Лекция 4. Анализ эффективности: сортировка выбором и вставками.

Казахский национальный университет имени аль-Фараби

Кафедра компьютерных наук

Алгоритмы и структуры данных

Лектор: старший преподаватель, PhD, Кенжебек Ержан Ғалымжанұлы

Цель и план лекции

Цель лекции

Изучить алгоритмы пузырьковой сортировки, сортировки вставками и сортировки выбором, научиться оценивать и сравнивать их временную сложность.

План лекции

1. Постановка задачи сортировки
2. Пузырьковая сортировка и её оптимизация
3. Сортировка вставками
4. Сортировка выбором
5. Сравнение сложности, функция sort в C++

Алгоритмы сортировки

- Пузырьковая сортировка
- Сортировка вставками
- Сортировка выбором
- Быстрая сортировка
- Сортировка слиянием

Алгоритмы сортировки

В информатике алгоритм сортировки - это алгоритм, который размещает элементы списка в определенном порядке. Наиболее часто используемые порядки - это числовой и лексикографический порядок. Эффективная сортировка важна для оптимизации эффективности других алгоритмов (таких как алгоритмы поиска и слияния), которые требуют, чтобы входные данные находились в отсортированных списках.

Алгоритмы сортировки

Другими словами, отсортированный массив - это массив в определенном порядке. Например, [a, b, c, d] отсортированы по алфавиту, [1,2,3,4,5] - это список целых чисел, отсортированных в порядке возрастания, а [5,4,3,2,1] - это список целых чисел, отсортированный в порядке убывания.

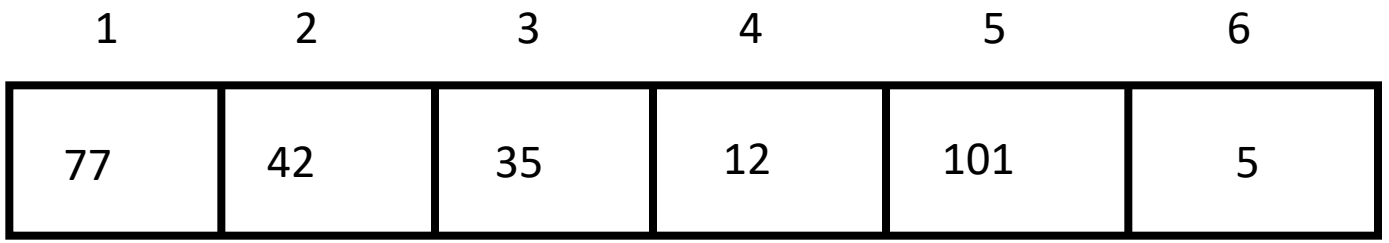
Алгоритм сортировки принимает массив в качестве входных данных и выводит перестановку этого отсортированного массива.

Пузырьковая сортировка

Пузырьковая сортировка - это простейший алгоритм сортировки, который работает, многократно меняя местами соседние элементы, если они находятся в неправильном порядке.

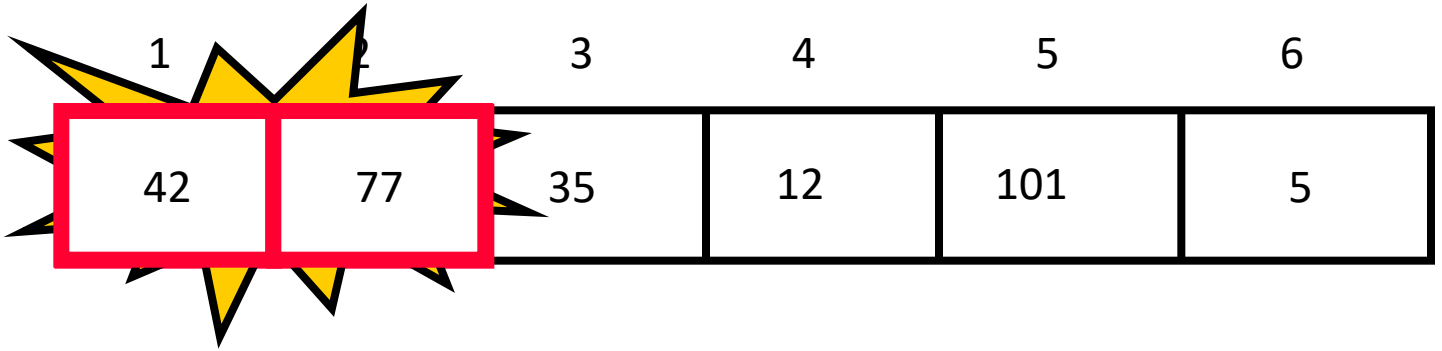
"Bubbling Up" самого большого элемента

- **Проход по всем элементам**
  - **Двигаться от начала к концу**
  - **“Bubble” наибольшее значение до конца с использованием попарных сравнений и обмена местами**



