Сортировка

Алтайский государственный университет
Факультет математики и ИТ
Кафедра информатики

Барнаул 2016

План

Сортировка: общие замечания

- Задача сортировки
- Внутренняя и внешняя сортировка
- Устойчивость
- Сортировка массива

Сортировка

- **Сортировка** – процесс перестановки объектов заданной совокупности в определенном порядке (возрастающем или убывающем)
- **Целью** сортировки обычно является облегчение последующего поиска элементов в отсортированном множестве
- В зависимости от объема и структуры данных методы сортировки подразделяются на:
  - Внутренние – сортировка массивов
  - Внешние – сортировка файлов

Сортировка: более формально

- **Дано:** N объектов $a_1, a_2, \ldots, a_N$

- **Требуется:** упорядочить заданные объекты, т. е. переставить их в такой последовательности $a_{p1}, a_{p2}, \ldots, a_{pN}$, чтобы их ключи расположились в неубывающем порядке $k_{p1} \le k_{p2} \le \ldots \le k_{pN}$

  - Ключ $k_i = f(a_i)$ – некоторая функция элемента
    - ai – целое число => $k_i = a_i$
    - ai – структура => $k_i = a_i.key$

Сортировка: устойчивость

- При устойчивой сортировке относительный порядок элементов с одинаковыми ключами не меняется
- Если $k_{pi} \leq k_{pj}$ и $i < j$, то $p_i < p_j$
- Устойчивость желательна, если элементы уже упорядочены

Сортировка массивов

- Массив – одна из наиболее распространенных совокупностей, подвергаемых сортировке

- От алгоритмов сортировки массива требуется
  - экономичность по памяти
    - Перестановки, упорядочивающие массив, должны выполняться на том же месте
  - экономичность по времени
    - Мера эффективности *C* – количество сравнений ключей и *M* – число перестановок элементов

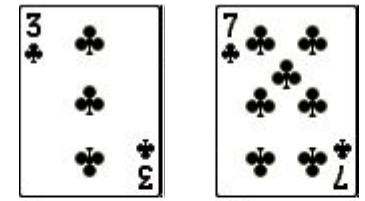
Сортировка массивов: алгоритмы

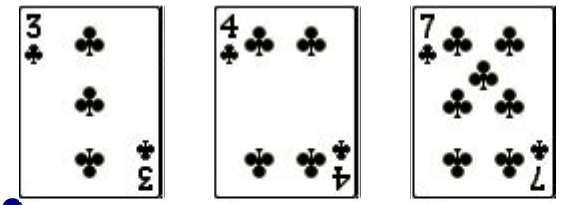
- Простые методы сортировки – прямые, временная сложность – $O(n^2)$
  - сортировка прямыми вставками (by insertion)
  - сортировка прямым выбором (by selection)
  - сортировка прямыми обменами выбором (by exchange)
- «Улучшенные» методы сортировки, временная сложность – $O(n \log n)$
  - быстрая сортировка Хоара (quicksort)
  - сортировка слияниями (mergesort)
  - сортировка Шелла (shellsort)
  - …

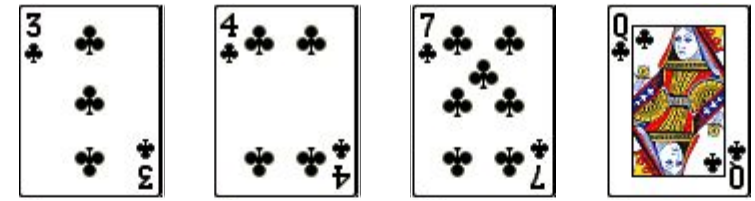
Сортировка прямыми выставками

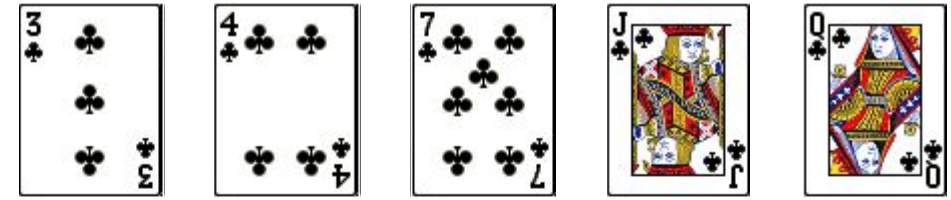
- Идея
- Псевдокод
- Анализ алгоритма
- Сортировка бинарными вставками

