int item = arr[cycle_start]; // Запоминаем текущий элемент как начальную точку цикла

        // Находим правильную позицию для элемента item
        int pos = cycle_start;
        for (int i = cycle_start + 1; i < n; i++)
            if (arr[i] < item) // Считаем количество элементов меньше item справа от текущей позиции
                pos++;

        // Если элемент уже находится на правильной позиции, переходим к следующему
        if (pos == cycle_start) continue;
        while (item == arr[pos]) pos += 1; // Пропускаем все дубликаты элементов

        // Помещаем элемент на правильную позицию, если это необходимо
        if (pos != cycle_start) {
            swap(item, arr[pos]);
            writes++; // Увеличиваем счетчик операций записи
        }

        // Обрабатываем оставшуюся часть цикла
        while (pos != cycle_start) {
            pos = cycle_start;  // Начинаем новый цикл с текущей позиции

            // Снова находим правильную позицию для элемента item
            for (int i = cycle_start + 1; i < n; i++)
                if (arr[i] < item) pos += 1;
            while (item == arr[pos]) pos += 1; // Пропускаем все дубликаты элементов

            // Помещаем элемент на правильную позицию, если он еще не на месте
            if (item != arr[pos]) {
                swap(item, arr[pos]);
                writes++;
            }
        }
    }
    // Количество операций записи для анализа эффективности
    cout << "Writes: " << writes << endl;
}

int main()
{
    int arr[] = { 6, 21, 10, -9, 19, 4, -11 };
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array: 6 21 10 -9 19 4 -11
Writes: 7
Cycle sort:    -11 -9 4 6 10 19 21
```

Сортировка по цепочкам (Strand Sort)

Рекурсивный алгоритм сортировки, который сортирует элементы списка в порядке возрастания. Сначала алгоритм перемещает первый элемент списка во вспомогательный список. Затем он сравнивает последний элемент во вспомогательном списке с каждым последующим элементом в исходном списке. Как только в исходном списке появляется элемент, который больше последнего элемента во вспомогательном списке, этот элемент удаляется из исходного списка и добавляется во вспомогательный список, пока последний элемент во вспомогательном списке не будет сравнен с оставшимися элементами в исходном списке. Затем подсписок объединяется с новым списком. Процесс повторяется и объединяются все подсписки, пока все элементы не будут отсортированы.

```
void strandSort(list<int>& ip, list<int>& op)
{
    if (ip.empty()) return;

    list<int> sublist;
    sublist.push_back(ip.front());
    ip.pop_front();

    for (auto it = ip.begin(); it != ip.end();)
    {
        if (*it > sublist.back()) {
            sublist.push_back(*it);
            it = ip.erase(it);
        }
        else it++;
    }
    op.merge(sublist);
    strandSort(ip, op);
}
```

4	2	5	1	3

- Он имеет ***O*(*n*²)** наихудшую временную сложность, которая возникает при обратной сортировке входного списка.
- Временная сложность в лучшем случае составляет ***O*(*n*)**, что происходит, когда входные данные уже отсортированы.
-«Not In-place» - алгоритм. Пространственная сложность ***O*(*n*).**

Сортировка обменом (Exchange Sort)

Алгоритм, используемый для сортировки как по возрастанию, так и по убыванию. Он сравнивает первый элемент со всеми остальными элементами, и если какой-либо элемент находится в неправильном порядке, он меняет местами.

Сортировку обменом иногда путают с пузырьковой сортировкой, хотя на самом деле эти алгоритмы различны. Сортировка обменом работает, сравнивая первый элемент со всеми элементами, расположенными выше него, меняя местами при необходимости, тем самым гарантируя, что первый элемент является правильным для окончательного порядка сортировки; затем она делает то же самое для второго элемента, и так далее.

```
void printArray(int arr[], int size)
{
    for (int i = 0; i < size; i++) cout << arr[i] << " ";
}

void exchangeSort(int num[], int size)
{
    int i, j, temp;
    for (i = 0; i < size - 1; i++) {
        // Внешний цикл: проходим по всем элементам массива, кроме последнего
        for (j = i + 1; j < size; j++) {
            // Внутренний цикл: проходим по всем элементам после текущего
            // Сортировка по возрастанию:
            // Если предыдущий элемент больше следующего, меняем их местами
            //if (num[i] > num[j]) {
            // Сортировка по убыванию:
            if (num[i] < num[j]) {
                // Обмен элементов местами
                temp = num[i];
                num[i] = num[j];
                num[j] = temp;
            }
        }
    }
}

int main()
{
    int arr[] = { 61, -42, 12, -9, 19, 23, -11 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Unsorted array: "; printArray(arr, n);
    exchangeSort(arr, n);
    cout << "\n\nExchange sort: "; printArray(arr, n);
    return 0;
}
```

```
Консоль отладки Microsoft Visual Studio
Unsorted array: 61 -42 12 -9 19 23 -11

Exchange sort: 61 23 19 12 -9 -11 -42
```

Имя	Лучшее	Среднее	Худшее	Память	Стабильный	In-place	Метод	Другие примечания
Comb sort	$n \log n$	n^2	n^2	1	No	Yes	Exchanging (Обмен)	Разновидность Bubble Sort с использованием динамически уменьшающегося интервала.
Insertion sort	n	n^2	n^2	1	Yes	Yes	Insertion (Вставка)	Сортирует подобно картам, вставляя каждый новый элемент в правильную позицию внутри уже отсортированной части.
Bubble sort	n	n^2	n^2	1	Yes	Yes	Exchanging	Постоянно сравнивает и меняет местами соседние элементы.
Cocktail shaker sort	n	n^2	n^2	1	Yes	Yes	Exchanging	Двунаправленный вариант Bubble Sort (слева-направо и справа-налево).
Gnome sort	n	n^2	n^2	1	Yes	Yes	Exchanging	Движется вперёд, если порядок правильный, и меняет местами + отступает на шаг, если порядок нарушен. Использует всего один цикл и один указатель.
Odd–even sort	n	n^2	n^2	1	Yes	Yes	Exchanging	Сравнение и обмен чётных/нечётных пар. Алгоритм для параллельных вычислений.
Strand sort	n	n^2	n^2	n	Yes	No	Selection	Извлекает отсортированные цепи (нити) из исходного списка и объединяет их.
Selection sort	n^2	n^2	n^2	1	No	Yes	Selection (Выбор)	Поиск минимума в неотсортированной части и обмен с её первым элементом. Минимальное количество обменов.
Exchange sort	n^2	n^2	n^2	1	No	Yes	Exchanging	Для каждого элемента сравнивает и обменивает его со всеми последующими, если нужно.
Cycle sort	n^2	n^2	n^2	1	No	Yes	Selection	Минимизирует количество операций записи, производя обмены только в рамках циклов (находит место для каждого элемента).