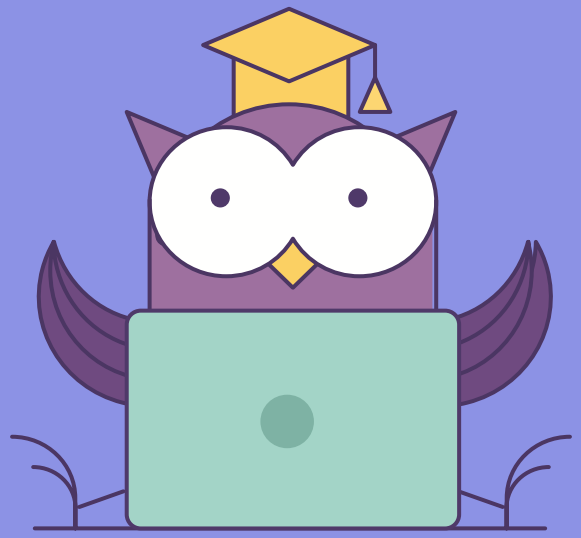
ОНЛАЙН-ОБРАЗОВАНИЕ

Не забыть включить запись!

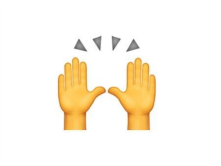
Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

Правила вебинара

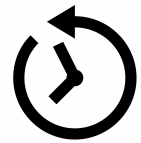


Активно участвуем

Задаем вопрос в чат или голосом. Лучше голосом.

slack

Off-topic обсуждаем в Slack #канал группы или #general



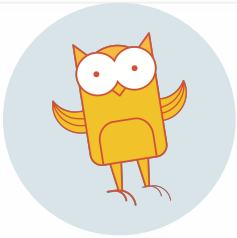
Вопросы в чате вижу, могу ответить не сразу

Цели вебинара

После занятия вы:

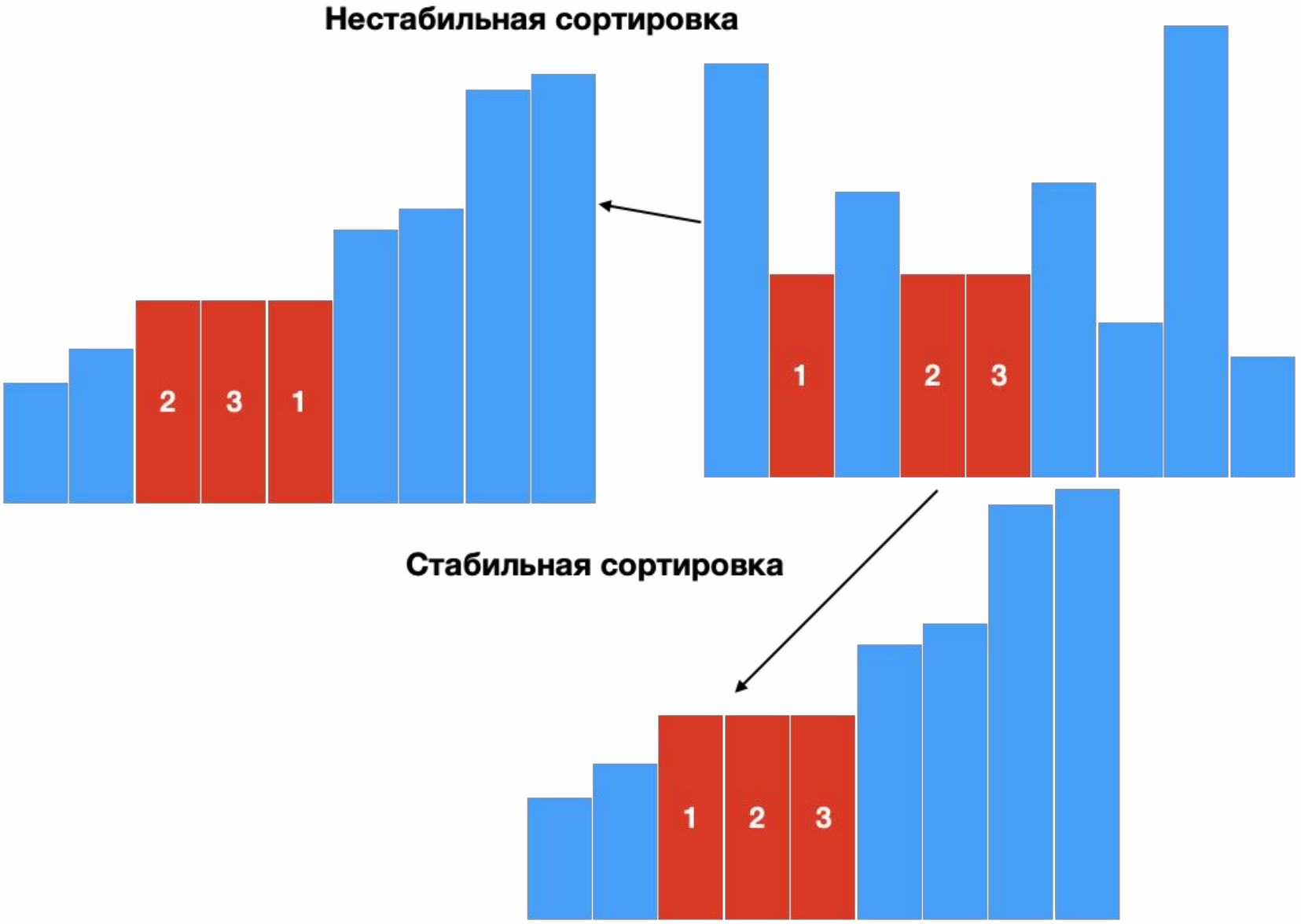
1. Изучим "наивные" подходы к сортировке, их ассимпотическую сложность
2. Определим понятие стабильной и нестабильной сортировки
3. Сравним разные "простые" алгоритмы сортировки

