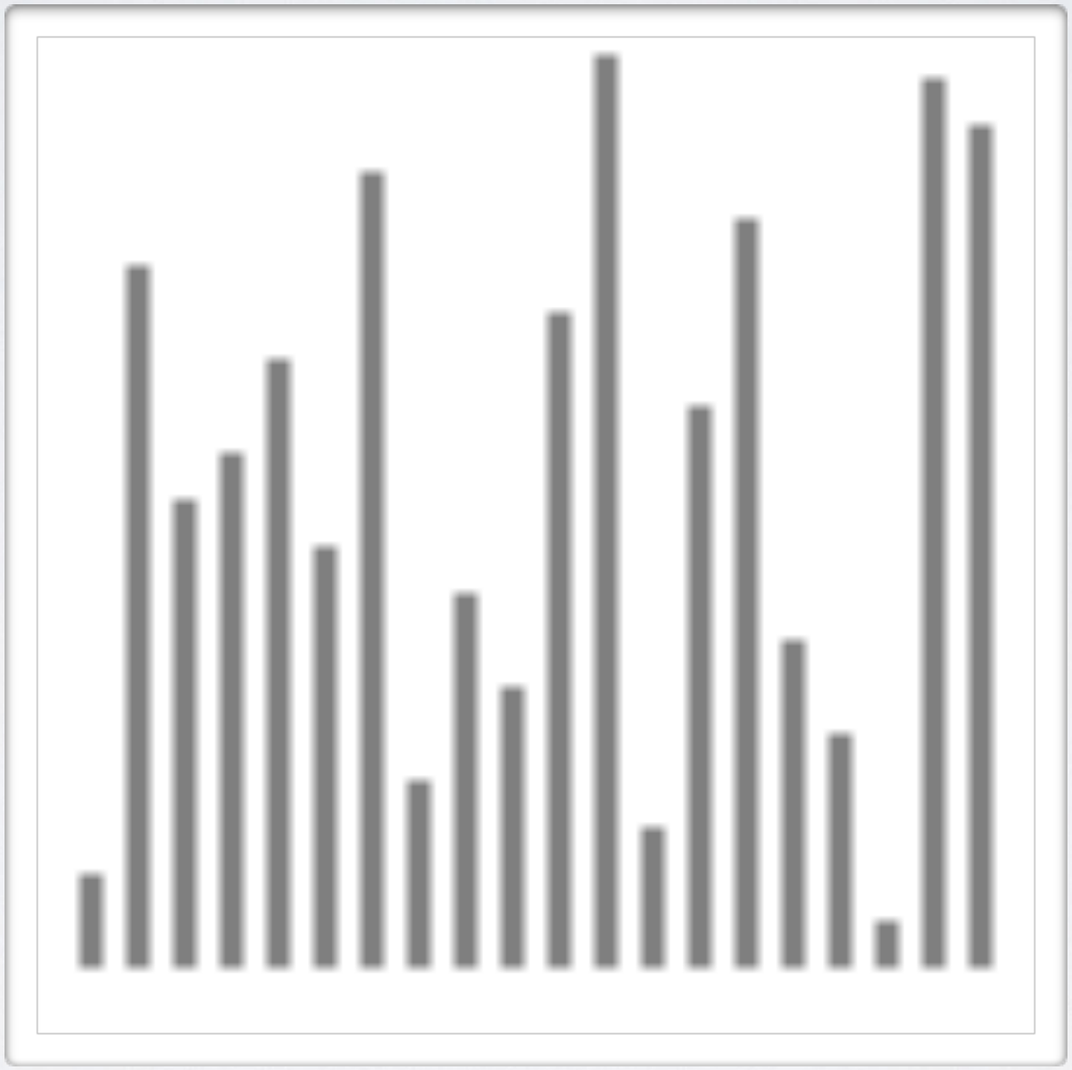
ОСНОВЫ ПРОГРАММНОГО КОНСТРУИРОВАНИЯ

Лекция № 4
26 сентября 2016 г.

СОРТИРОВКА ШЕЛЛА

1. Выбираем длину промежутка **1 ≤ d < N**.
2. Разбиваем массив на **d** подмассивов:
   [1, 1+d, 1+2d, …], [2, 2+d, 2+2d, …], …, [d-1, 2d-1, 3d-1, …].
3. Каждый подмассив сортируем сортировкой вставками.
4. Если **d > 1**, то уменьшаем **d** и переходим на шаг 2.
   Иначе массив отсортирован.

АНИМАЦИЯ



http://sorting-algorithms.com/shell-sort

АНАЛИЗ СОРТИРОВКИ ШЕЛЛА

Быстродействие зависит от выбора длин промежутков.

- Шелл: **[N/2, N/4, N/8, …, 1] ⇒ O(N^2)**,
- Хиббард: **(2^i – 1), […, 15, 7, 3, 1] ⇒ O($N^{3/2}$)**,
- Пратт: **2^p3^q, […, 9, 8, 6, 4, 3, 2, 1] ⇒ O(N ($\log N$)^2)**,
- Седжвик: *нетривиальные последовательности* **⇒ O($N^{4/3}$)**.