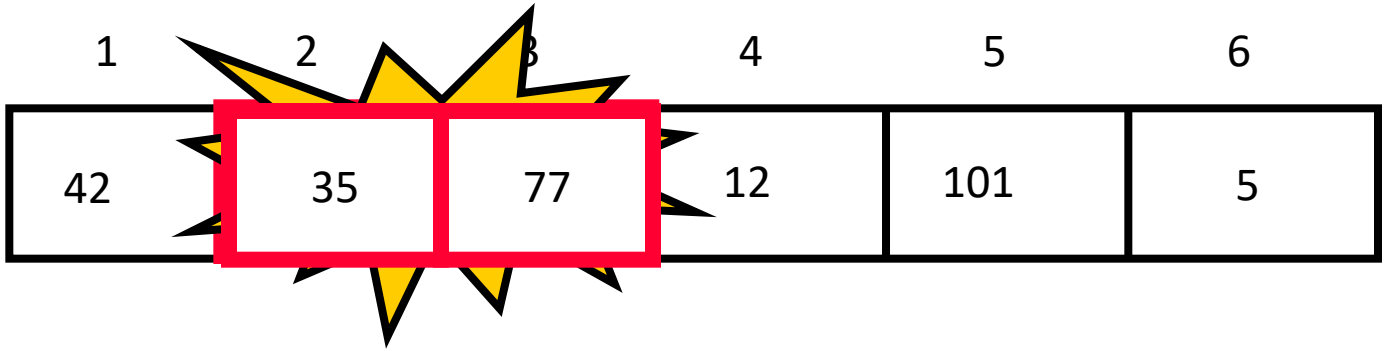
"Bubbling Up" самого большого элемента

- **Проход по всем элементам**
  - **Двигаться от начала к концу**
  - **“Bubble” наибольшее значение до конца с использованием попарных сравнений и обмена местами**



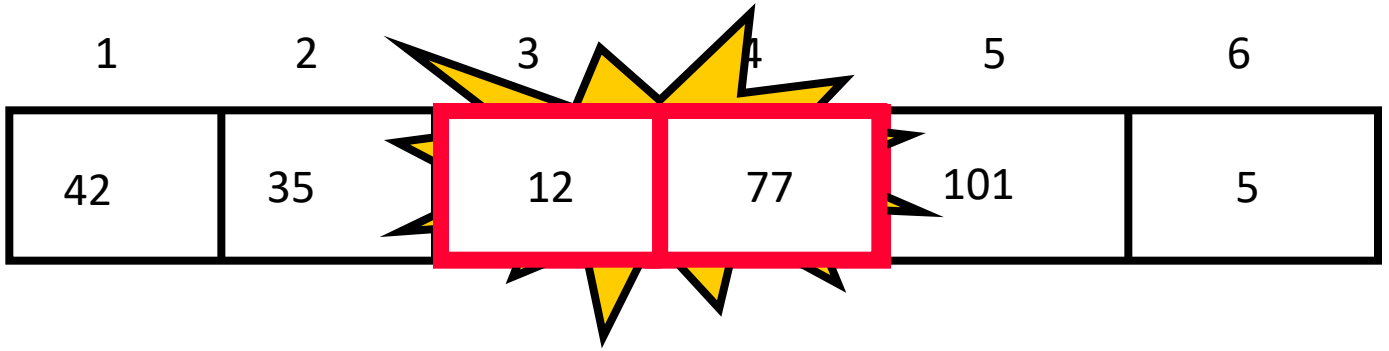
"Bubbling Up" самого большого элемента

- **Проход по всем элементам**
  - **Двигаться от начала к концу**
  - **“Bubble” наибольшее значение до конца с использованием попарных сравнений и обмена местами**



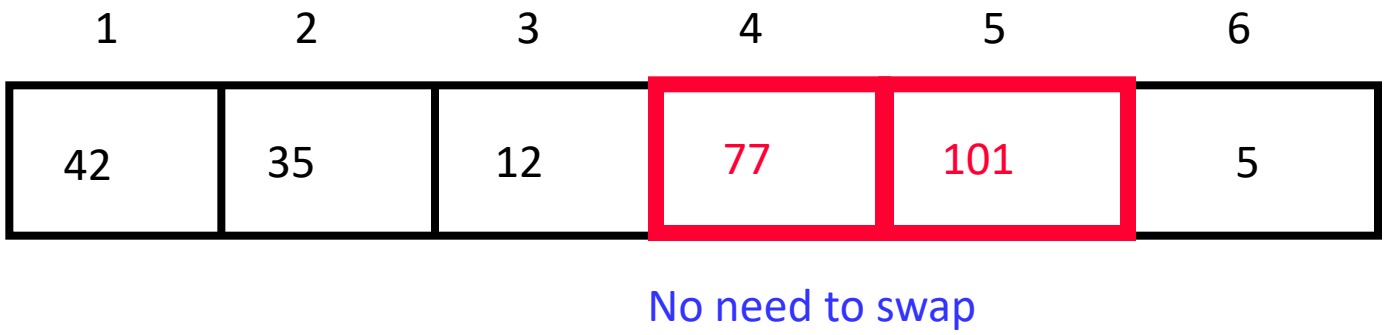
"Bubbling Up" самого большого элемента

- **Проход по всем элементам**
  - **Двигаться от начала к концу**
  - **“Bubble” наибольшее значение до конца с использованием попарных сравнений и обмена местами**