Классификация

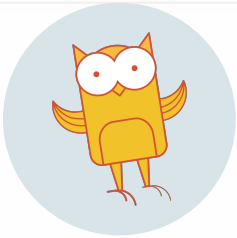


- Время работы
- Память
- Стабильность
- Количество обменов

Стабильность сортировки



Рассматриваемые алгоритмы



- Сортировка "пузырьком"
- Сортировка выбором
- Сортировка вставкой
- Сортировка Шелла

Сортировка "пузырьком"

Простой алгоритм, но неэффективный.

Его даже называют настолько плохим, что он не нужен в качестве примера алгоритма при обучении. Именно поэтому мы его разберем!

Сортировка "пузырьком"

```
void BubbleSort(ref int[] A)
{
   for (int i = 0; i < A.Length; i++)
      for (int j = 0; j < A.Length - 1; j++)
         if (A[j] > A[j + 1])
         {
           int z = A[j];
           A[j]=A[j+1];
           A[j + 1] = z;
         }
}
```

Сортировка "пузырьком"

Улучшение:
Если при выполнении прохода не было ни одного обмена элементов массива, то массив уже отсортирован и остальные проходы не нужны.

```
repeat
    flag := False; { обнуляем флаг }
    for j:=N-1 downto 1 do
      if A[j] > A[j+1] then
      begin
        c := A[j];
        A[j] := A[j+1];
        A[j+1] := c;
        flag := True; { поднимаем флаг }
      end;
until not flag;  { выход при flag=False }
```

Сортировка "пузырьком"

[4, 5, 8, 3, 1, 2, 6, 7, 9]