«РАЗДЕЛЯЙ И ВЛАСТВУЙ»

- **Разделение** задачи на несколько подзадач.
- **Покорение**: решение подзадач.
- **Комбинирование** решения исходной задачи из решений подзадач.

БЫСТРАЯ СОРТИРОВКА (QUICKSORT)

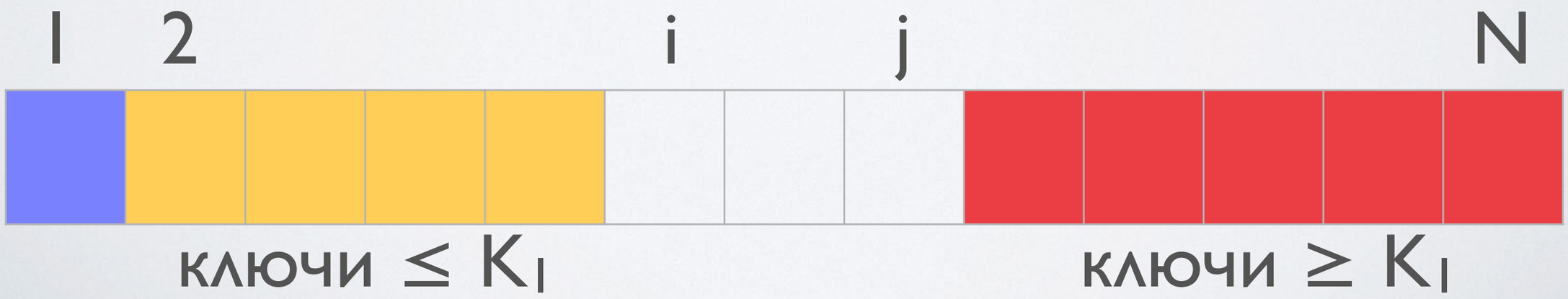
- **Разделение**. Массив путем переупорядочения элементов разбивается на две части $[R_1, \ldots, R_{q-1}]$ и $[R_{q+1}, \ldots, R_N]$. При этом запись R_q «стоит на своем месте», т.е. все ключи левой части не больше K_q, а все ключи правой части – не меньше.

- **Покорение**. Процедура Quicksort вызывается рекурсивно для левой и правой частей.

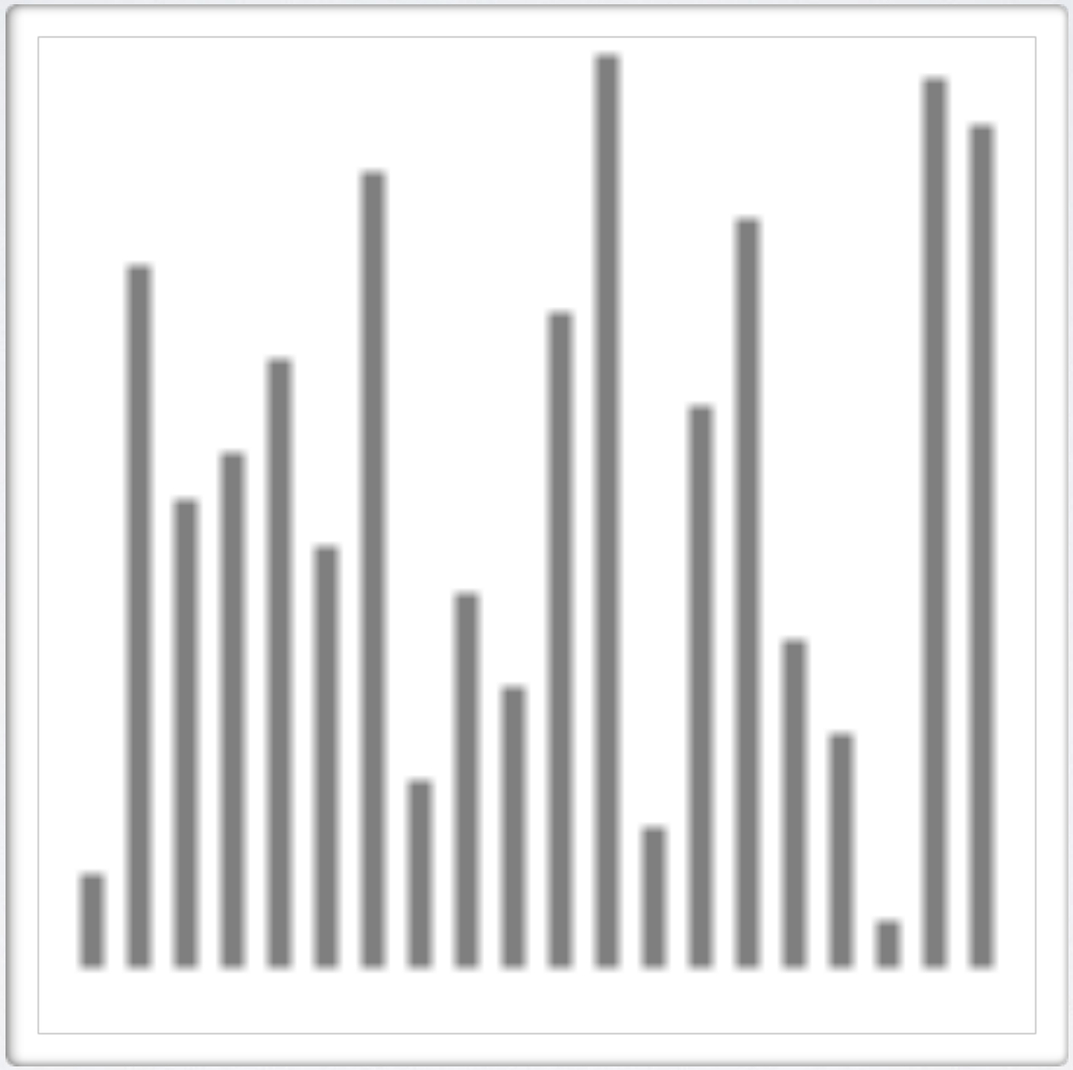
- **Комбинирование**. Не требуется.