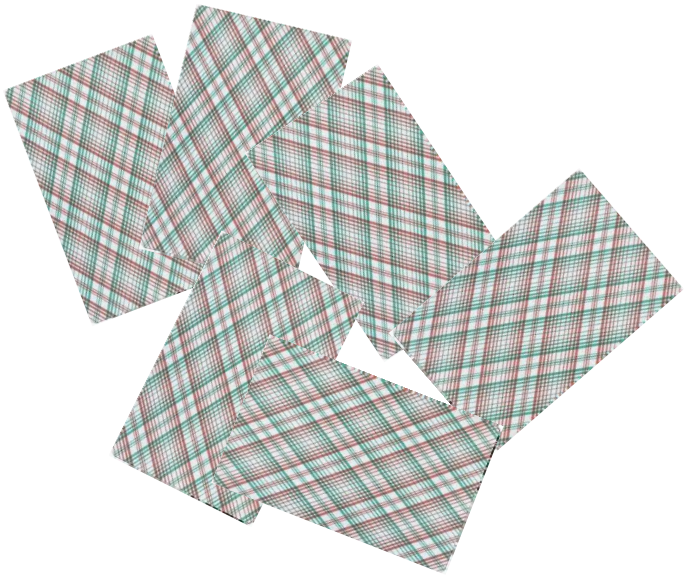
Сортировка вставками





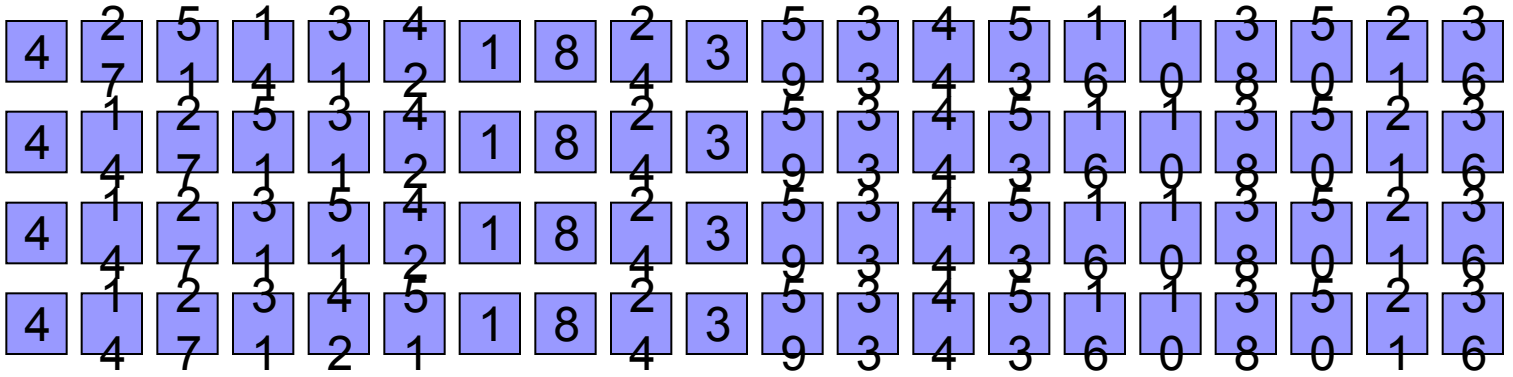






Сортировка простыми вставками

- Массив делится на две части
  - «готовую» $a_1, a_2, \ldots, a_{i-1}$
  - исходную $a_i, a_{i+1}, \ldots, a_N$
- Для каждого i от 2 до N
  - из исходной части извлекается i-й элемент
  - вставляется в готовую часть на нужное место