(4 5 **8 3** 1 2 6 7 9) -> (4 5 **3 8** 1 2 6 7 9)
(4 5 3 **8 1** 2 6 7 9) -> (4 5 3 **1 8** 2 6 7 9)
(4 5 3 1 **8 2** 6 7 9) -> (4 5 3 1 **2 8** 6 7 9)
(4 5 3 1 2 **8 6** 7 9) -> (4 5 3 1 2 **6 8** 7 9)
(4 5 3 1 2 6 **8 7** 9) -> (4 5 3 1 2 6 **7 8** 9)
(4 **5 3** 1 2 6 7 8 9) -> (4 **3 5** 1 2 6 7 8 9)
(4 3 **5 1** 2 6 7 8 9) -> (4 3 **1 5** 2 6 7 8 9)
(4 3 1 **5 2** 6 7 8 9) -> (4 3 1 **2 5** 6 7 8 9)
(**4 3** 1 2 5 6 7 8 9) -> (**3 4** 1 2 5 6 7 8 9)
(3 **4 1** 2 5 6 7 8 9) -> (3 **1 4** 2 5 6 7 8 9)
(3 1 **4 2** 5 6 7 8 9) -> (3 1 **2 4** 5 6 7 8 9)
(**3 1** 2 4 5 6 7 8 9) -> (**1 3** 2 4 5 6 7 8 9)
(1 **3 2** 4 5 6 7 8 9) -> (1 **2 3** 4 5 6 7 8 9)

Сортировка "пузырьком"

(4 5 8 3 1 2 6 7 9) -> (4 5 8 3 1 2 6 7 9)
(4 5 8 3 1 2 6 7 9) -> (4 5 8 3 1 2 6 7 9)
(4 5 **8 3** 1 2 6 7 9) -> (4 5 **3 8** 1 2 6 7 9)
(4 5 3 **8 1** 2 6 7 9) -> (4 5 3 **1 8** 2 6 7 9)
(4 5 3 1 **8 2** 6 7 9) -> (4 5 3 1 **2 8** 6 7 9)
(4 5 3 1 2 **8 6** 7 9) -> (4 5 3 1 2 **6 8** 7 9)
(4 5 3 1 2 6 **8 7** 9) -> (4 5 3 1 2 6 **7 8** 9)
(4 5 3 1 2 6 7 8 9) -> (4 5 3 1 2 6 7 8 9)
(4 5 3 1 2 6 7 8 9) -> (4 5 3 1 2 6 7 8 9)
(4 **5 3** 1 2 6 7 8 9) -> (4 **3 5** 1 2 6 7 8 9)
(4 3 **5 1** 2 6 7 8 9) -> (4 3 **1 5** 2 6 7 8 9)
(4 3 1 **5 2** 6 7 8 9) -> (4 3 1 **2 5** 6 7 8 9)
(4 3 1 2 5 6 7 8 9) -> (4 3 1 2 5 6 7 8 9)
(4 3 1 2 5 6 7 8 9) -> (4 3 1 2 5 6 7 8 9)
(4 3 1 2 5 6 7 8 9) -> (4 3 1 2 5 6 7 8 9)
(**4 3** 1 2 5 6 7 8 9) -> (**3 4** 1 2 5 6 7 8 9)
(3 4 1 2 5 6 7 8 9) -> (3 1 4 2 5 6 7 8 9)

...

Сортировка "пузырьком"

Асимптотическая сложность:

В лучшем случае O(n)

В худшем и среднем O(n²) (n+(n-1)+...1 = n(n-1)/2)

Память O(1)

Обменов в среднем O(n²)

Сортировка выбором

принцип: находится наименьший элемент, меняется местами с первым неотсортированным