ПРОЦЕДУРА РАЗДЕЛЕНИЯ

- Выбираем *разделяющий элемент* (K_1). Он в конце станет K_q.
- Два указателя: $i = 2$ и $j = N$.
    1. Увеличиваем i, пока не найдем $K_i \geq K_1$.
    2. Уменьшаем j, пока не найдем $K_j \leq K_1$.
    3. Если $i < j$, меняем местами R_i и R_j, увеличиваем i, уменьшаем j и продолжаем просмотр (пока не станет $i \geq j$).
- Меняем местами R_j и R_1.



АНИМАЦИЯ



http://www.sorting-algorithms.com/quick-sort

ОСНОВНАЯ ТЕОРЕМА О РЕКУРРЕНТНЫХ СООТНОШЕНИЯХ (THE MASTER METHOD)

количество подзадач

сложность разделения и объединения

Пусть $T(n) = a\,T(n/b) + O(n^d)$, $a \geq 1$, $b \geq 2$, $d \geq 0$

размер каждой подзадачи

$T(n) = O(n^d \log n)$, если $a = b^d$

$T(n) = O(n^d)$, если $a < b^d$

$T(n) = O(n^{\log a / \log b})$, если $a > b^d$

БЫСТРОДЕЙСТВИЕ QUICKSORT

- Удачный разделяющий элемент:
  - $T(N) = 2\,T(N/2) + O(N) = O(N \log N)$,
- В худшем случае:
  - $T(N) = O(N^2)$.

ВЫБОР РАЗДЕЛЯЮЩЕГО ЭЛЕМЕНТА

- Как выбирать?
  - Выбирать случайный элемент.
    - Или перед сортировкой можно перемешать весь массив (например, тасованием Фишера–Йетса).
  - Выбрать медиану малого подмассива (например, из первого, последнего и серединного элементов).
- Но в худшем случае все-равно **O(N²)**.

QUICKSORT НА СОБЕСЕДОВАНИИ

АЛГОРИТМ ВЫБОРА

- Задача состоит в поиске **k**-го по величине элемента в массиве.
  - Для **k = 1**, **k = N**, задача решает очевидно за **O(N)**.
- Можно отсортировать и взять **k**-ый: в среднем **O(N log N)**.
- Можно ли за **O(N)**?

ЛИНЕЙНЫЙ АЛГОРИТМ ВЫБОРА

- **Разделение**. Разделяем массив аналогично быстрой сортировке.

- **Покорение**. Если **k < q**, то продолжаем выбор **k**-го в левой половине. Если **k > q**, то продолжаем выбор **(k-q)**-го в правой половине. Если **k = q**, то заканчиваем алгоритм.

- **Комбинирование**. Не требуется.

ОСНОВНАЯ ТЕОРЕМА О РЕКУРРЕНТНЫХ СООТНОШЕНИЯХ (THE MASTER METHOD)

количество подзадач

сложность разделения и объединения

Пусть $T(n) = a\,T(n/b) + O(n^d)$, $a \geq 1, b \geq 2, d \geq 0$

размер каждой подзадачи

$T(n) = O(n^d \log n)$, если $a = b^d$

$T(n) = O(n^d)$, если $a < b^d$

$T(n) = O(n^{\log a / \log b})$, если $a > b^d$

СЛОЖНОСТЬ ЛИНЕЙНОГО АЛГОРИТМА ВЫБОРА

- В среднем **O(N)**, в худшем случае **O(N^2)**.
  - Все проблемы аналогичны проблемам быстрой сортировки.
- Существует алгоритм BFPRT работающий за **O(N)** в худшем случае.

ПРОБЛЕМА СТАБИЛЬНОСТИ

- Неотсортированный массив:
  [Юля – 17 лет, Петя – 16 лет, Коля – 18 лет, Вася – 17 лет]

- Стабильная сортировка по возрасту:
  [Петя – 16 лет, Юля – 17 лет, Вася – 17 лет, Коля – 18 лет]

- Нестабильная сортировка по возрасту:
  [Петя – 16 лет, Вася – 17 лет, Юля – 17 лет, Коля – 18 лет]

СТАБИЛЬНОСТЬ СОРТИРОВОК

- Стабильная сортировка не меняет относительный порядок элементов с равными ключами.

  - Стабильные: выбором, пузырьковая, вставками.

  - Нестабильные: bozosort, bogosort, быстрая.