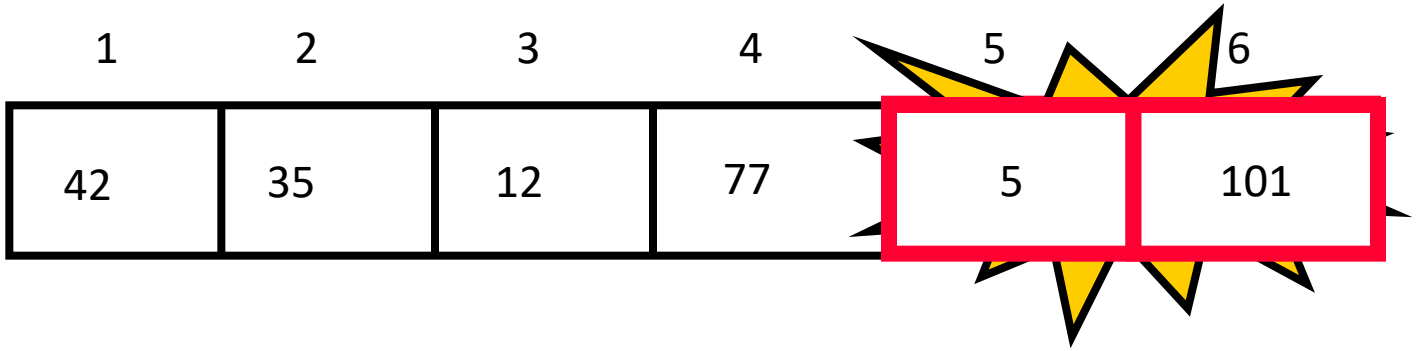
"Bubbling Up" самого большого элемента

- **Проход по всем элементам**
  - **Двигаться от начала к концу**
  - **“Bubble” наибольшее значение до конца с использованием попарных сравнений и обмена местами**



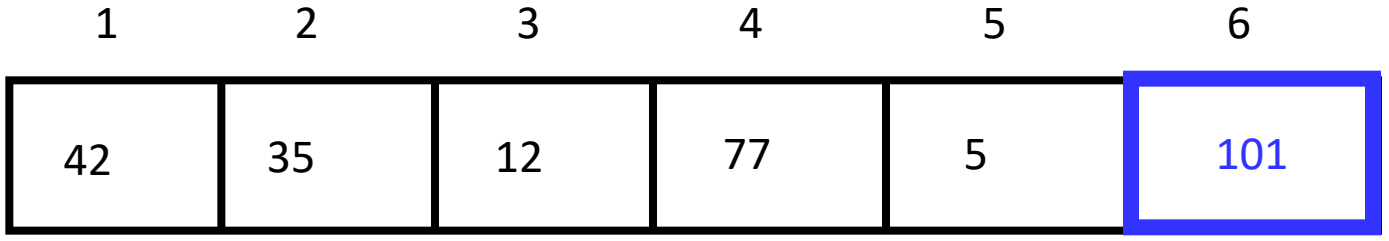
"Bubbling Up" самого большого элемента

- **Проход по всем элементам**
  - **Двигаться от начала к концу**
  - **“Bubble” наибольшее значение до конца с использованием попарных сравнений и обмена местами**



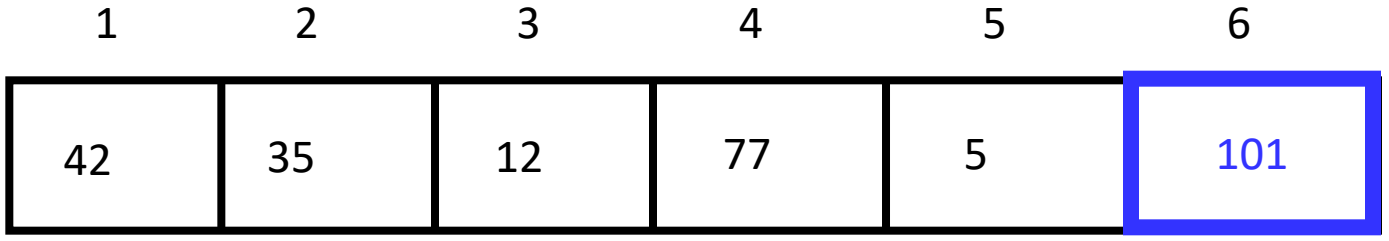
"Bubbling Up" самого большого элемента

- **Проход по всем элементам**
  - **Двигаться от начала к концу**
  - **“Bubble” наибольшее значение до конца с использованием попарных сравнений и обмена местами**