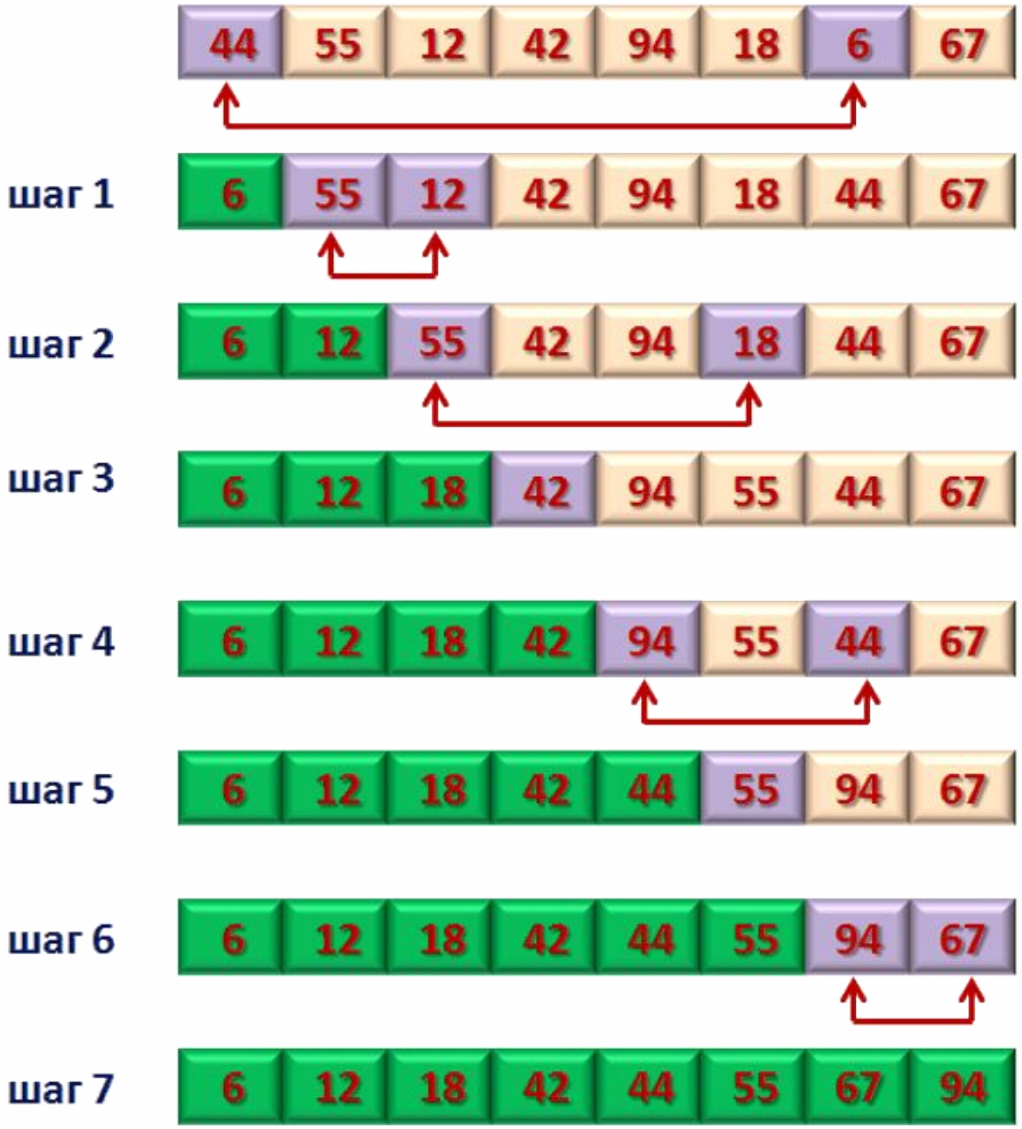
Сортировка выбором

- Массив делится на сортированную и неотсортированную часть.
- В неотсортированном подмассиве ищется локальный максимум (минимум).
- Найденный максимум (минимум) меняется местами с последним (первым) элементом в подмассиве.
- Если в массиве остались неотсортированные подмассивы — смотри пункт 1.

Сортировка выбором

```
void selectionsort(int* l, int* r) {
	for (int *i = l; i < r; i++) {
		int minz = *i, *ind = i;
		for (int *j = i + 1; j < r; j++) {
			if (*j < minz) minz = *j, ind = j;
		}
		swap(*i, *ind);
	}
}
```

Сортировка выбором



Сортировка выбором

Анализ сложности

- Сколько всего сравнений выполняется в сортировке?
- Сколько шагов во внешнем цикле?
- Сколько шагов во вложенном цикле?