СОРТИРОВКА СЛИЯНИЕМ (MERGESORT)

- **Разделение**. Как-либо делим массив на 2 равные части.

- **Покорение**. Для каждой из частей размером больше 1 алгоритм вызывается рекурсивно.

- **Комбинирование**. Линейное слияние отсортированных подмассивов.

СОРТИРОВКА СЛИЯНИЕМ (MERGESORT)

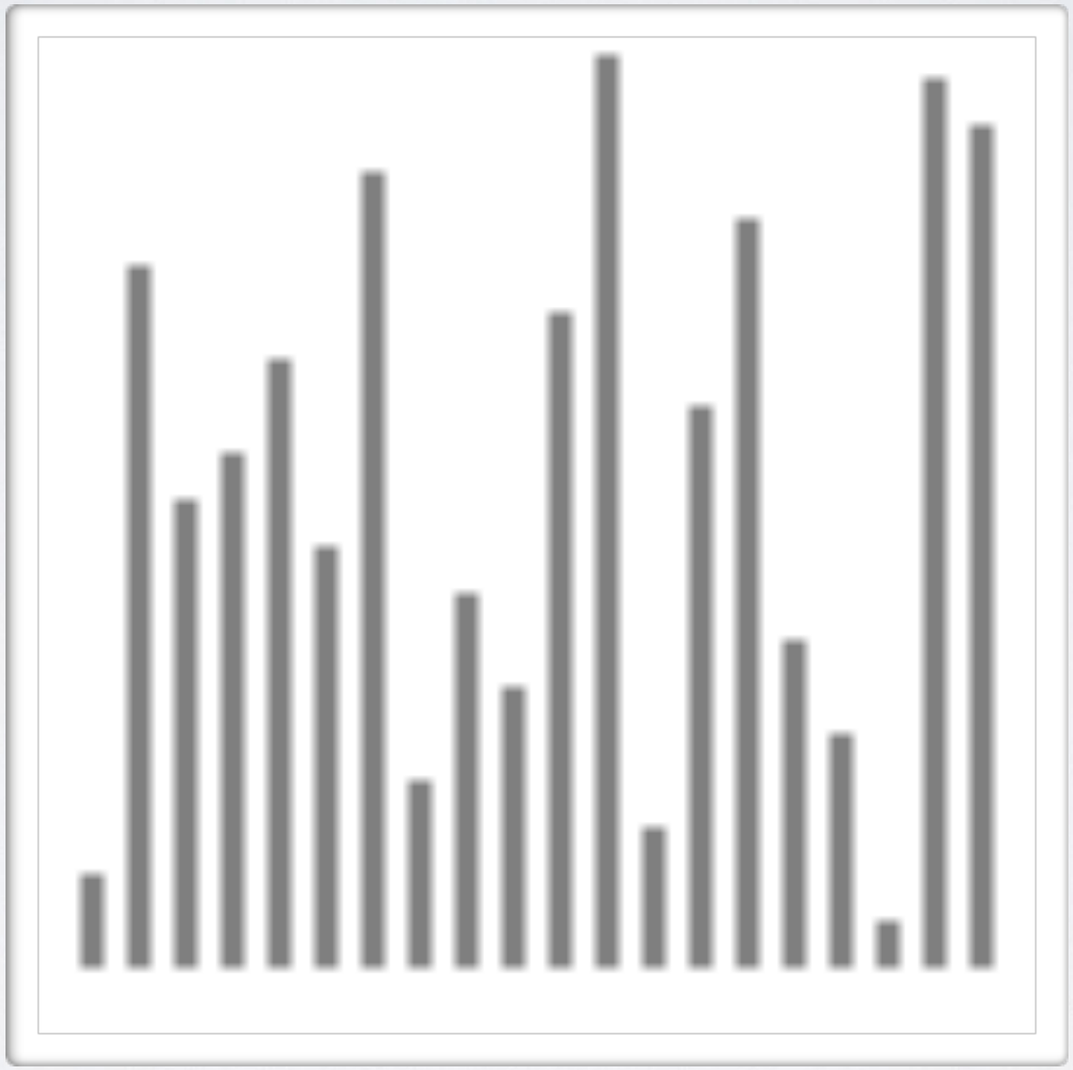
Линейный алгоритм слияния 2 отсортированных массивов:

- Заводим 2 индекса, изначально указывающих на начала массивов.
- На каждом шаге выбираем наименьший элемент из двух, на которые указывают индексы, записываем его в результирующий массив и сдвигаем этот индекс. Если соответствующий массив кончился, отбрасываем индекс вместе с массивом.
- Когда все массивы кончились, в результирующем массиве лежат все элементы, отсортированные.

АНАЛИЗ СОРТИРОВКИ СЛИЯНИЕМ

- Сложность в худшем случае **O(N log N)**.
- Требуется дополнительная память (еще один массив размером **N**).
- Осуществляет доступ к сортируемым элементам *последовательно*. Поэтому годится для сортировки связанных списков.
- Является *стабильной*.