Интересные вещи

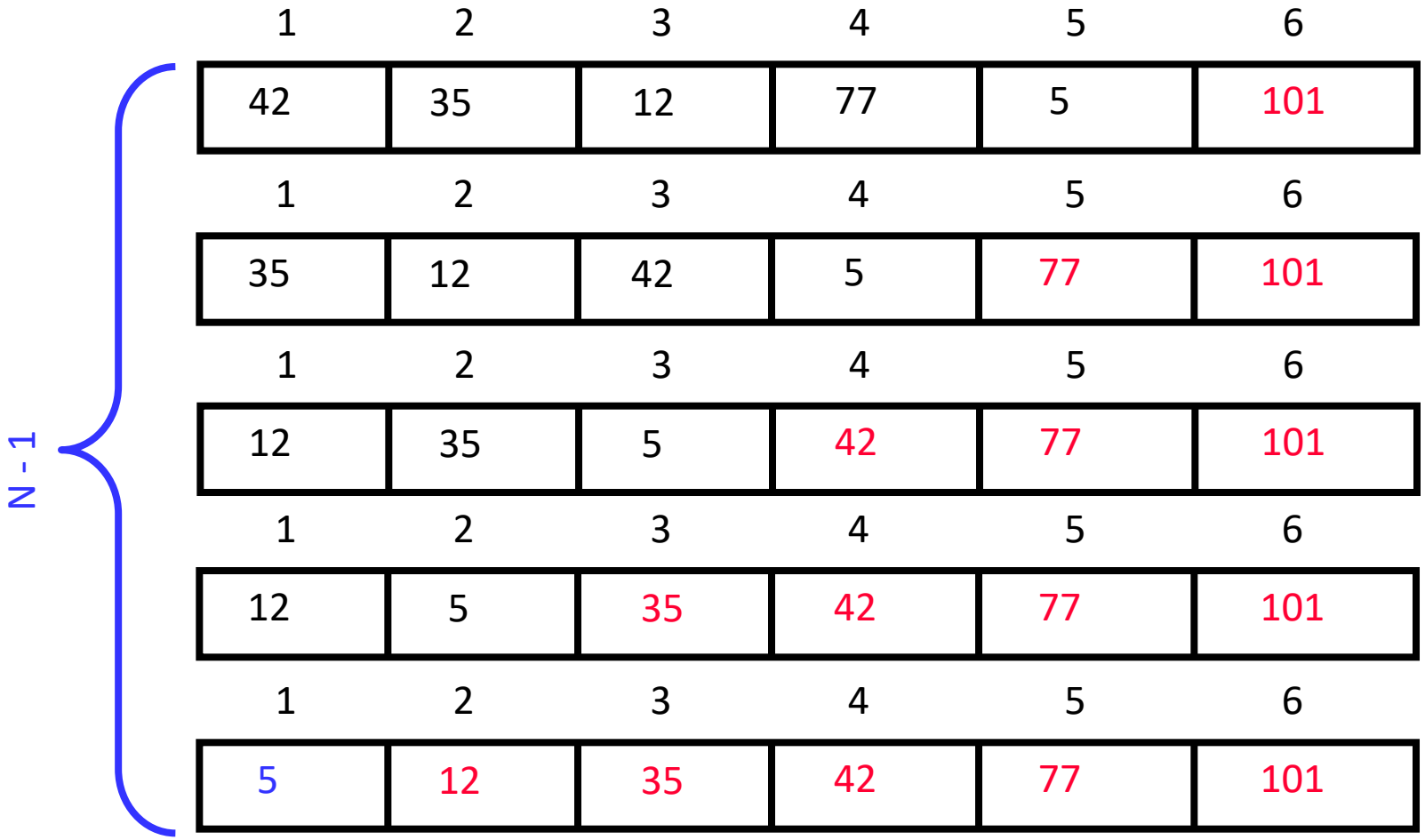
- **Обратите внимание, что правильно размещено только самое большое значение**
- **Все остальные значения по-прежнему не в порядке**
- **Итак, нам нужно повторить этот процесс**