1. сложность O(n²), причём как в лучшем, так и худшем вариантах.
2. количество сравнений, как и в "пузырьке", вычисляется по формуле n*(n-1)/2.

Сортировка выбором

Доказательство корректности

Инвариант цикла

- представляет собой математическое выражение, в которое неустранимым образом входят переменные, значения которых изменяются от одного прохода цикла до другого

- истинен перед началом выполнения цикла (перед входом в первую итерацию)и после каждого прохода тела цикла.

Сортировка выбором

Доказательство корректности

Математическая индукция — метод математического доказательства, который используется, чтобы доказать истинность некоторого утверждения для всех натуральных чисел.

- сначала проверяется истинность утверждения с номером 1 — база (базис) индукции

- затем доказывается, что если верно утверждение с номером n, то верно и следующее утверждение с номером n + 1 — шаг индукции, или индукционный переход.

Сортировка выбором

Доказательство корректности

Порядок доказательства корректности цикла с помощью инварианта

- Доказывается, что выражение инварианта истинно перед началом цикла.
- Доказывается, что выражение инварианта сохраняет свою истинность после выполнения тела цикла (по правилу).
- Доказывается, что при истинности инварианта после завершения цикла переменные примут именно те значения, которые требуется получить.
- Доказывается , что цикл завершится, то есть условие завершения рано или поздно будет выполнено.

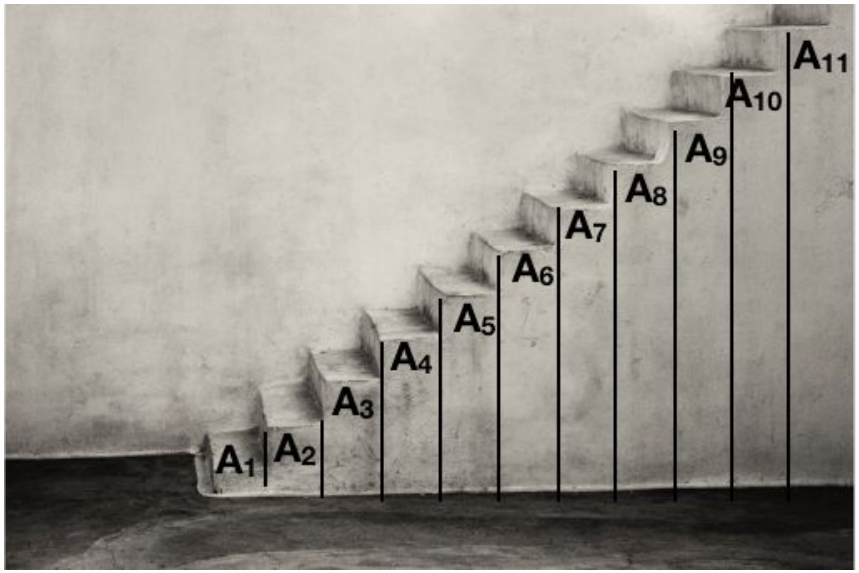
Сортировка выбором

Доказательство корректности

Определим понятие сортированного списка:

Такая перестановка (элементов массива), что для всех i, где i - индекс элемента в списке,

$$A_i \leq A_{i+1}$$



Сортировка выбором

Доказательство корректности

В нашем случае, если в алгоритме присутствует цикл (то есть почти всегда):

- Определим инвариант: начало списка до i -го элемента отсортировано.
- При инициализации (пустой список) отсортирован: верно
- Сохранение инварианта: после итерации цикла i+=1 ; начало списка отсортировано
- Завершение цикла: i = длине списка, весь список отсортирован

Сортировка выбором

Доказательство корректности

Доказательство завершимости цикла

```
//n = число элементов в массиве a[n]
int j = 0;
int min = 0;

do
{
   If  (min > a[j]) min = a[j]
    j++; //если не сделать, цикл будет бесконечным
}
while (j < n);
```