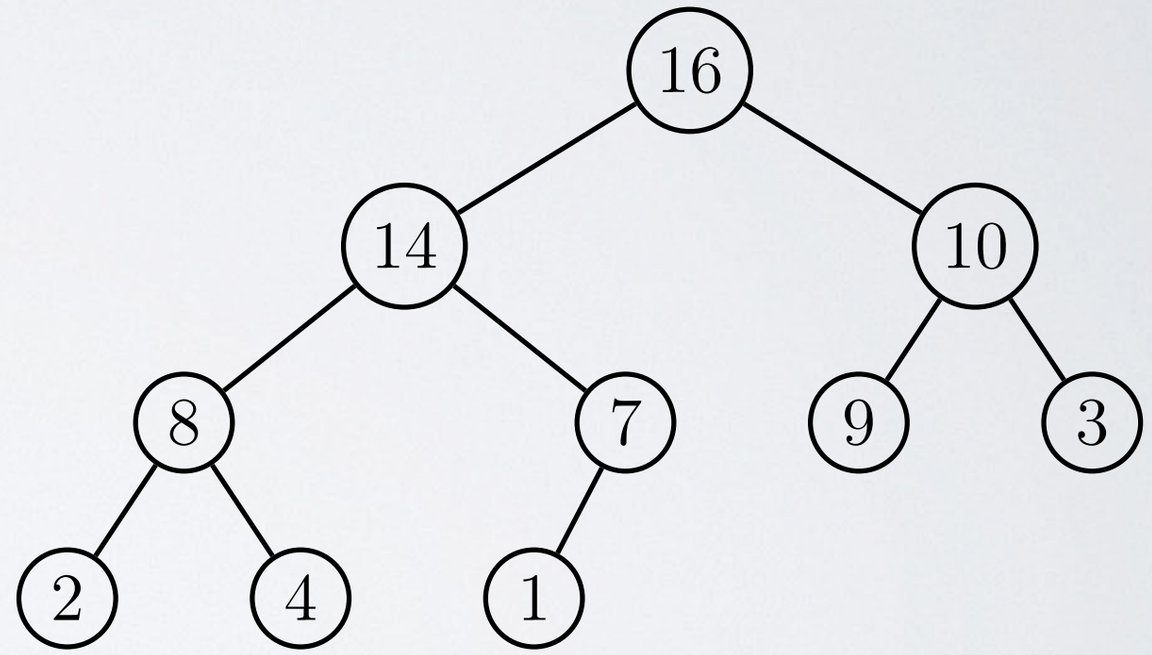
АНИМАЦИЯ



http://www.sorting-algorithms.com/merge-sort

ПИРАМИДАЛЬНАЯ СОРТИРОВКА (HEAPSORT)

- Пирамида – массив, в котором для каждого элемента **n** выполнено:
  - $K_n \geq K_{2n}$ (если $2n \leq N$).
  - $K_n \geq K_{2n+1}$ (если $2n+1 \leq N$).
- Пример: 16, 14, 10, 8, 7, 9, 3, 2, 4, 1.



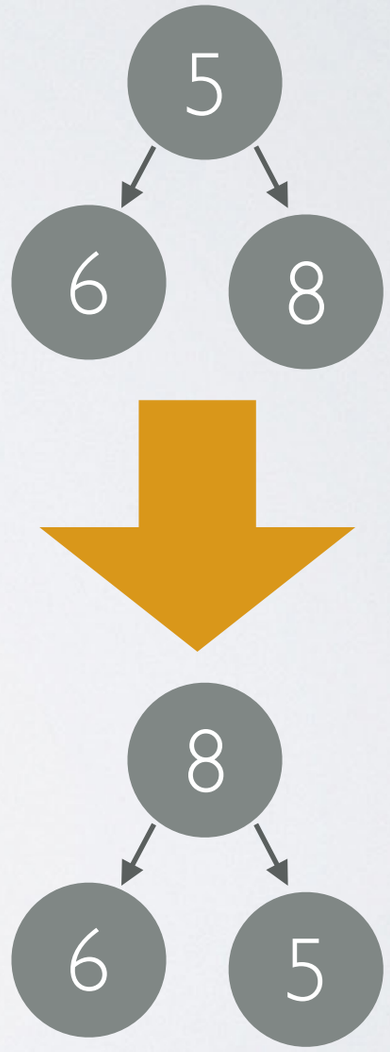
- Алгоритм сортировки:
  - Из массива делаем пирамиду (перестановками).
  - Из пирамиды делаем отсортированный массив (перестановками).

УТОПЛЕНИЕ ЭЛЕМЕНТА (SINK)

Если элемент K_n не удовлетворяет условиям пирамиды:

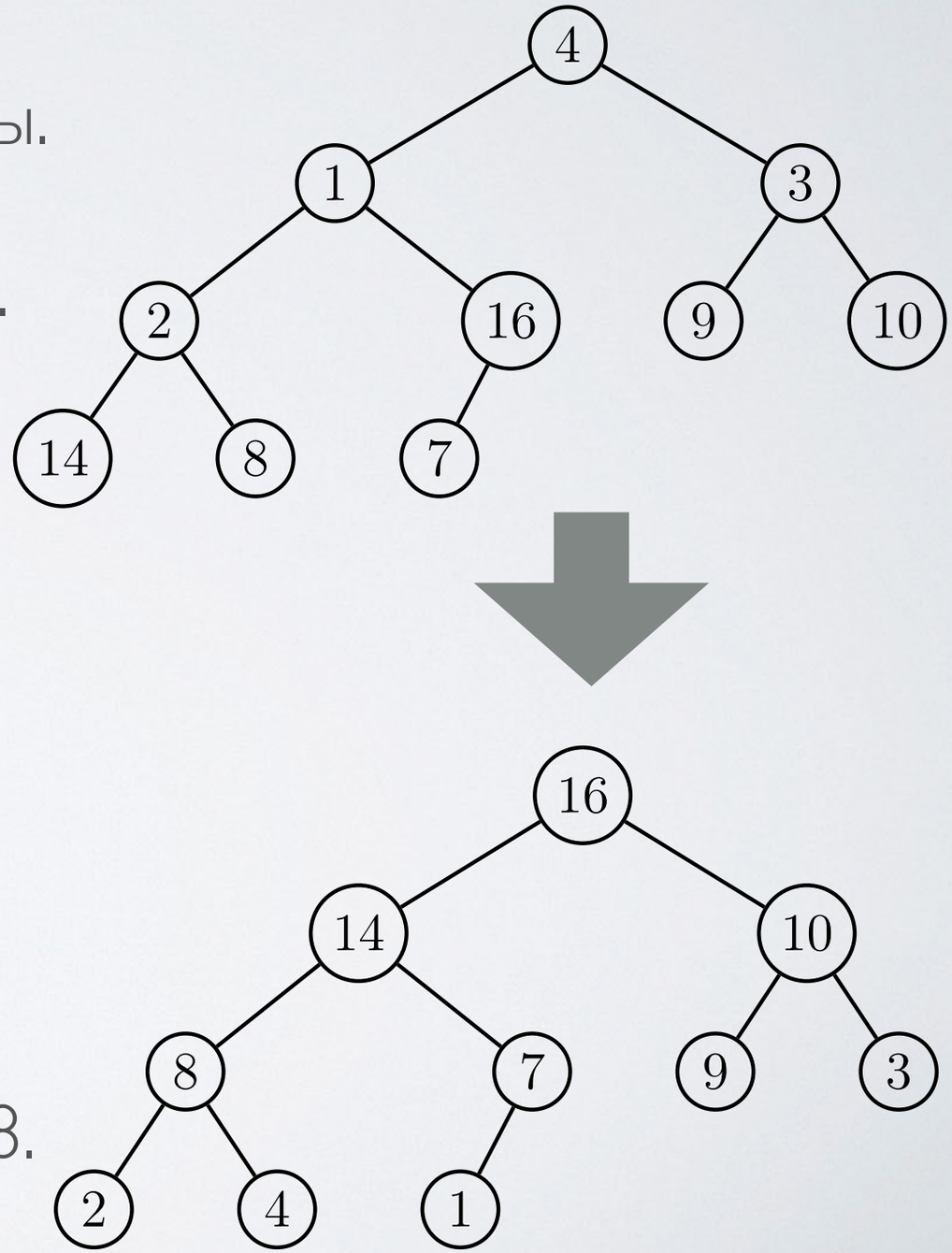
- Находим $X = \max(K_{2n}, K_{2n+1})$.
- Меняем местами K_n и X.
- При необходимости повторяем процедуру для нового положения K_n.

Если для всех узлов K_m, $m > n$, пирамида корректна, то для K_n тоже.



I. ПОСТРОЕНИЕ ПИРАМИДЫ

- Идем с конца массива и топим элементы.
- Данный массив: 4, 1, 3, 2, 16, 9, 10, 14, 8, 7.
- Вторая половина массива (9, 10, 14, 8, 7) удовлетворяет условию пирамиды.
- Проблема с 2: меняем 2 ↔ 14.
- Проблема с 3: меняем 3 ↔ 10.
- Проблема с 1: меняем 1 ↔ 16 ↔ 7.
- Проблема с 4: меняем 4 ↔ 16 ↔ 14 ↔ 8.



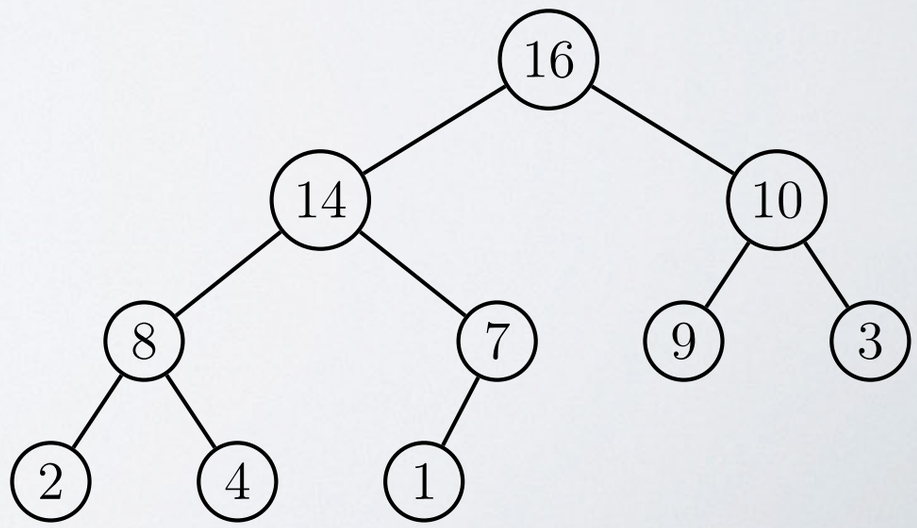
2. ПОЛУЧЕНИЕ МАССИВА

- По условию пирамиды наибольший элемент находится на первом месте. Меняем его местами с последним (где он должен быть), укорачиваем пирамиду на 1 и восстанавливаем ее, утопляя элемент, оказавшийся на вершине.