Сортировка простыми вставками

```
INSERTION_SORT(A)
 1  for j ← 2 to length[A]
 2       do key ← A[j]
 3          ▷ Вставка элемента A[j] в отсортированную
            ▷ последовательность A[1..j − 1]
 4          i ← j − 1
 5          while i > 0 и A[i] > key
 6               do A[i + 1] ← A[i]
 7                  i ← i − 1
 8          A[i + 1] ← key
```

Сортировка простыми вставками

- Анализ алгоритма
  - Лучший случай: массив упорядочен
  - Худший случай: массив упорядочен в обратном порядке

  - $C_{min} = N - 1$ $\quad M_{min} = 3(N-1)$
  - $C_{avg} = (N^2 + N - 2)/4$ $\quad M_{avg} = (N^2 + 9N - 10)/4$
  - $C_{max} = (N^2 + N - 4)/2$ $\quad M_{max} = (N^2 + 3N - 4)/2$

- Итог: $\mathbf{T}(N) = C(N) + M(N) = O(N^2)$

Сортировка бинарными вставками

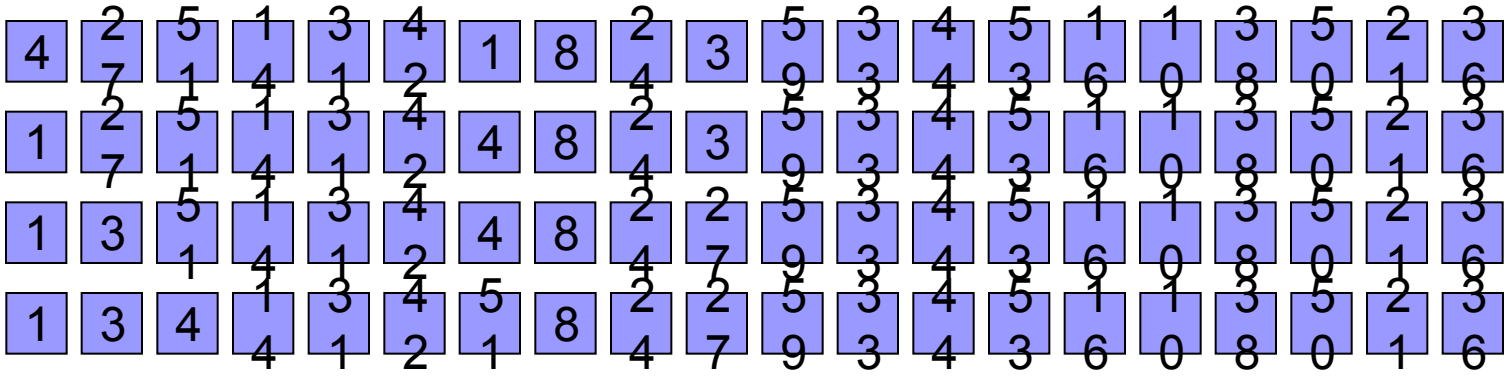
- Сортировка простыми вставками может быть улучшена
  - Можно ускорить поиск подходящего места в «готовой» части, т.к. она упорядочена
  - В упорядоченной последовательности применим бинарный поиск!
    - Сложность бинарного поиска в худшем случае есть $O(\log N)$
    - Количество сравнений есть $O(N \log N)$
  - Но по-прежнему, $M(N) = O(N^2)$
- Итог: $\mathbf{T}(N) = O(N \log N) + O(N^2) = O(N^2)$

Сортировка прямым выбором

- Идея
- Псевдокод
- Анализ алгоритма

Сортировка простым выбором

- Массив делится на две части
  - «готовую» $a_1, a_2, \ldots, a_{i-1}$
  - исходную $a_i, a_{i+1}, \ldots, a_N$
- Для каждого i от 1 до N–1
  - присвоить k индекс минимального элемента в исходной части
  - поменять местами элементы a_i и a_k



Сортировка простым выбором

```
SELECTIONSORT(A)
1  for i ← 1 to length[A] – 1 do
2    k ← i
3    x ← A[i]
4    for j ← 1 to length[A] – 1 do
5         if A[j] < x then
6              k ← j
7              x ← A[j]
8         A[k] ← A[i]
9         A[i] ← x
```

Сортировка простым выбором

- Анализ алгоритма
  - Количество сравнений не зависит от начального порядка элементов:
  - Лучший случай: массив упорядочен
  - Худший случай: массив упорядочен в обратном порядке

