СОРТИРОВКА

Глотова Т.В.

- **В общем случае сортировку следует понимать как процесс перегруппировки заданного множества объектов в некотором определённом порядке.**
- Цель сортировки – облегчить последующий поиск элементов в таком отсортированном (упорядоченном) множестве.

- Сортировка является одной из фундаментальных алгоритмических задач программирования. Сортировка применяется во всех без исключения областях программирования, будь то базы данных или математические программы. Алгоритмы сортировки имеют большое практическое применение.
- Решению проблем, связанных с сортировкой, посвящено множество научных исследований, разработано множество алгоритмов.

Алгоритм сортировки

- **Алгоритмом сортировки** называется алгоритм для упорядочения некоторого множества элементов. Обычно под алгоритмом сортировки подразумевают алгоритм упорядочивания множества элементов по возрастанию или убыванию.
- Выбор алгоритма зависит от структуры обрабатываемых данных.

Практически каждый алгоритм сортировки можно разбить на 3 части:

- **сравнение**, определяющее упорядоченность пары элементов;
- **перестановку**, меняющую местами пару элементов;
- **собственно** сортирующий **алгоритм**, который осуществляет сравнение и перестановку элементов до тех пор, пока все элементы множества не будут упорядочены.

- Универсального, наилучшего алгоритма сортировки на данный момент не существует.
- Однако, имея приблизительные характеристики входных данных, можно подобрать метод, работающий оптимальным образом.
- Для этого необходимо знать параметры, по которым будет производиться оценка алгоритмов.

Параметры оценки алгоритмов

- Время сортировки – основной параметр, характеризующий быстродействие алгоритма.

- Память – один из параметров, который характеризуется тем, что ряд алгоритмов сортировки требуют выделения дополнительной памяти под временное хранение данных. При оценке используемой памяти не будет учитываться память, которую занимает исходный массив данных и независящие от входной последовательности затраты, например, на хранение кода программы.

Параметры оценки алгоритмов

- Устойчивость – это параметр, который отвечает за то, что сортировка не меняет взаимного расположения равных элементов.
- Естественность поведения – параметр, которой указывает на эффективность метода при обработке уже отсортированных, или частично отсортированных данных. Алгоритм ведет себя естественно, если учитывает эту характеристику входной последовательности и работает лучше.

Классификация алгоритмов сортировок

- Все разнообразие и многообразие алгоритмов сортировок можно классифицировать по различным признакам:
- по устойчивости,
- по поведению,
- по использованию операций сравнения,
- по потребности в дополнительной памяти,
- по потребности в знаниях о структуре данных, выходящих за рамки операции сравнения, и другие.

Классификация алгоритмов сортировки по сфере применения

- Внутренняя сортировка или сортировка массивов
- Внешняя сортировка или сортировка последовательностей (файлов)

Классификация алгоритмов сортировки по книге «Искусство программирования для ЭВМ. Т.3. Сортировка и поиск», автор Д.Кнут

Внутренняя сортировка

- Это алгоритм сортировки, который в процессе упорядочивания данных использует только оперативную память компьютера. То есть оперативной памяти достаточно для помещения в нее сортируемого массива данных с произвольным доступом к любой ячейке и собственно для выполнения алгоритма.
- Внутренняя сортировка применяется во всех случаях, за исключением однопроходного считывания данных и однопроходной записи отсортированных данных.
- В зависимости от конкретного алгоритма и его реализации данные могут сортироваться в той же области памяти, либо использовать дополнительную оперативную память.

Внешняя сортировка

- Это алгоритм сортировки, который при проведении упорядочивания данных использует внешнюю память, как правило, жесткие диски.
- Внешняя сортировка разработана для обработки больших списков данных, которые не помещаются в оперативную память.
- Обращение к различным носителям накладывает некоторые дополнительные ограничения на данный алгоритм: доступ к носителю осуществляется последовательным образом, то есть в каждый момент времени можно считать или записать только элемент, следующий за текущим.

- Внутренняя сортировка является базовой для любого алгоритма внешней сортировки – отдельные части массива данных сортируются в оперативной памяти и с помощью специального алгоритма сцепляются в один массив, упорядоченный по ключу.
- Следует отметить, что внутренняя сортировка значительно эффективней внешней, так как на обращение к оперативной памяти затрачивается намного меньше времени, чем к носителям.

Перечислим наиболее известные алгоритмы внутренней сортировки:

- 1. Сортировка вставками (или прямого включения).
  2. Обменная сортировка.
  3. Сортировка посредством выбора.
  4. Сортировка подсчётом.
  5. Сортировка слиянием (на линейных списках).
  6. Распределяющая сортировка.

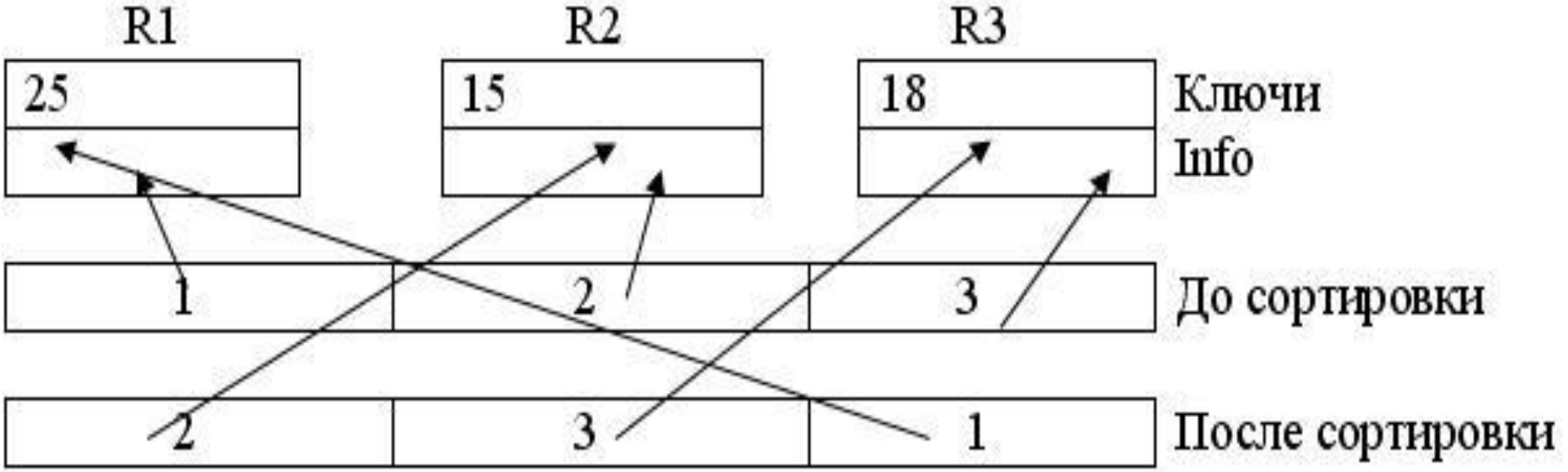
Введём некоторые понятия и обозначения

- Если у нас есть элементы $\mathbf{a_1, a_2, \ldots, a_n}$, то **сортировка есть перестановка** этих элементов $\mathbf{a_{k1}, a_{k2}, \ldots, a_{kn}}$,
- где при некоторой упорядочивающей функции f выполняются отношения
- $\mathbf{f(a_{k1}) \le f(a_{k2}) \le \ldots \le f(a_{kn}).}$

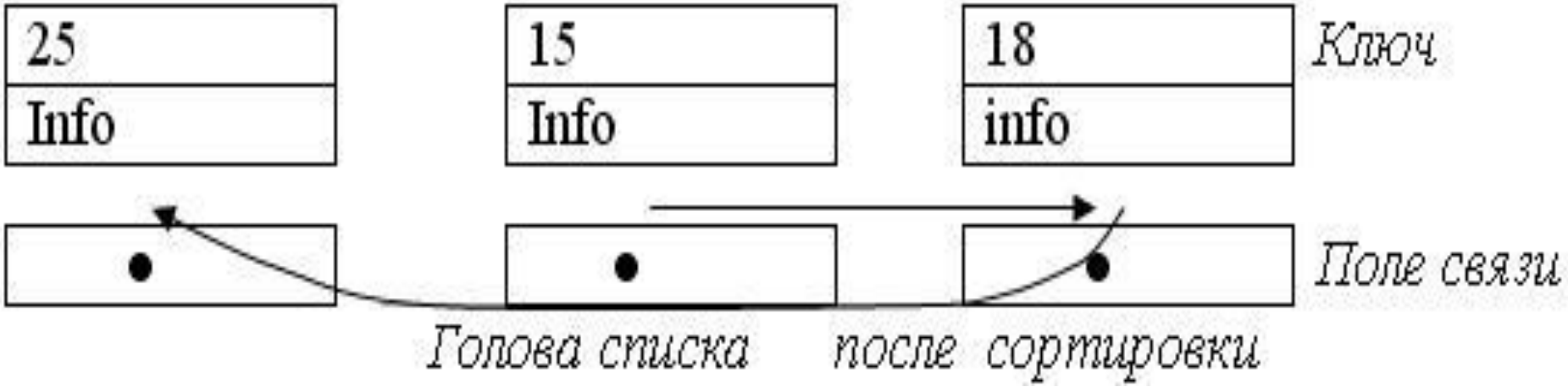
- Обычно упорядочивающая функция не выполняется по какому-либо правилу, а хранится как явная компонента (поле) каждого элемента.
- Её значение называется ключом (key) элемента. Поэтому для представления элементов хорошо подходят такие структуры данных как записи
- *struct Item {*
  int Key;
  /… другие поля или компоненты*/*
  };