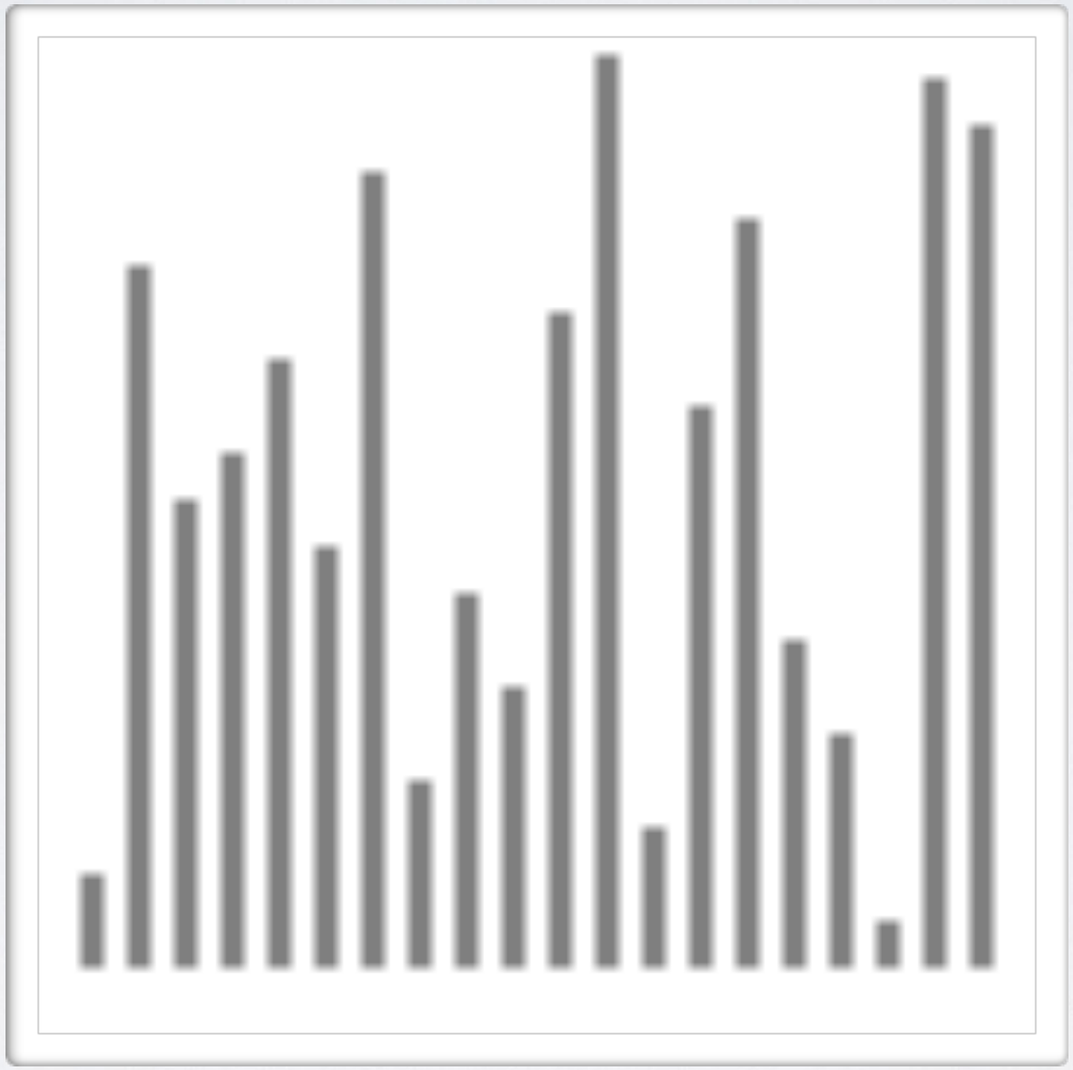
- Повторяем процесс, пока в пирамиде не останется один элемент.

- 16 ↔ 1. Затем 1 ↔ 14 ↔ 8 ↔ 4.

  В отсортированной части будет 16.
  На вершине останется 14.



АНИМАЦИЯ



http://www.sorting-algorithms.com/heap-sort

АНАЛИЗ ПИРАМИДАЛЬНОЙ СОРТИРОВКИ

- Утопляем элемент в худшем случае за **O(log N)**.
- Для построения пирамиды утопляем **N/2** элементов.
- Для получения отсортированного массива **N** раз утопляем вершину.
- Итого: **O(N log N)** в худшем случае.
- Трудоемкая реализация.

СОРТИРОВКИ СРАВНЕНИЯМИ

Сортировка, основанная на сравнениях – сортировка, оперирующая сравнением двух элементов и их перестановкой.