Сортировка выбором

Доказательство корректности

Почему так:

- Пустой список отсортирован по соглашению
- На каждом следующем шаге из списка берётся минимальный элемент
- Находим минимальный элемент, вставляем в конец отсортированного списка. При этом список остается **отсортированным**!
- Завершение цикла: добравшись до конца списка получаем, что список отсортирован

Сортировка выбором

Шаг 1:

Внешний For
  Внутренний For
(**9** **3** 4 6 5 7 8 2 0 1)
(**9** **3** 4 6 5 7 8 2 0 1)
(**9** **3** 4 6 5 7 8 2 0 1)
(**9** **3** 4 6 5 7 8 2 0 1)
(**9** **3** 4 6 5 7 8 2 0 1)
(**9** **3** 4 6 5 7 8 2 0 1)
(**9** 3 4 6 5 7 8 **2** 0 1)
(**9** 3 4 6 5 7 8 2 **0** 1)
(**9** 3 4 6 5 7 8 2 **0** 1)
swap
(**0** 3 4 6 5 7 8 2 **9** 1)

Шаг 2:

Внешний For
  Внутренний For
(**0** **3** 4 6 5 7 8 2 9 1)
(**0** **3** 4 6 5 7 8 2 9 1)
(**0** **3** 4 6 5 7 8 2 9 1)
(**0** **3** 4 6 5 7 8 2 9 1)
(**0** **3** 4 6 5 7 8 2 9 1)
(**0** **3** 4 6 5 7 8 **2** 9 1)
(**0** **3** 4 6 5 7 8 **2** 9 1)
(**0** **3** 4 6 5 7 8 2 9 **1**)
swap
(**0** **1** 4 6 5 7 8 2 9 **3**)

Сортировка выбором

Асимптотическая сложность:

В лучшем случае $O(n^2)$

В худшем и среднем $O(n^2)$

Память $O(1)$

Обменов в среднем $O(n)$

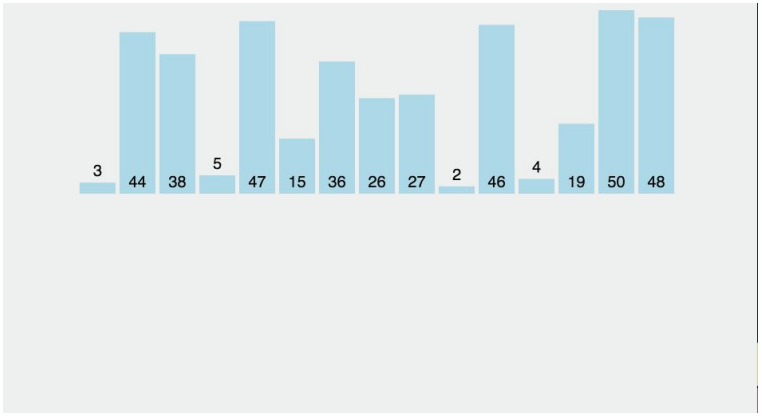
Нестабильная

Сортировка вставкой

принцип:
Перебираются элементы в неотсортированной части массива и по одному вставляются в отсортированную часть массива на то место, где он должен находиться.

Сортировка вставкой

```
InsertionSort(Array):
    for i in 0..n-1
        x = Array[i]
        j = i - 1
        while j >= 0 and Array[j] > x
            Array[j+1] = Array[j] // "сдвиг" вправо
            j = j - 1
        Array[j+1] = x  // Вставка x в отсортированную часть
        i = i + 1
```



Сортировка вставкой

Online алгоритм - который выполняет операции над потоком данных, не видя сразу весь набор данных.

К примеру, сортировка выбором не является Online алгоритмом - нужно искать минимум, а для этого важно иметь все элементы сортируемого массива.

Сортировка вставкой

1. Сложность $O(n^2)$. В лучшем случае $O(n)$ - если массив частично отсортирован.