- В одних случаях необходимо размещать записи в памяти так, чтобы их ключи были упорядочены; в других случаях можно обойтись вспомогательной таблицей некоторого вида, которая определяет перестановку. Если записи и/или ключи занимают несколько ячеек (слов) памяти, то часто лучше составить новую таблицу адресов (ссылок), которые указывают на записи, и работать с этими адресами, не перемещая громоздкие записи.
- Такой метод называется *сортировкой таблицы адресов (индексов)*
- Если ключи короткие, а сопутствующая информация в записях велика, то для повышения скорости ключи можно вынести в таблицу адресов. Это называется *сортировкой ключей.*

Сортировка таблицы адресов



- Другие схемы сортировки используют вспомогательное поле связи, которое включается в каждую запись. Связи обрабатываются таким образом, что в результате все записи оказываются связанными в линейный список, в котором каждая связь указывает на следующую по порядку запись. Это называют сортировкой списка



После сортировки таблицы адресов или сортировки списка можно либо расположить сами записи в заданном порядке (в данном примере неубывающем). Для этого имеется несколько способов:

- Используя одну дополнительную ячейку памяти.
- Используя дополнительную память размером равную исходному массиву.

Сортировка подсчетом

- Идея алгоритма состоит в том, чтобы сравнить попарно все ключи и подсчитать, сколько из них больше (или меньше) каждого отдельного ключа.
- Далее, при увеличении каждого из полученных значений на 1, мы получим данные о том, где должны располагаться упорядоченные элементы массива.
- Знак больше соответствует сортировке по возрастанию, а меньше – по убыванию. N – число записей (ключей-элементов).
- ***N(N-1)/2*** – число сравнений.

Алгоритм сортировки подсчетом

- 1. *Начало алгоритма.*
  2. В массив счетчиков Count записать 1.
  3. Цикл 1 i=0, N-2. Выполнять пункт 4 .
  4. Цикл 2 j=i+1, N-1. Выполнять пункт 5.
  5. Если K[i]>K[j], тогда Count[i]+=1, иначе Count[j]+=1.
  6. Цикл i=0,N-1 выполнить
  newK [Count[i]]= K[i].
  7. Конец алгоритма.

По окончании счетчики содержат номера позиций, на которых должны стоять элементы в упорядоченном массиве.

Пример работы алгоритма

K	Count					newK
	Шаг 0	Шаг 1 Эл-т 5	Шаг 2 Эл-т 8	Шаг 3 Эл-т 10	Шаг 4 Эл-т 3	
5	1	2	2	2	2	3
8	1	2	4	4	4	5
10	1	2	3	5	5	6
3	1	1	1	1	1	8
6	1	2	2	2	3	10

- Для работы нужен дополнительный массив размером N - счётчик Count [N-1]. В этом алгоритме записи не перемещаются. Count [i] указывает на место, куда нужно переслать запись.
- Перемещение записи производится с использованием дополнительной памяти:

  Цикл i =0,N-1 выполнять newK[Count[i]]= K[i], K-исходный массив
-

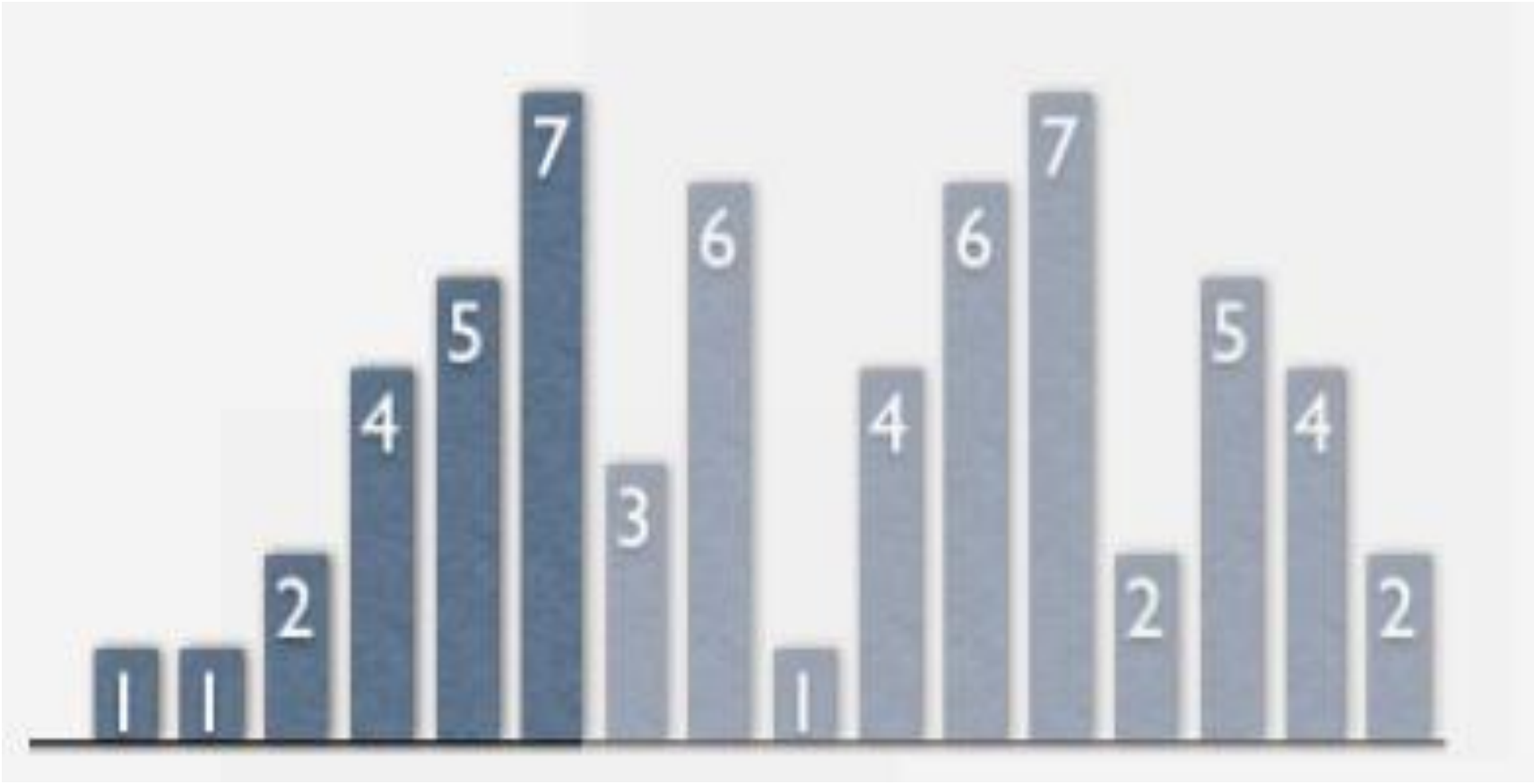
- Временная сложность этого алгоритма приведена в [Кнут].
- $O(t) = 3N^2 + 10\ N$ до $5{,}5N^2 + 7{,}5N$.
- $O(t) = C\ N^2$
- Эта сортировка впервые упоминается в работе Фрэнда(E.H.Friend) в 1956 году.