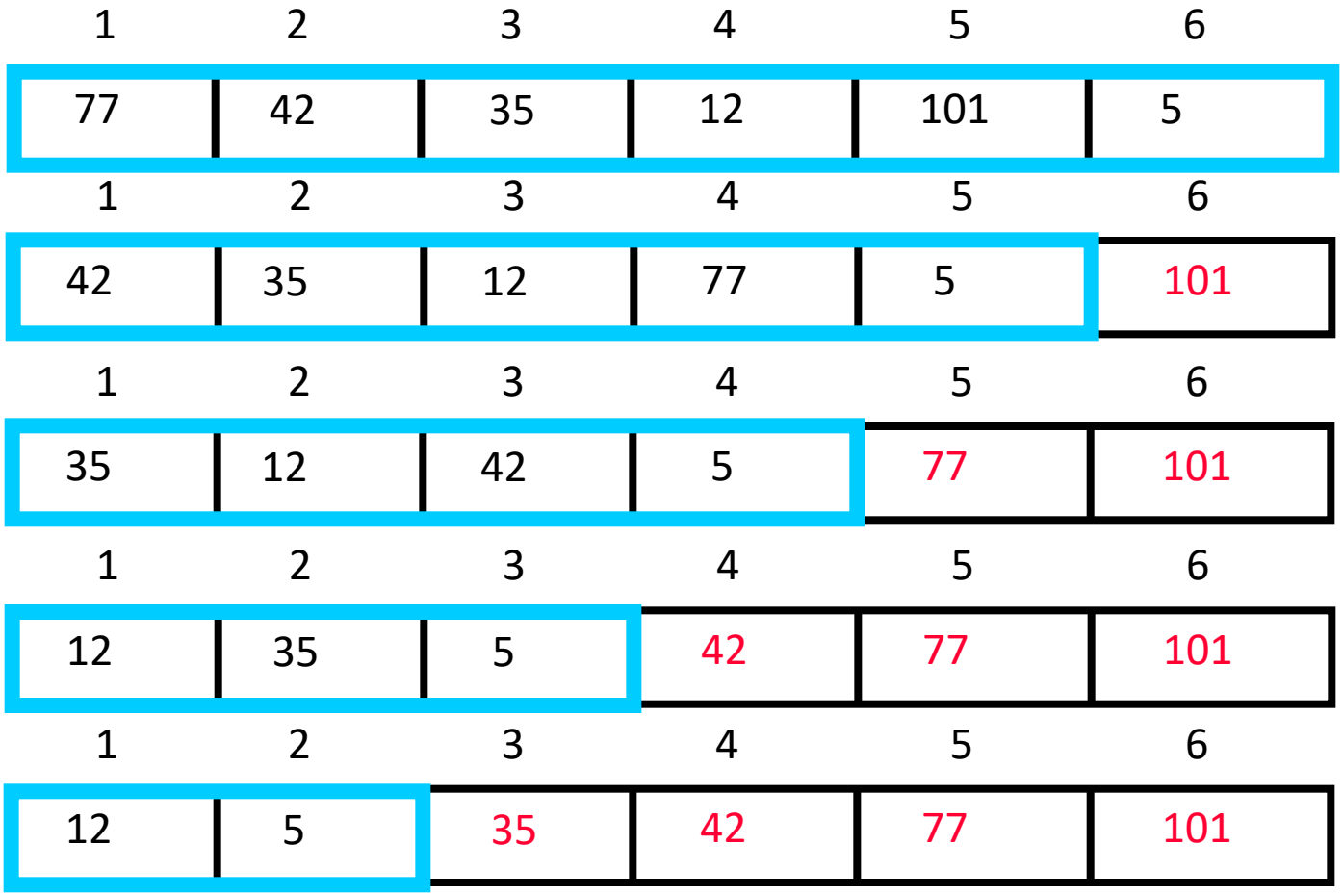
Сколько раз повторять “Bubble Up”?

- **Если у нас есть N элементов…**
- **И если каждый раз, когда мы всплываем элемент, мы помещаем его в правильное место ...**
- **Затем повторяем процесс «bubble up» N - 1 раз.**
- **Это гарантирует, что мы правильно разместим все N элементов.**

“Bubbling” всех элементов



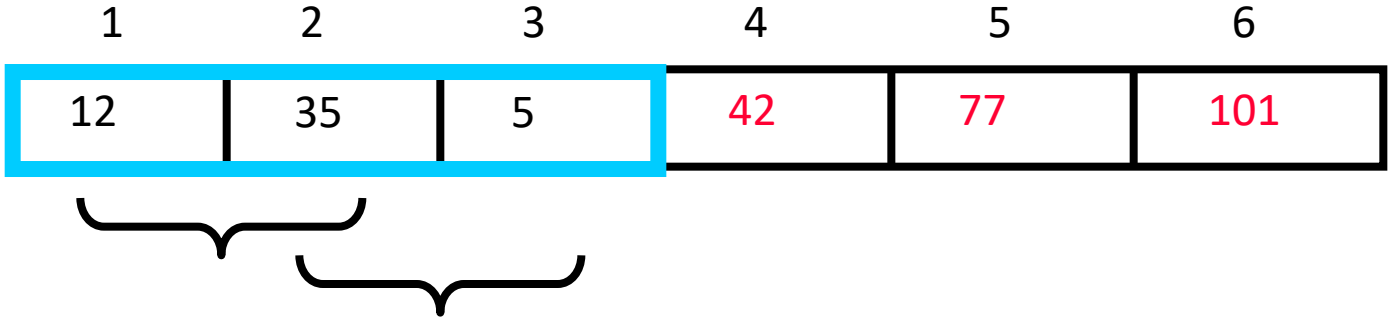
Сокращение количества сравнений



Сокращение количества сравнений

- **На t-шаге “bubble up”, нам нужно всего N-t сравнений.**

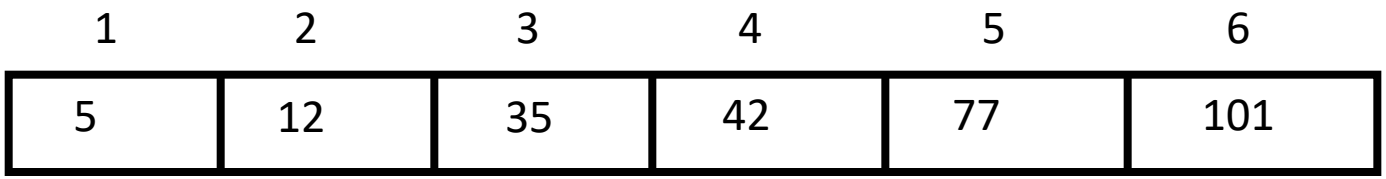
- **Для примера:**
  - **Это 4-тый “bubble up”**
  - **N есть 6**
  - **Нам нужно провести 2 сравнения**



Что если список уже отсортирован?