2. Адаптивный - алгоритм эффективен при обработке уже отсортированных или частично отсортированных данных. Сложность $O(n)$

3. Стабильный.

4. Online.

Сортировка вставкой

[0, 1, 2, 8, 3, 4, 5, 6, 7, 9]

(**0 1 2 8** **3** 4 5 6 7 9)
(**0 1 2 3 8** **4** 5 6 7 9)
(**0 1 2 3 4 8** **5** 6 7 9)
(**0 1 2 3 4 5 8** **6** 7 9)
(**0 1 2 3 4 5 6 8** **7** 9)
(**0 1 2 3 4 5 6 7 8** **9**)

Сортировка вставкой

Асимптотическая сложность:

В лучшем случае O(n)

В худшем и среднем $O(n^2)$

Память O(1)

Обменов в среднем $O(n^2)$

Стабильная

Сортировка Шелла

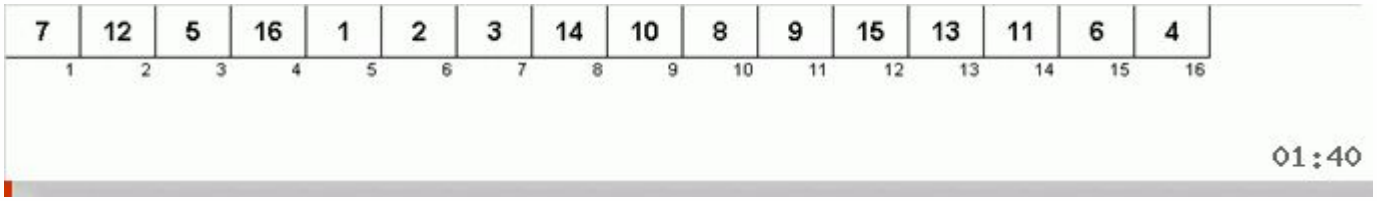
Сортировка Шелла (1959г.) - "улучшение" сортировки вставками / сортировки пузырьком

Сортировка Шелла

- Сначала сравниваются и сортируются (вставкой) между собой значения, стоящие один от другого на некотором расстоянии d.

- После этого процедура повторяется для некоторых меньших значений d

- Завершается сортировка Шелла упорядочиванием элементов при d=1 (то есть обычной сортировкой вставками).



Сортировка Шелла

Исходный массив	32	95	16	82	24	66	35	19	75	54	40	43	93	68	
После сортировки с шагом 5	32	35	16	68	24	40	43	19	75	54	66	95	93	82	6 обменов
После сортировки с шагом 3	32	19	16	43	24	40	54	35	75	68	66	95	93	82	5 обменов
После сортировки с шагом 1	16	19	24	32	35	40	43	54	66	68	75	82	93	95	15 обменов

На первом шаге сортируются подсписки, составленные из всех элементов A, различающихся на 5 позиций, то есть подсписки
A{5,1}=(32,66,40),
A{5,2}=(95,35,43),
A{5,3}}=(16,19,93),
A{5,4}=(82,75,68),
A{5,5}}=(24,54)

Сортировка Шелла

- Нестабильна
- сортировка работает существенно медленнее чем, сортировки со сложностью n*log(n) (например, сортировка слиянием).
- но бывает, что её сложность - O(n log n)
- Средняя временная сложность $O(n^2)$
- Зависит это и от того, насколько отсортирован массив и от коэффициента d
  Варианты подбора описаны здесь

Сортировка Шелла

Почему алгоритм может быть быстр?

Сортировка вставками хорошо работает, когда массив частично отсортирован.

- Происходит k раз сортировка вставкой. В лучшем случае у этой сортировки сложность O(n)
- Получаем сложность вроде O(k*n).
- Если k<<n, то с точки зрения асимптотической сложности это выгоднее, чем сложность $O(n^2)$

Заполните, пожалуйста, опрос о занятии

Спасибо
за внимание!