Сортировки вставками (insert sort)

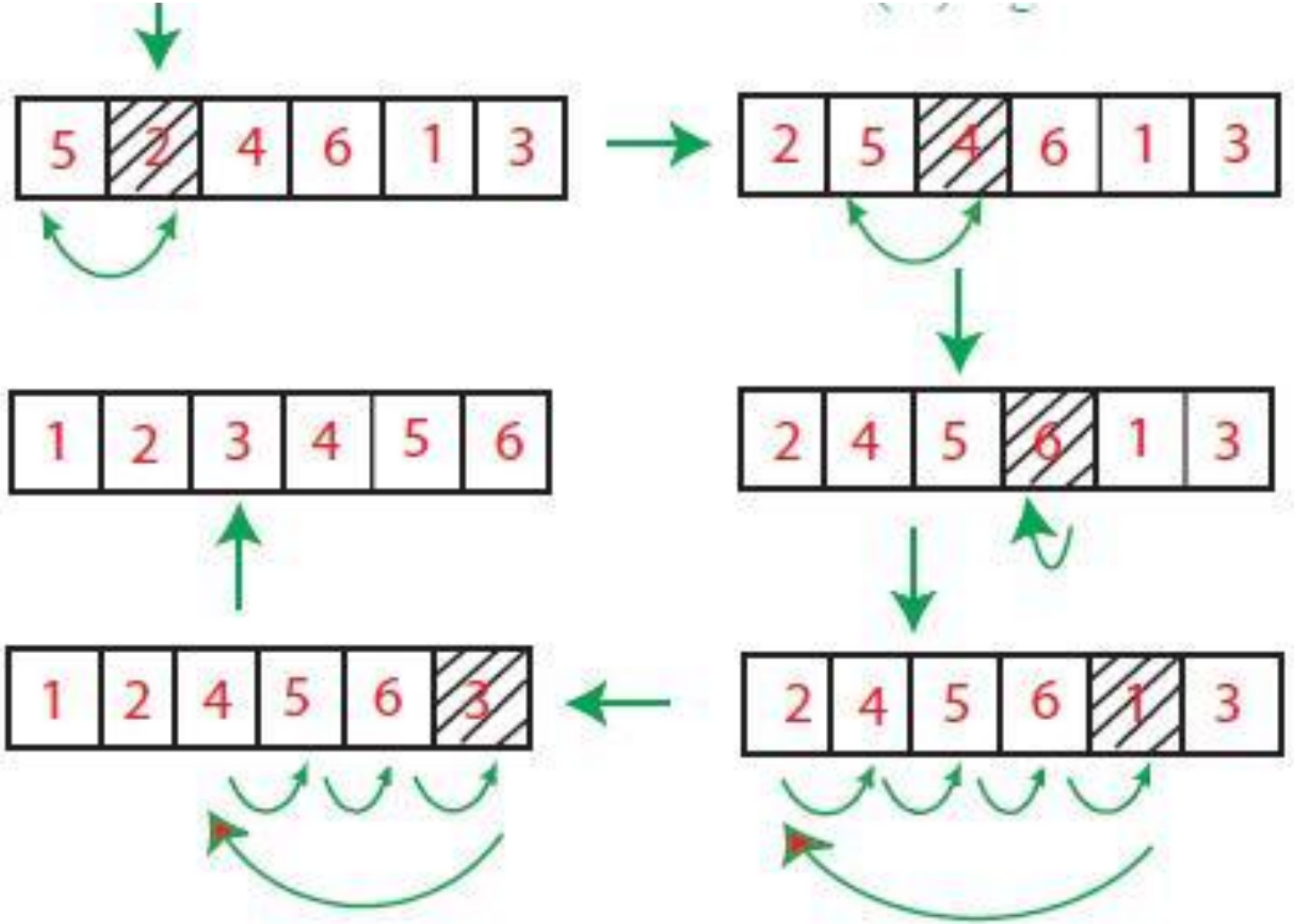
Простые вставки

- Пусть **1 < j ≤ N** и записи $R_1 \ldots R_{j-1}$ уже размещены так, что их ключи упорядочены: $К_1 \leq К_2 \leq \ldots \leq К_{j-1}$.
- Будем сравнивать по очереди: $К_j$ **с** $К_{j-1}, К_{j-2}, \ldots$ до тех пор пока не обнаружим, что запись R_j следует вставить между R_i и R_{i+1}:
- тогда подвинем записи $R_{i+1}, \ldots, R_{j-1}$ на одно место и поместим новую запись в позицию **i+1**.

Сортировки вставками
(insert sort)



Пример сортировки простыми вставками



Алгоритм сортировки вставками

- 1. Начало алгоритма.
  2. Цикл j = 1, N-1. Выполнить пункты (3-6)
  3. Установить значения: i = j-1, key =K[j].
  4. Пока (i>0) && (key<K[i]) выполнять 5.
  5. K[i+1] = K[i]; i = i - 1.
  6. K[i+1] = key.
  7. Конец алгоритма.

Бинарные вставки

- Когда при сортировке простыми вставками обрабатывается j-я запись, её ключ сравнивается в среднем примерно с ***j/2*** ранее упорядоченными ключами, поэтому общее число сравнений равно ***(1+2+…+N)/2 ≈ N²/4.***
- Если для поиска места вставки в уже упорядоченную последовательность воспользоваться бинарным поиском, то можно значительно сократить число сравнений.

- Этот метод называется бинарными вставками, он был упомянут в 1946 году Джоном Мочли в первой публикации по машинной сортировке.
- Но этот метод снижает только время поиска, но для вставки $\mathbf{R_j}$ придётся передвигать в среднем примерно ***j/2*** ранее отсортированных записей.

Двухпутевые вставки

- Рассматриваются различные усовершенствования, которые позволяет сократить число необходимых перемещений.
- Например: первый элемент помещают в середину области вывода, и место для последующих элементов освобождается при помощи сдвигов влево и вправо. Это позволяет сэкономить примерно половину времени по сравнению с простыми вставками, но алгоритм становится сложнее. Он называется ***двухпутевыми вставками***.
- Для алгоритмов сортировки, которые каждый раз перемещают запись только на одну позицию, среднее время примерно N^2.

Анализ алгоритма

- **O(n^2)** сравнений и перестановок
- **O(1)** дополнительной памяти
- Устойчивый
- Естественность поведения **O(n)**

Сортировка Шелла
Сортировка с убывающим шагом

- Сортировка Шелла была названа в честь ее изобретателя – Дональда Шелла, который опубликовал этот алгоритм в 1959 году.
- Общая идея сортировки Шелла состоит в сравнении на начальных стадиях сортировки пар значений, расположенных достаточно далеко друг от друга в упорядочиваемом наборе данных. Такая модификация метода сортировки позволяет быстро переставлять далекие неупорядоченные пары значений

Общая схема метода

- 1. Происходит упорядочивание элементов n/2 пар ($x_i, x_{n/2+i}$) для $1 < i < n/2$.
- 2. Упорядочиваются элементы в $n/4$ группах из четырех элементов ($x_i, x_{n/4+i}, x_{n/2+i}, x_{3n/4+i}$) для $1 < i < n/4$.
- 3. Упорядочиваются элементы уже в $n/4$ группах из восьми элементов и т.д.
- На последнем шаге упорядочиваются элементы сразу во всем массиве $x_1, x_2, \ldots, x_n$.
- На каждом шаге для упорядочивания элементов в группах используется ***метод сортировки вставками***

- В результате проведенного в [Кнуте] анализа предлагается различные способы задания числа групп, например:

$$T = [\log_2 N]-1 \text{ или } T = [\log_3 N]-1 ,$$

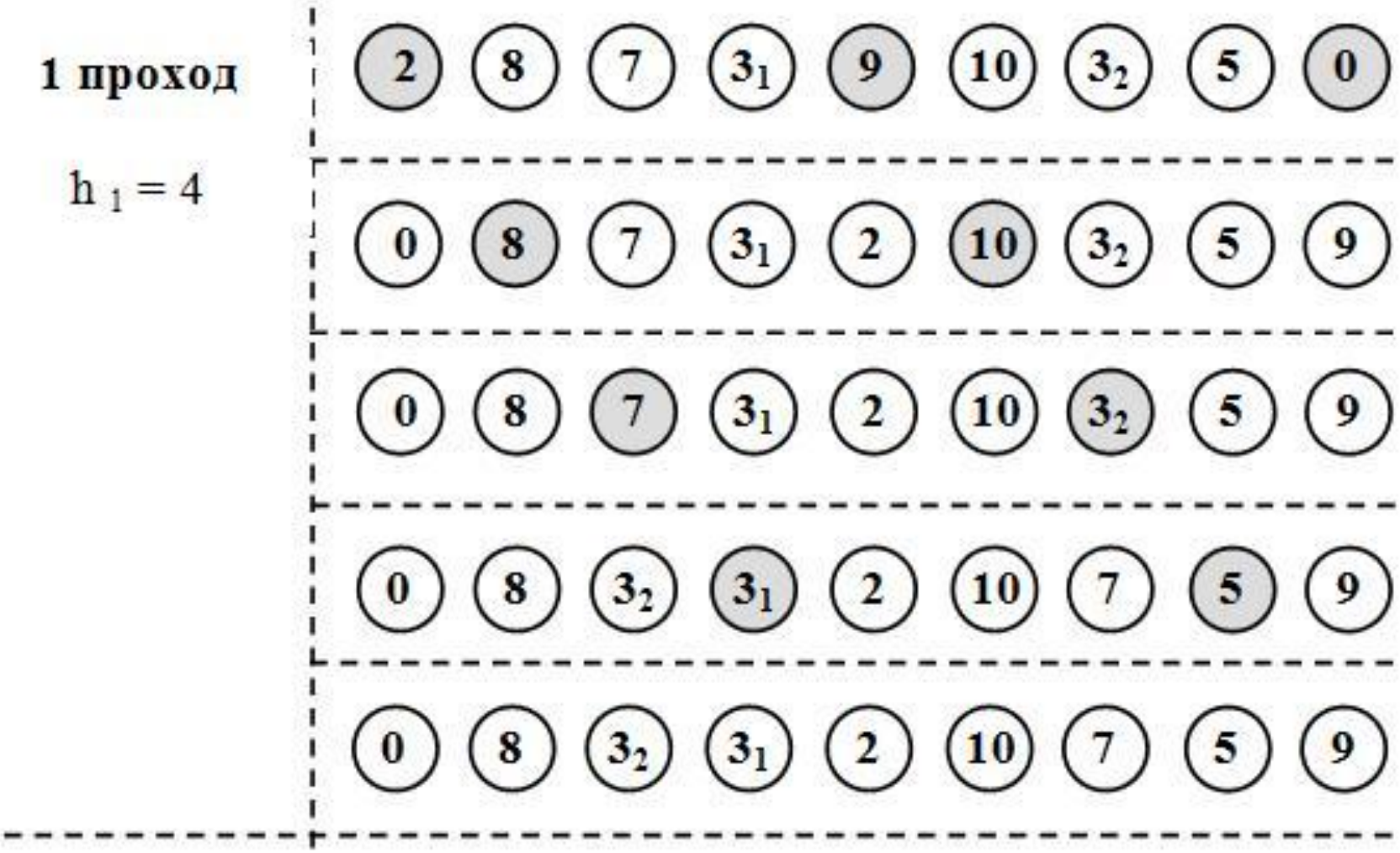
где обозначение *[k]* соответствует наибольшему целому значению меньшему или равному *k*. Для 1 формулы при $N=16$ $T= 3$.
Число записей в каждой из T групп определяется по формуле: $H_1 = 1$, $H_{-1} = 2*H_s + 1$.