  - $C = (N^2 - N)/2$ $\qquad M_{min} = 3(N - 1)$
  - $M_{avg} \approx N(\ln N + \gamma),\ \gamma = 0.577216\ldots$
  - $M_{max} = \lfloor N^2/4 \rfloor + 3(N - 1)$

- Итог (худший случай) : $\mathbf{T}(N) = C(N) + M(N) = O(N^2)$

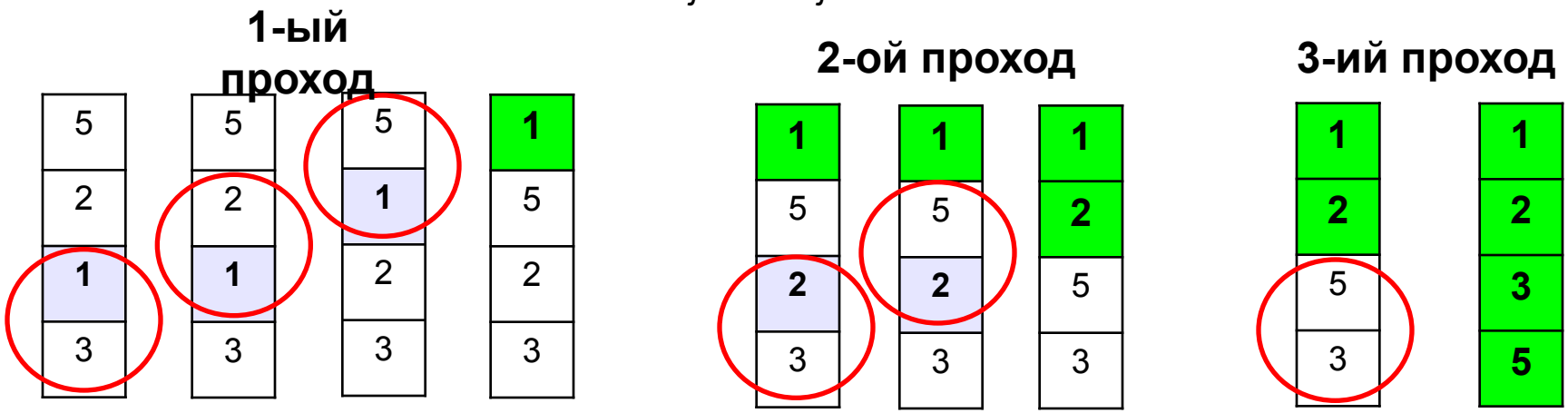
- В среднем сортировка выбором выгоднее сортировки вставками

Сортировка прямыми обменами

- Идея
- Псевдокод
- Анализ алгоритма

Сортировка простыми обменами (пузырьковая сортировка)

- **Идея**: пузырек воздуха в стакане воды поднимается со дна вверх – самый маленький («легкий») элемент массива перемещается вверх («всплывает»)
- Для каждого *i* от 2 до *N*
- Для каждого *j* от *N* до *i*
  - Если в паре элементов a_{j-1} и a_j нарушен порядок,
  - то поменять местами a_{j-1} и a_j



Сортировка простыми обменами (пузырьковая сортировка)

```
BUBBLESORT(A)
1  for i ← 1 to length[A]
2        do for j ← length[A] downto i + 1
3              do if A[j] < A[j − 1]
4                    then Поменять местами A[j] ↔ A[j − 1]
```

Си-программа

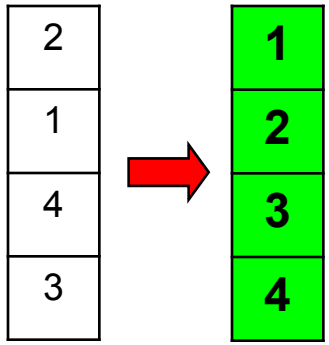
```
void main() {
  const int N=10;
  int A[N], i, j, c;
  // заполнить массив
  // вывести исходный массив
  for (i=0; i<N-1; i++){
    for (j=N-2; j>=i; j--)
      if (A[j] > A[j+1]) {
        c=A[j];
        A[j]=A[j+1];
        A[j+1]=c;
      }
  }
  // вывести полученный массив
}
```