- **Что делать, если коллекция уже отсортирована?**
- **Что, если бы всего несколько элементов оказались не на своем месте, и после пары «bubble up» коллекция была отсортирована?**

- **Мы хотим, чтобы это можно было обнаружить и «остановить раньше»!**



Использование булевого “флага”

- **Мы можем использовать логическую переменную, чтобы определить, произошел ли какой-либо обмен во время «всплытия».**
- **Если изменения местами не произошло, значит, мы знаем, что массив уже отсортирован!**
- **Этот логический «флаг» необходимо сбрасывать после каждого «всплытия».**

Сложность

Лучший случай: O(n)

Средний случай: O(n^2)

Худший случай: O(n^2)

Сортировка вставками

Сортировка вставками - это простой алгоритм сортировки, который работает аналогично тому, как вы сортируете игральные карты в руках. Массив виртуально разделен на отсортированную и несортированную части. Значения из неотсортированной части выбираются и помещаются в правильное положение в отсортированной части.

Сортировка вставками

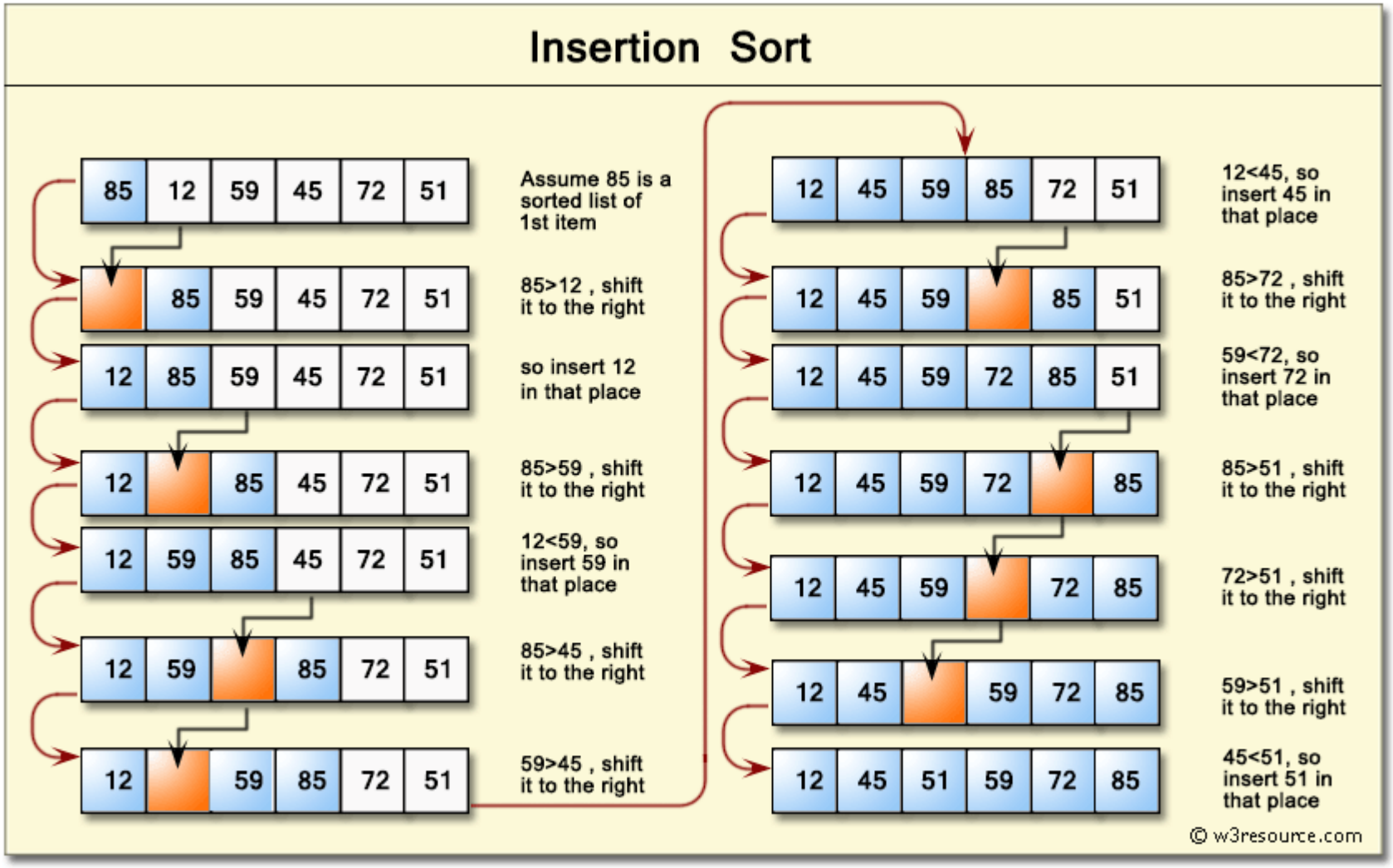
Алгоритм:

Чтобы отсортировать массив размера n в порядке возрастания:

1: перейти от arr [1] к arr [n] по массиву.

2: Сравните текущий элемент (ключ) с его предшественником.

3: Если ключевой элемент меньше своего предшественника, сравните его с предыдущими элементами. Переместите большие элементы на одну позицию вверх(направо), чтобы освободить место для замененного элемента.