Тогда для $T= 3$ имеем последовательность:
1 3 7 вместо: *1 2 4 8*, при $T = 4$: *1 3 7 15* и т.д.

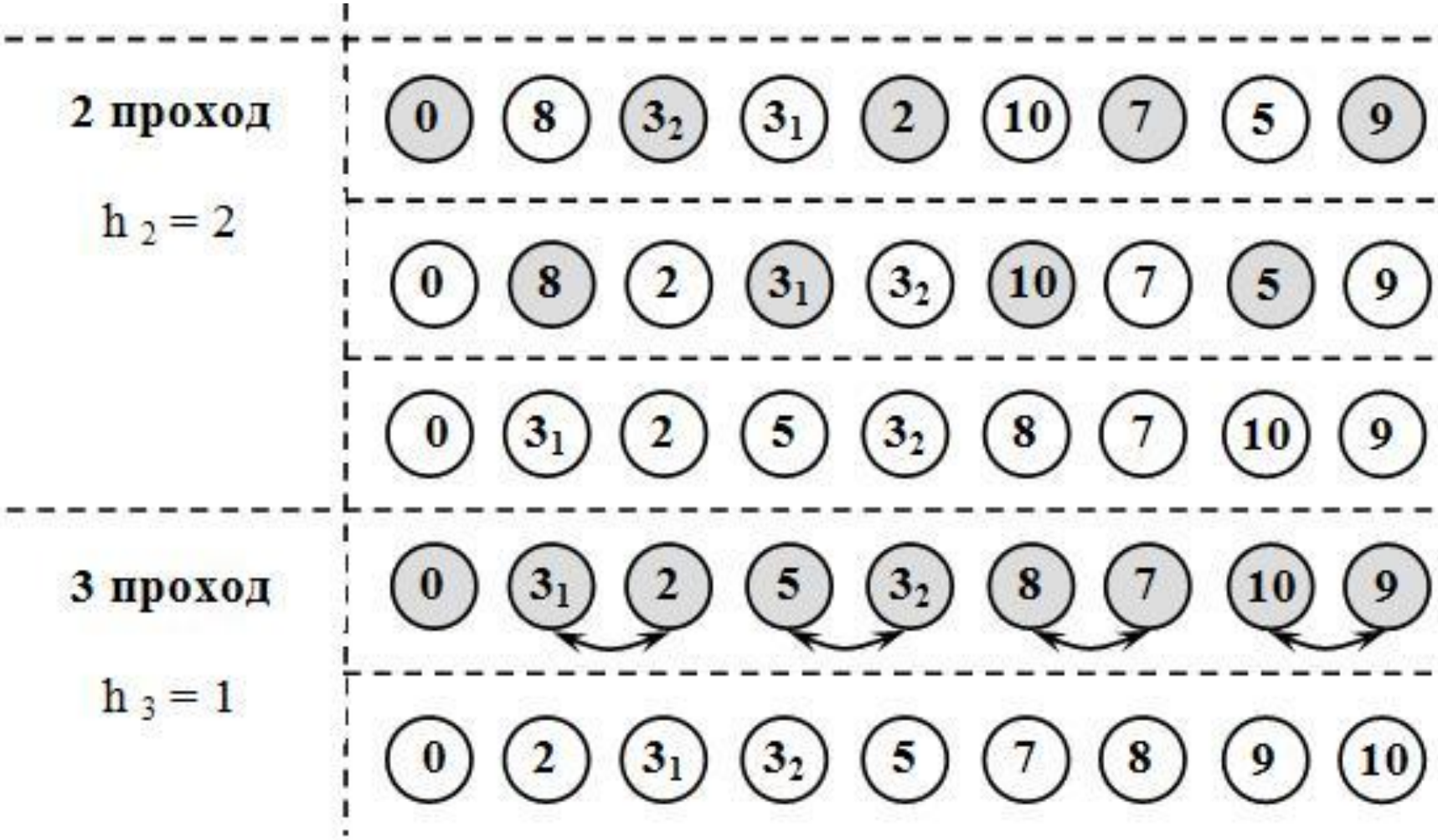
При $N=10000$ $T = [\log_2 10000]-1=12$.

- В настоящее время неизвестна последовательность *hi,hi-1,hi-2,…,h1,,* оптимальность которой доказана.
- Для достаточно больших массивов рекомендуемой считается такая последовательность, что $h_{i+1}=3h_i+1$, *a* $h_1=1$. Начинается процесс с h_m, что $h_m > [n/9]$.
- Иногда значение h вычисляют проще: $h_i+1=h_i/2, h_1=1, h_m=n/2$. Это упрощенное вычисление h и будем использовать в примере .

Пример сортировки Шелла



Пример сортировки Шелла



- 1. Начало алгоритма
  2. Определяем T. H[0]=1.
  3. Цикл 1: s=0,T-1. Выполнять H[s+1]= 2*H[s]+1.
  4. Цикл 2: s=T-1,0. Выполнять пункты 5 – 11.
  5. Ht = H[s].
  6. Цикл 3 j=Ht, N-1. Выполнять пункты 7 – 10 .
  7. i=j-Ht, key=K[j].
  8. Цикл 4 пока i<0 и key<K[i] выполнять п. 9.
  9. K[i+Ht]=K[i], i=i-Ht.
  10. K[i+Ht]=key;
  11. Конец цикла 2.
  12. Конец алгоритма