Может ли такая сортировка в худшем случае работать быстрее, чем за **O(N log N)**?

Нет, время работы в худшем случае любой сортировки, основанной на сравнениях: **Ω(N log N)**.

$\Omega(N \log N)$

- Зафиксируем некоторую сортировку и входной массив длины **N**, содержащий числа **1, 2, … N** в некотором порядке.

- Вариантов входного массива **N!**.

- Предположим, что сортировка во время своей работы делает $\leq$ **k** операций сравнений.

- Значит, у алгоритма возможно максимум 2^k различных путей исполнения.

$\Omega(N \log N)$

Так как алгоритм корректно сортирует все входные массивы за **k** сравнений, то 2^k должно быть не меньше **N!**.

$$2^k \geq N!$$

$N! \approx (2\pi N)^{1/2}(N/e)^N$ (формула Стирлинга)

$k = \Omega(\frac{1}{2} \log N + N(\log N - \log e)) = \Omega(N \log N)$

ИНТЕРВЬЮ БАРАКА ОБАМЫ В GOOGLE

Eric Schmidt: What is the most efficient way to sort a million 32-bit integers?

Barack Obama: Well...

Eric Schmidt: Maybe–I'm sorry...

Barack Obama: No, no, no, no. I think–I think the bubble sort would be the wrong way to go.

СОРТИРОВКА ЗА ЛИНЕЙНОЕ ВРЕМЯ

- Для произвольных ключей доказали:
  количество сравнений не менее **N log N**.

- А что, если ключи не произвольные?

- Сколько операций нужно, чтобы отсортировать последовательность битов?

0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 0 1 1 1 0 0 1 0 0 1 1 1 1

СОРТИРОВКА ПОДСЧЕТОМ (COUNTING SORT)

- Предположим, что все ключи – целые числа **0…K−1**.
- Заведем массив счетчиков **C[k]** размером **K**, изначально заполненный нулями.
- Подсчитаем количество вхождений каждого ключа.
- Добавим к каждому элементу **C[k]** все предыдущие, получая: **C[k]** = количество ключей, не превышающих **k**.
- Идем с конца массива элементов, ставя элемент **A[n]** на позицию **C[Key(A[n])]−1** и уменьшая счетчик.

СВОЙСТВА СОРТИРОВКИ ПОДСЧЕТОМ

- Временная сложность **O(N+K)**.
- *Стабильность.*
- Дополнительная память:
  - вспомогательный массив длины N
    (если стабильность не нужна, можно обойтись без него);
  - массив счетчиков.

ПОРАЗРЯДНАЯ СОРТИРОВКА (RADIX SORT)

- Предположим, что ключи состоят из **d** «цифр».
- Алгоритм «от младшей к старшей»:
  - Стабильная сортировка массива по 1-й цифре (младшей).
  - … по 2-й цифре.
  - …
  - … по **d**-й цифре (старшей).
- Годится для сортировки по составным ключам.
- Сложность **O(N)**.

КАРМАННАЯ СОРТИРОВКА

- Предположим, что ключи – **N** равномерно распределенных в интервале **[0,1)** чисел.
- Поделим интервал на **N** подинтервалов (*карманы*).
- Раскидаем элементы по карманам.
- Отсортируем элементы в каждом кармане отдельно (сортировка вставками).
- Последовательно выложим содержимое карманов.
- В среднем работает за **O(N)**.

СОРТИРОВКИ НА ПРАКТИКЕ

- В прикладных программах достаточно стандартной библиотеки.
- Серьезные алгоритмы зачастую представляют смесь:
  - быстрая + вставками (C, C++),
  - быстрая + пирамидальная (Introsort)
  - слиянием + вставками (Timsort) (Python, Java, Android),
- Используйте алгоритм, подходящий к вашей задаче.

ЗАДАЧКА О СОРТИРОВКАХ

Даны 7 массивов чисел:

1. `C 1 4 F 2 B 8 D E 7 A 9 0 6 3 5`;
2. `1 4 2 B 8 C D 7 A 9 0 6 3 5 E F`;
3. `D C B 5 A 9 8 3 1 7 2 4 0 6 E F`;
4. `0 1 2 3 4 B 8 D E 7 A 9 C 6 F 5`;
5. `1 2 4 8 B C F D E 7 A 9 0 6 3 5`;
6. `0 1 4 5 2 B 8 3 6 7 A 9 C E D F`;
7. `0 1 2 3 4 5 6 7 8 9 A B C D E F`.

Массив 1 — начальный.

Массив 7 — отсортированный.

Массивы 2-6 являются промежуточными состояниями при сортировке начального массива разными алгоритмами:

a. сортировка выбором,
b. пузырьковая сортировка (обычная, слева направо),
c. сортировка вставками,
d. быстрая сортировка (без перемешивания, первый элемент как медиана),
e. пирамидальная сортировка.

Определить, какой массив 2-6 соответствует какой сортировке a-e. Ответа недостаточно, нужно объяснение, почему именно этот массив может соответствовать именно этой сортировке.

Подсказка: нужно заметить характерные свойства каждой из сортировок.

КОНЕЦ ЧЕТВЕРТОЙ ЛЕКЦИИ