Insertion Sort
85 12 59 45 72 51
Assume 85 is a sorted list of 1st item
85 59 45 72 51
85>12 , shift it to the right
12 85 59 45 72 51
so insert 12 in that place
12 85 45 72 51
85>59 , shift it to the right
12 59 85 45 72 51
12<59, so insert 59 in that place
12 59 85 72 51
85>45 , shift it to the right
12 59 85 72 51
59>45 , shift it to the right
12 45 59 85 72 51
12<45, so insert 45 in that place
12 45 59 85 51
85>72 , shift it to the right
12 45 59 72 85 51
59<72, so insert 72 in that place
12 45 59 72 85
85>51 , shift it to the right
12 45 59 72 85
72>51 , shift it to the right
12 45 59 72 85
59>51 , shift it to the right
12 45 51 59 72 85
45<51, so insert 51 in that place
© w3resource.com

Анализ сложности

```
for (i=1; i<n; i++) {
     j=i;
     while ((j>0) && (s[j] < s[j-1])) {
          swap(&s[j] ,&s[j-1]);
          j = j-i;
     }
}
```

Сколько итераций осуществляет внутренний цикл `while`? На этот вопрос сложно дать однозначный ответ, т. к. цикл может быть остановлен досрочно, если произойдет выход за границы массива (`j>0`) или элемент окажется на должном месте в отсортированной части массива (`s[j]<s[j-1]`). Так как в анализе наихудшего случая мы ищем верхнюю границу времени исполнения, то мы игнорируем досрочное завершение и полагаем, что количество исполняемых в этом цикле итераций всегда будет i. Более того, мы можем допустить, что *всегда* исполнятся n итераций, т. к. $i < n$. А т. к. внешний цикл исполняется n раз, то алгоритм сортировки вставками должен быть квадратичным, т. е. $O(n^2)$.

Такой грубый анализ методом округления всегда оказывается результативным, в том смысле, что полученная верхняя граница времени исполнения (O-большое) всегда будет правильной. Иногда этот результат может быть даже завышен в худшую сторону, т. е. в действительности время исполнения худшего случая окажется меньшим, чем результат, полученный при анализе. Тем не менее, я настоятельно рекомендую этот подход в качестве основы для простого анализа алгоритмов.

Сложность

Лучший случай: O(n)

Средний случай: O(n^2)

Худший случай: O(n^2)

Сортировка выбором

Сортировка выбором сортирует массив нахождением минимального элемента и заменяя его с первым элементом неотсортированной части.