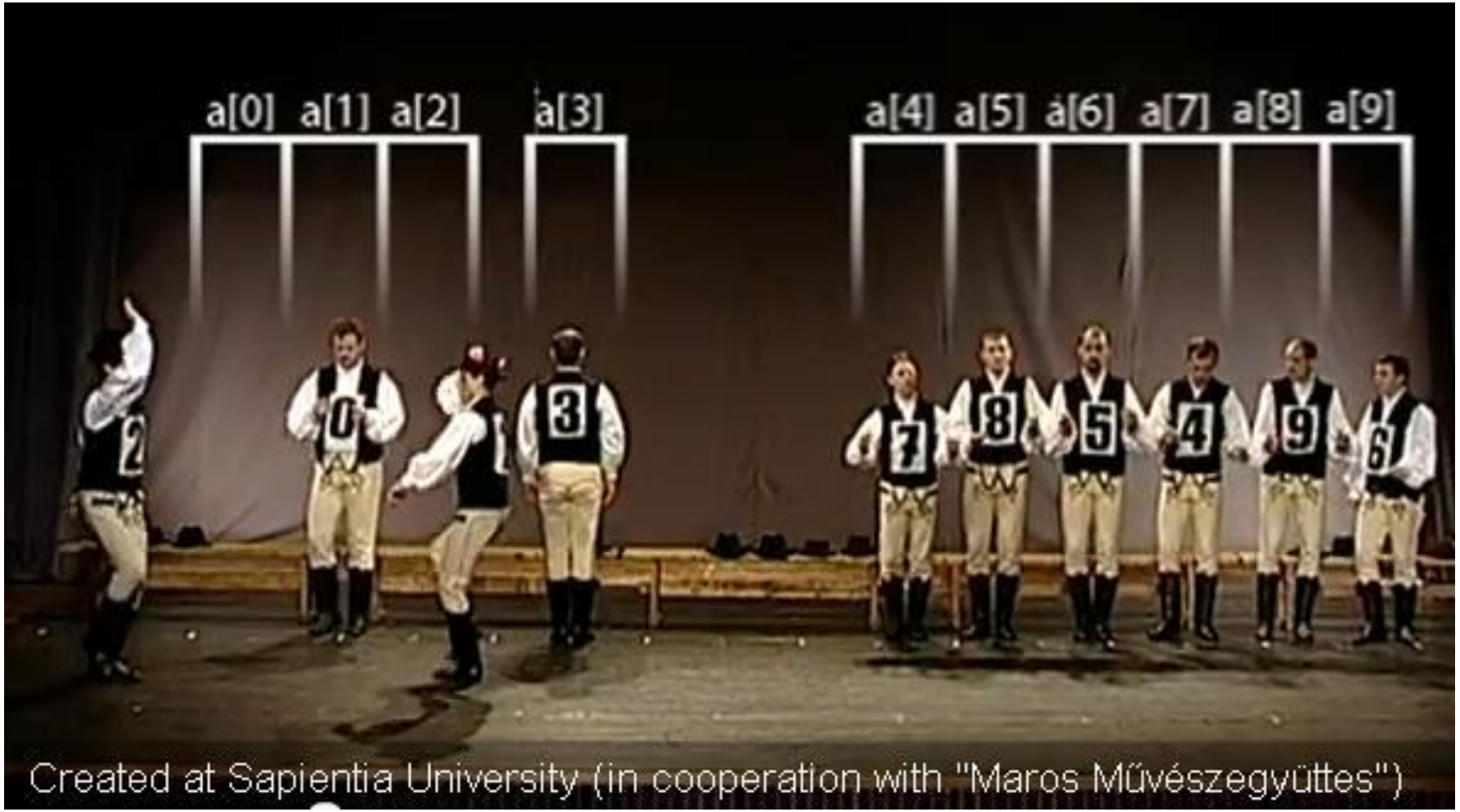
Анализ алгоритма

- **$O(t) \sim N^{1,2} \div N^{1,3}$** зависит от выбора
- **$O(n^{3/2})$** при $t=[\log_2 N]-1$
- **O(1)** дополнительной памяти
- Неустойчивый
- Естественность поведения **$O(n \cdot lg(n))$**

Shell-sort with Hungarian (Székely) folk dance

Bubble-sort with Hungarian ("Csángó") folk dance

Quick-sort with Hungarian (Küküllőmenti legényes) folk dance



Обменные сортировки

- 1. Обменная сортировка с выбором и её усовершенствование – метод пузырька, шейкерная сортировка.
  2. Обменная сортировка со слиянием (Бэтчера) – параллельная.
  3. Обменная с разделением (“быстрая сортировка” Хоара).
  4. Поразрядная обменная сортировка.

Обменная сортировка

- Обменная сортировка основана на последовательном сравнении двух соседних элементов и если они расположены не по порядку, то их обмене местами.
- Сравнение пар соседних элементов производится до тех пор, пока имеются перестановки.
- В результате одного цикла сравнения пар, начиная с первого элемента, последний элемент встает на свое окончательное место, и в следующем цикле не участвует.
- В каждом следующем цикле число просматриваемых пар уменьшается на 1. Такой алгоритм называют ***алгоритмом простого обмена*.**

Пузырьковая сортировка Bubble-sort

- Если процесс сравнения пар начинается с конца последовательности, то такой алгоритм называют пузырьковой сортировкой.
- В этом случае на первом месте оказывается элемент, который в дальнейшем сравнении не участвует.

Шейкерная сортировка

- Усовершенствованием этих алгоритмов является алгоритм «шейкерной» сортировки
- Первый цикл сравнения производится, начиная с начала последовательности, второй цикл, начиная с ее конца.
- Третий цикл начинается со второго элемента, а четвертый - с предпоследнего, и так далее, пока есть обмены.

- Сложность обменной, в том числе шейкерной сортировки: $O(t) \sim cn^2$.

Быстрая сортировка Хоара Quicksort

- Быстрая сортировка – это общее название ряда алгоритмов, которые отражают различные подходы к получению критичного параметра, влияющего на производительность метода.
- Метод основывается на последовательном разделении сортируемого набора данных на блоки меньшего размера таким образом, что между значениями разных блоков обеспечивается отношение упорядоченности (для любой пары блоков все значения одного из этих блоков не превышают значений другого блока).

- Опорным (ведущим) элементом называется некоторый элемент массива, который выбирается определенный образом.
- С точки зрения корректности алгоритма выбор опорного элемента безразличен. С точки зрения повышения эффективности алгоритма выбираться должна медиана, но без дополнительных сведений о сортируемых данных ее обычно невозможно получить.
- Необходимо выбирать постоянно один и тот же элемент (например, средний или последний по положению) или выбирать элемент со случайно выбранным индексом.

Интересно

- что Хоар разработал этот метод применительно к машинному переводу: дело в том, что в то время словарь хранился на магнитной ленте, и если упорядочить все слова в тексте, их переводы можно получить за один прогон ленты.
- Алгоритм был придуман Хоаром во время его пребывания в Советском Союзе, где он обучался в Московском университете компьютерному переводу и занимался разработкой русско-английского разговорника (говорят, этот алгоритм был подслушан им у русских студентов).
- [An Interview with C.A.R. Hoare. Communications of the ACM, March 2009 ("premium content").]

Сортировка Хоара

- В алгоритме используется подход, в котором следующее действие зависит от результата предыдущего сравнения
- Рассмотрим схему сравнений/обменов:
- Необходимо установить 2 указателя адресов i и j. Вначале устанавливаем i=0, j=N-1.
- Сравниваем K[i] и K[j], если обмен не потребуется, то j=j-1 и повторяем сравнение. После первого обмена увеличиваем i на 1 и далее продолжаем сравнения, увеличивая счетчик i , пока не произойдёт следующий обмен, тогда снова j=j-1 и т. д.
- Все эти действия производятся до тех пор, пока i не станет равным j. К тому моменту, когда i == j запись R1 (ключ K1) займет свою окончательную позицию, т. к. слева от него все ключи меньше него, а справа – больше него.
- Теперь к каждой из частей, расположенных слева и справа от установленного ключа, необходимо применить тот же подход.

Алгоритм быстрой сортировки

- Пусть дан массив x[n] размерности n.
- 1. Выбирается опорный элемент массива.
- 2. Массив разбивается на два – левый и правый – относительно опорного элемента. Реорганизуем массив таким образом, чтобы все элементы, меньшие опорного элемента, оказались слева от него, а все элементы, большие опорного – справа от него.
- 3. Далее повторяется шаг 2 для каждого из двух вновь образованных массивов. Каждый раз при повторении преобразования очередная часть массива разбивается на два меньших и т. д., пока не получится массив из двух элементов

Рекурсивный алгоритм быстрой сортировки

- 1. Начало алгоритма.
  2. L=0; r=N; Начало процедуры sort(0,N-1);
  3. s=(L+r)/2; kl:=K[s];
  4. i=L; j=r;
  5. Повторять пункты 6 – 10.
  6. Пока выполняется условие K[i]<kl, повторять i=i+1;
  7. Пока выполняется условие kl<K[j] повторять j=j-1;
  8. Если i<=j, тогда перейти к 9, иначе к 10.
  9. Произвести обмен: z=K[i]; K[i]=K[j]; K[j]=z; i=i+1; j=j-1;
  10.Проверить условие окончания цикла: если i>j, то завершить цикл и перейти к 11, в противном случае возвратиться к 4.
  11. Если L<j, тогда выполнить sort[L,j].
  12. Если i<r, тогда выполнить sort[i,r].
  13. Конец алгоритма.

Таблица 4

Шаг	Значения элементов K								kl	Стек
0	5	10	2	8	6	9	4	3	5	Пусто
1	3	4	2	5	6	9	8	10	3	(5,8)
2	2	3	4	5	6	9	8	10	6	(6,8)
4	2	3	4	5	6	8	9	10		Пусто

Сортировка Хоара

- В алгоритме используется подход, в котором следующее действие зависит от результата предыдущего сравнения
- Рассмотрим схему сравнений/обменов:
- Необходимо установить 2 указателя адресов i и j. Вначале устанавливаем i=0, j=N-1.
- Сравниваем K[i] и K[j], если обмен не потребуется, то j=j-1 и повторяем сравнение. После первого обмена увеличиваем i на 1 и далее продолжаем сравнения, увеличивая счетчик i , пока не произойдёт следующий обмен, тогда снова j=j-1 и т. д.
- Все эти действия производятся до тех пор, пока i не станет равным j. К тому моменту, когда i == j запись R1 (ключ K1) займет свою окончательную позицию, т. к. слева от него все ключи меньше него, а справа – больше него.
- Теперь к каждой из частей, расположенных слева и справа от установленного ключа, необходимо применить тот же подход.