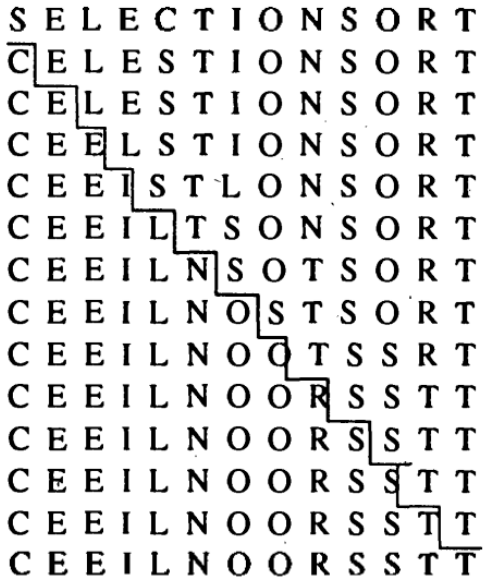
Сортировка выбором

Алгоритм

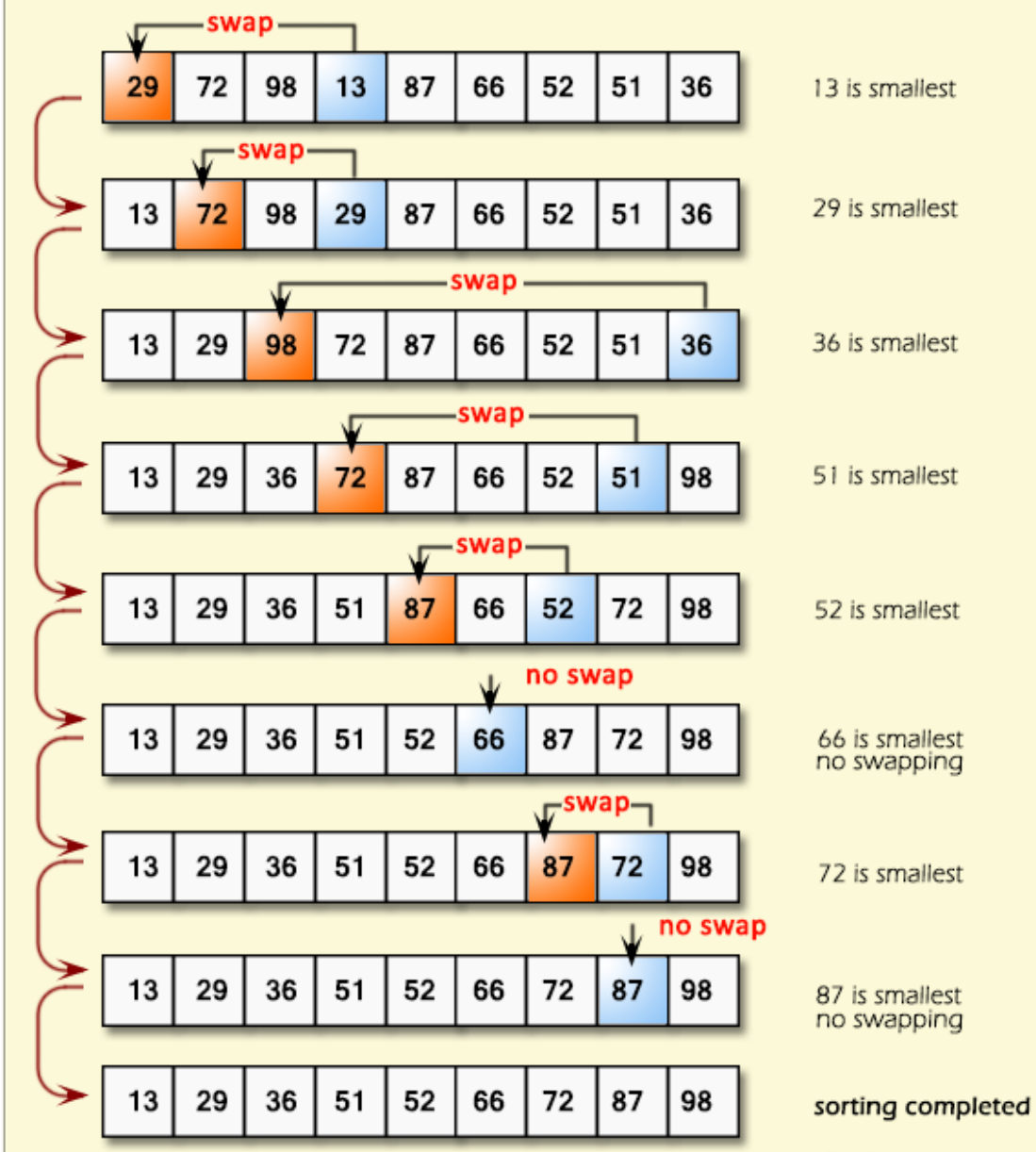
Алгоритм делит изначальный массив на две части.

1) Подмассив который уже отсортирован.
2) Подмассив который пока не отсортирован.

На каждой итерации алгоритма выбором, минимальный элемент неотсортированной части находится и помещается в конец отсортированной части.



Selection Sort



Сложность алгоритма

$$(n-1)+(n-2)+\ldots+1=\sum_{i=1}^{n-1} i$$

Арифметическая прогрессия,

$$\sum_{i=1}^{n-1} i=\frac{(n-1)+1}{2}(n-1)=\frac{1}{2}n(n-1)=\frac{1}{2}(n^2-n)$$

поэтому, сложность – $O(n^2)$

Сложность

Лучший случай: O(n^2)

Средний случай: O(n^2)

Худший случай: O(n^2)

Sort c++

```
vector<int> v = {4,2,5,3,5,8,3};
sort(v.begin(),v.end());
```

```
sort(v.rbegin(),v.rend());
```

```
int n = 7; // array size
int a[] = {4,2,5,3,5,8,3};
sort(a,a+n);
```

Sorting the string type vector:

```
bool comp(string a, string b) {
    if (a.size() != b.size()) return a.size() < b.size();
    return a < b;
}
```

```
sort(v.begin(), v.end(), comp);
```

Выводы

1. Пузырьковая сортировка, сортировка вставками и выбором просты в реализации, но имеют сложность $O(n^2)$ в худшем случае.
2. В пузырьковой сортировке «флаг» обмена позволяет завершить работу на уже отсортированном массиве за $O(n)$.
3. Сортировка вставками эффективна на почти отсортированных данных и сохраняет порядок равных элементов.
4. Сортировка выбором выполняет $O(n^2)$ сравнений, но не более $n - 1$ обменов.
5. Для больших массивов применяют более быстрые алгоритмы и стандартную функцию sort в C++.

Контрольные вопросы

1. Что такое сортировка? Какая сортировка называется устойчивой?
2. Опишите пузырьковую сортировку. Сколько проходов она выполняет?
3. Как «флаг» сокращает время работы пузырьковой сортировки?
4. В чём идея сортировки вставками? В каком случае она работает быстрее всего?
5. Опишите сортировку выбором. Сколько сравнений и обменов она выполняет?
6. Сравните три алгоритма по лучшему, среднему и худшему случаю.

Список литературы

Основная литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 3-е изд. — М.: Вильямс, 2013.
2. Седжвик Р., Уэйн К. Алгоритмы на Java. 4-е изд. — М.: Вильямс, 2013.
3. Кнут Д. Искусство программирования. Т. 3. Сортировка и поиск. 2-е изд. — М.: Вильямс, 2007.

Дополнительная литература

4. Скиена С. Алгоритмы. Руководство по разработке. 2-е изд. — СПб.: БХВ-Петербург, 2011.
5. Вирт Н. Алгоритмы и структуры данных. — М.: ДМК Пресс, 2010.
6. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. 4th ed. — MIT Press, 2022.

Электронные ресурсы

7. VisuAlgo — визуализация алгоритмов и структур данных: visualgo.net

Спасибо за внимание!