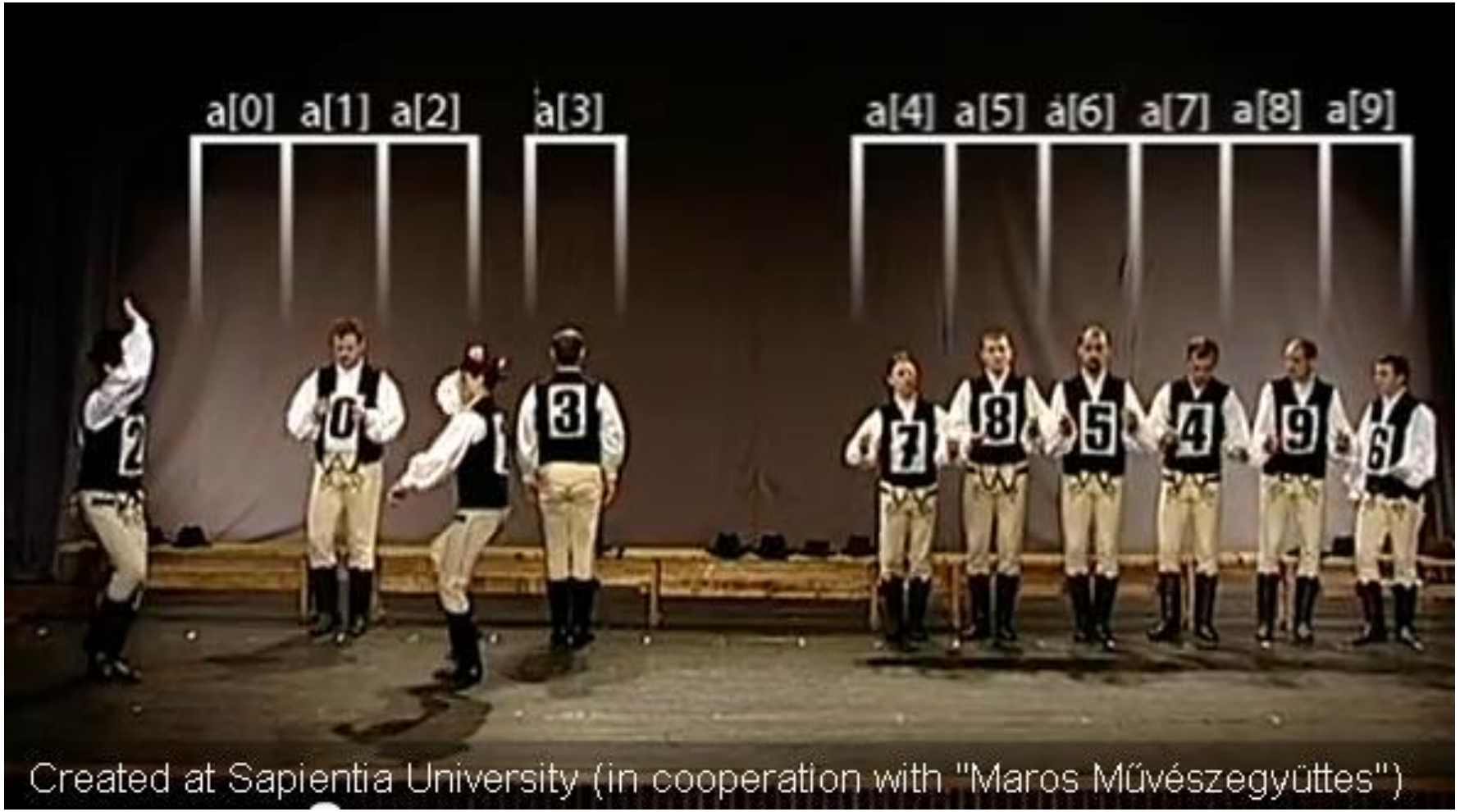
Пример работы

6 5 3 1 8 7 2 4

Анализ алгоритма

- **$O(n^2)$**
- **$O(n \cdot lg(n))$**
- **$O(lg(n))$** дополнительной памяти
- Неустойчивый
- Неестественность поведения

Quick-sort with Hungarian (Küküllőmenti legényes) folk dance



Сортировка выбором

- При сортировке простым выбором массив условно делится на две части: упорядоченную и неупорядоченную.
- Первоначально упорядоченная часть пуста, а неупорядоченная часть (остаток) включает все элементы.
- В остатке ищется минимальный элемент (при сортировке по возрастанию).
- Найденный элемент меняется местами с первым элементом из остатка, и упорядоченная часть увеличивается на один элемент, а остаток уменьшается на один элемент.

- Сложность такого алгоритма **$O(n^2)$**

  (в среднем, немного медленнее простых вставок). В общем случае эта сортировка быстрее, чем обменная, т. к. здесь меньше обменов.

Алгоритм

- 1. Начало алгоритма.
  2. Цикл 1 i=0, N-2 выполнять пункты 3-7.
  3. min=K[i]; imin=i;
  4. Цикл j=i+1,N-1 выполнять пункты 5-6.
  5. Если K[j]<min, тогда выполнить пункт 6, иначе продолжить цикл 2.
  6. Установить min=K[j]; imin=j; продолжить цикл 2
  7. Установить K[imin]=K[i]; K[i]=min; продолжить цикл 1.
  8. Конец алгоритма.

Метод квадратичного выбора

- Простой выбор можно усовершенствовать.
- Э.Х.Фрэнд (в 1956 году) впервые был опубликовал метод квадратичного выбора, заключающийся в том, что весь массив разбивается на $\sqrt{n}$ подмассивов (сложность такого алгоритма $O(n\sqrt{n})$. Далее определяется минимальный элемент в каждом подмассиве.
- На рисунке подчеркнуты элементы, которые выбираются на каждом шаге алгоритма. Нужен дополнительный массив для записи результата.

Пример

5	10	2	8	6	9	4	3	7
	2			6			3	
	5			6			3	
	5			6			4	
	5			6			7	
	10			6			7	
	10			8			7	
	10			8			все	
	10			9				
	10			все				

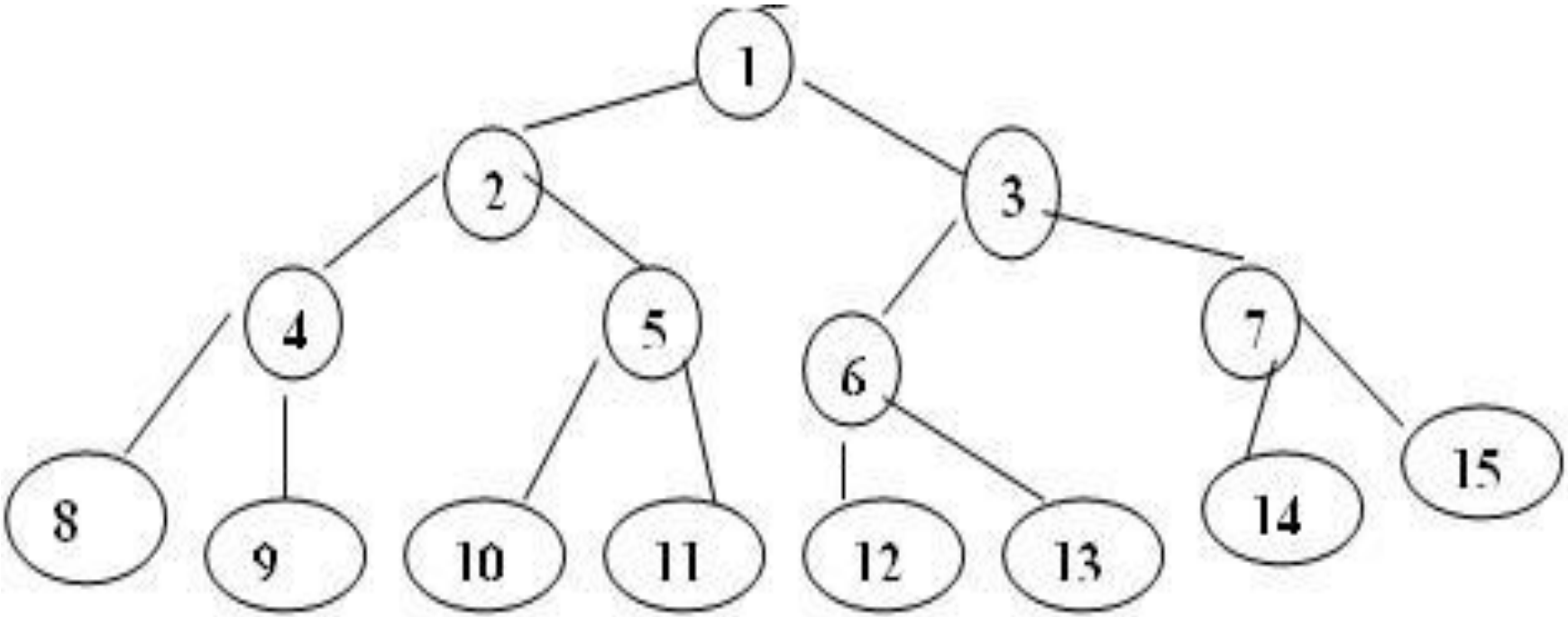
Дальнейшее совершенствование

- Далее тот же автор предложил идею кубического выбора, т. е. разделить массив на $\sqrt[3]{N}$ подмассивов.
- Далее используя $\sqrt[4]{N}$, $\sqrt[5]{N}$ и т. д. подмассивов, мы придём к тому, что Фрэнд назвал выбором n-й степени, основанной на структуре бинарного дерева и тогда сложность такого алгоритма **O(n·lg(n))** и он называется алгоритмом выбора из дерева.

Пирамидальная сортировка **Heap Sort**

- Дж. Уильямс в 1964г., используя идею выбора из дерева, опубликовал алгоритм пирамидальной сортировки.
- Массив ключей $K_1, K_2, \ldots, K_N$ называется «пирамидой», если K[j/2] >=Kj, при $1 \le [j/2] < j \le N$.
- В пирамиде потомки всегда меньше своих родителей.
- В этом случае $K_1 \ge K_2$, $K_1 \ge K_3$, $K_2 \ge K_4$, $K_2 \ge K_5$, $K_3 \ge K_6$, $K_3 \ge K_7$, … .

- Потомки К –го элемента вычисляется как (2*К) и (2*К+1).
- Необходимо, сначала преобразовать исходный массив К в пирамиду, а затем использовать бинарное дерево, перемещая на каждом шаге в его вершину максимальный элемент.
- Бинарное дерево (адреса элементов в одномерном массиве) будет иметь вид


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Алгоритм пирамидальной сортировки (heapsort)

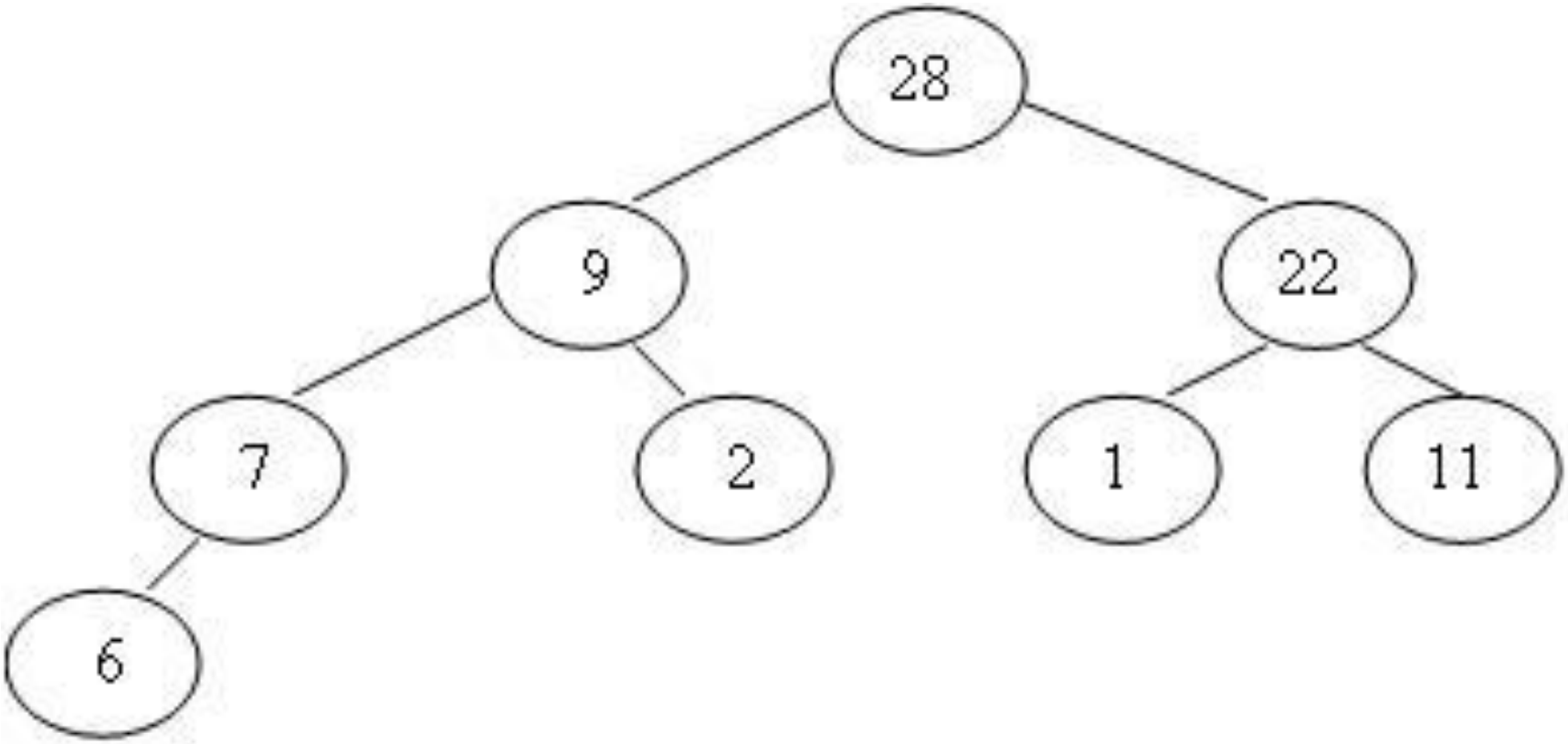
- 1. Начало алгоритма.
  2. r=N-1, L=N/2.
  3. Цикл 1 пока L>0 выполнять пункты 4-5.
  4. L=L-1; kl=K[L].
  5. Вызов процедуры Form.
  6. Цикл пока r>1 выполнять пункты 7-8.
  7. kl=K[r]; K[r]=K[1]; r=r-1.
  8. Вызов процедуры Form.
  9. K[0]=kl.
  10. Конец алгоритма.

Алгоритм процедуры Form

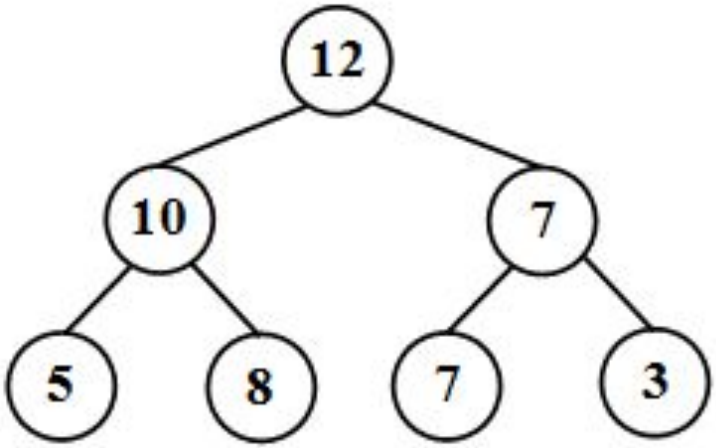
1. Начало алгоритма процедуры
2. i=L; j=2*i+1;
3. Цикл 1: пока j<=r выполнять пункты 4-7.
4. Если j<r && k[j]<k[j+1], тогда j=j+1;
5. Если kl>= K[j], тогда выйти цикла.
6. K[i]=K[j]; i=j; j=2*j;
7. Продолжить цикл 1.
8. K[i]=kl.
9. Конец алгоритма процедуры

Таблица 5

Шаг	Значения элементов массива K								L	kl
1	1	2	28	7	9	22	11	6	4	7
2	1	2	28	7	9	22	11	6	3	28
3	1	2	28	7	9	22	11	6	2	2
4	1	9	28	7	2	22	11	6	1	1
5	28	9	22	7	2	1	11	6	0	
Пирамида построена (рисунок 9)									r	Kl
1	28	9	22	7	2	1	11	6	8	6
2	22	9	11	7	2	1	6	28	7	6
3	11	9	6	7	2	1	22	28	6	1
4	9	7	6	1	2	11	22	28	5	2
5	7	2	6	1	9	11	22	28	4	1
6	6	2	1	7	9	11	22	28	3	1
7	2	1	6	7	9	11	22	28	2	1
8	1	2	6	7	9	11	22	28	1	1

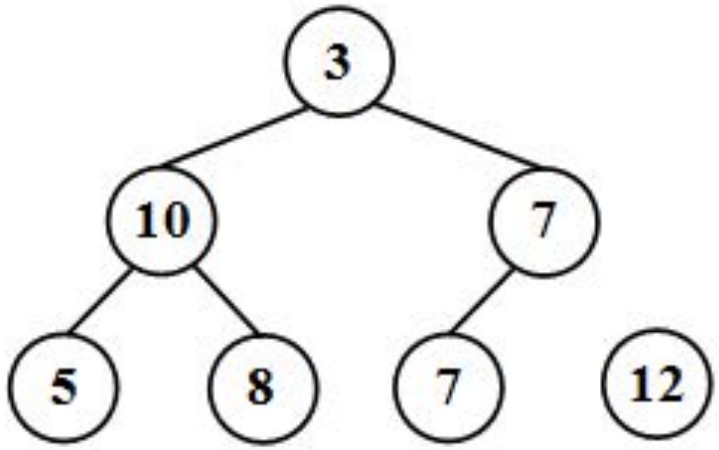

28
9
22
7
2
1
11
6

A



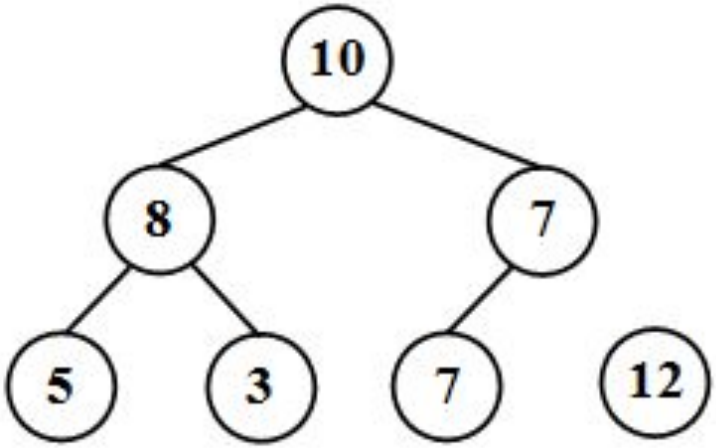
12 10 7 5 8 7 3

B



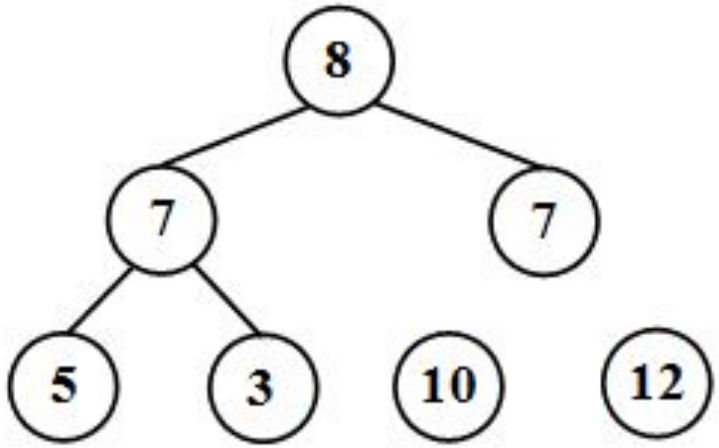
3 10 7 5 8 7 12

C



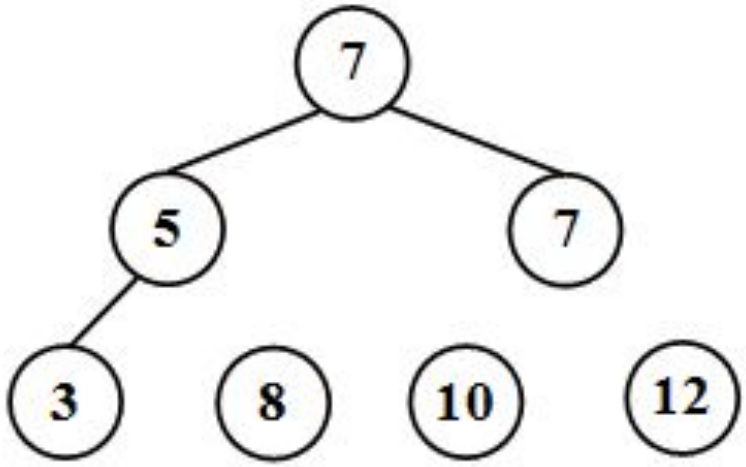
10 8 7 5 3 7 12

D



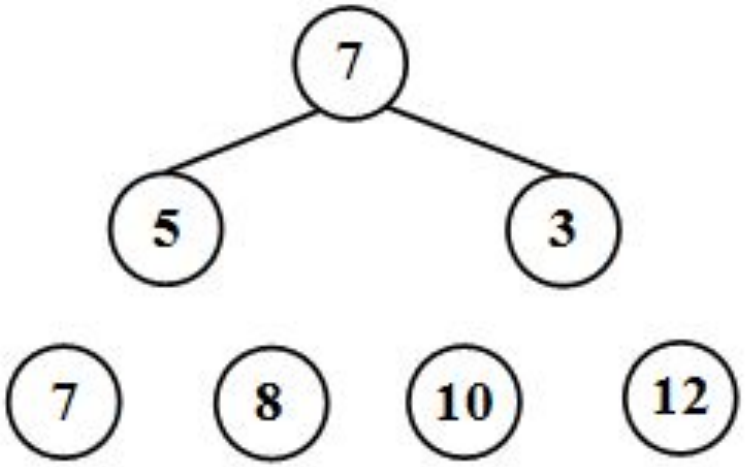
8 7 7 5 3 10 12

E



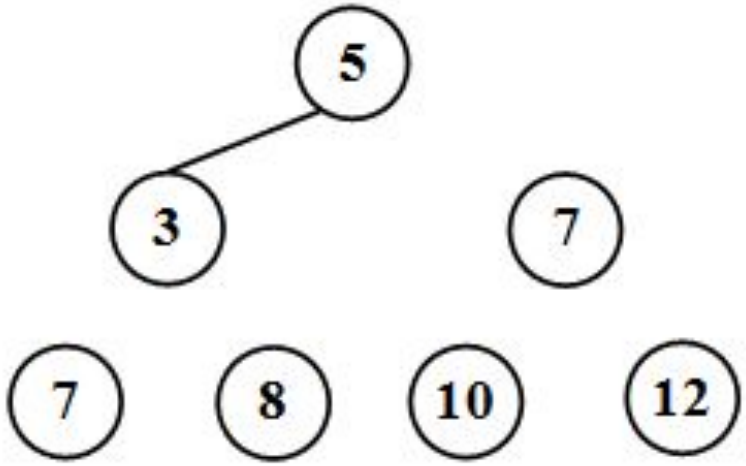
7 5 7 3 8 10 12

F



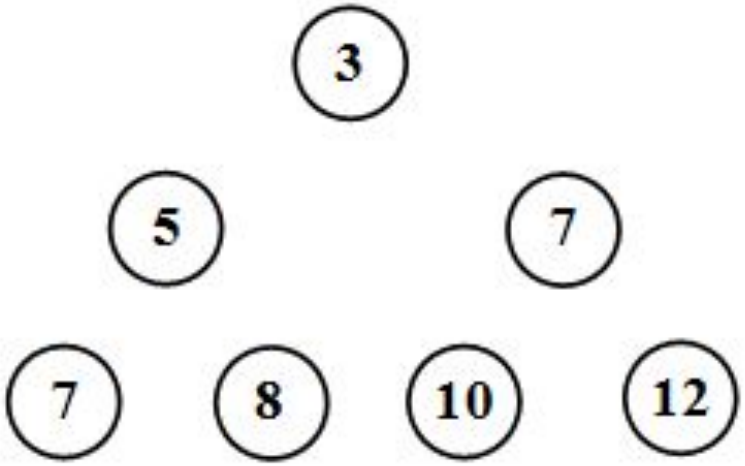
7 5 3 7 8 10 12

G



5 3 7 7 8 10 12

H



3 5 7 7 8 10 12

Анализ алгоритма

- **O(n·lg(n))**
- Не использует дополнительной памяти
- Неустойчивый
- Неестественность поведения

(Данная сортировка на почти отсортированных массивах работает также долго, выигрыш ее получается только на больших n.)

Сортировка слиянием (Merge sort)

- Эффективный алгоритм сортировки предложенный легендарным Джоном фон Нейманом в 1945 году.
- Сортировка была придумана во время работы над «Манхеттенским проектом» как средство обработки больших массивов статистических данных.

Работа алгоритма

2	1	6	0	5	3	7	4

Алгоритм

- Разделение: массив разбивается на два подмассива.

- Упорядочивание: подмассивы сортируются (к ним рекурсивно применяется сортировка слиянием).
- Слияние: упорядоченные подмассивы объединяются в один отсортированный массив.

Анализ алгоритма

- **O**(**n** log **n**) в лучшем, среднем, худшем случае
- используует дополнительную память **O**(**n**)
- Устойчивый
- Естественность поведения O(*n*) на упорядоченном массиве