элементы выше A[i] уже поставлены

меняем A[j] и A[j+1]

Улучшенный метод «пузырька»

- Если при выполнении очередного прохода не было обменов, то массив уже отсортирован и остальные проходы не нужны
- Реализуется через **переменную-флаг**, показывающую, были ли обмены
  - Если флаг поднят, то обмены были и нужен еще один проход
  - Если флаг опущен, то – выход



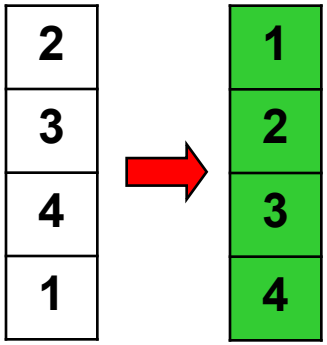
Улучшенный метод «пузырька»

```
i=0;
do {
  flag=0; // сбросить флаг
  for(j=N-2; j>= i ; j--)
    if (A[j] > A[j+1]) {
      c=A[j];
      A[j]=A[j+1];
      A[j+1]=c;
      flag=1;  // поднять флаг
    }
  i++;
}
while ( flag );  // выход при flag = 0
```

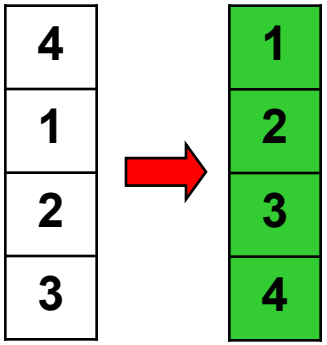
Шейкерная сортировка

- Метод пузырька несимметричен
  - При нарушении почти полного порядка «легкими» элементами, требуется мало проходов
  - При нарушении почти полного порядка «тяжелыми» элементами, требуется много проходов

1 проход



3 прохода



- Выход: чередовать направления проходов

Шейкерная сортировка

```
procedure shakersort;
    var j,k,l,r: index;   x: item;
begin l := 2; r := n; k := n;
    repeat
        for j := r downto l do
            if a[j-1] .key > a[j] .key then
            begin x := a[j-1]; a[j-1] := a[j]; a[j] := x;
                k := j
            end ;
        l := k+1;
        for j := l to r do
            if a[j-1] .key > a[j] .key then
            begin x := a[j-1]; a[j-1] := a[j]; a[j] := x;
                k := j
            end ;
        r := k-1;
    until l > r
end {shakersort}
```

Сортировка простыми обменами

- Анализ алгоритма
  - Лучший случай: массив упорядочен
  - Худший случай: массив упорядочен в обратном порядке

  - $C_{min} = (N^2 - N)/2$ $\quad M_{min} = 0$
  - $C_{avg} = (N^2 - N)/2$ $\quad M_{avg} = (N^2 - N)/4$
  - $C_{max} = (N^2 - N)/2$ $\quad M_{max} = (N^2 - N)/2$

- Итог: $\mathbf{T}(N) = C(N) + M(N) = O(N^2)$

«Шейкерная» сортировка

- Анализ алгоритма
  - Лучший случай: массив упорядочен
  - Худший случай: массив упорядочен в обратном порядке

  - $C_{min} = N - 1$ $M_{min} = 0$
  - $C_{avg} = (N^2 - N(k+\ln N))/2$ $M_{avg} = (N^2 - N)/4$
  - $C_{max} = (N^2 - N)/2$ $M_{max} = (N^2 - N)/2$

- Итог: $\mathbf{T}(N) = C(N) + M(N) = O(N^2)$

Прямые методы сортировки

- Сортировка обменами несколько менее эффективна сортировок вставками и выбором
- Шейкерная сортировка выгодна, когда массив почти упорядочен
- Общее свойство: перемещение элементов ровно на одну позицию за один прием
  - Можно показать, что среднее расстояние, на которое должен сдвигаться элемент равно $N/3$
- Надо стремиться к дальним пересылкам элементов

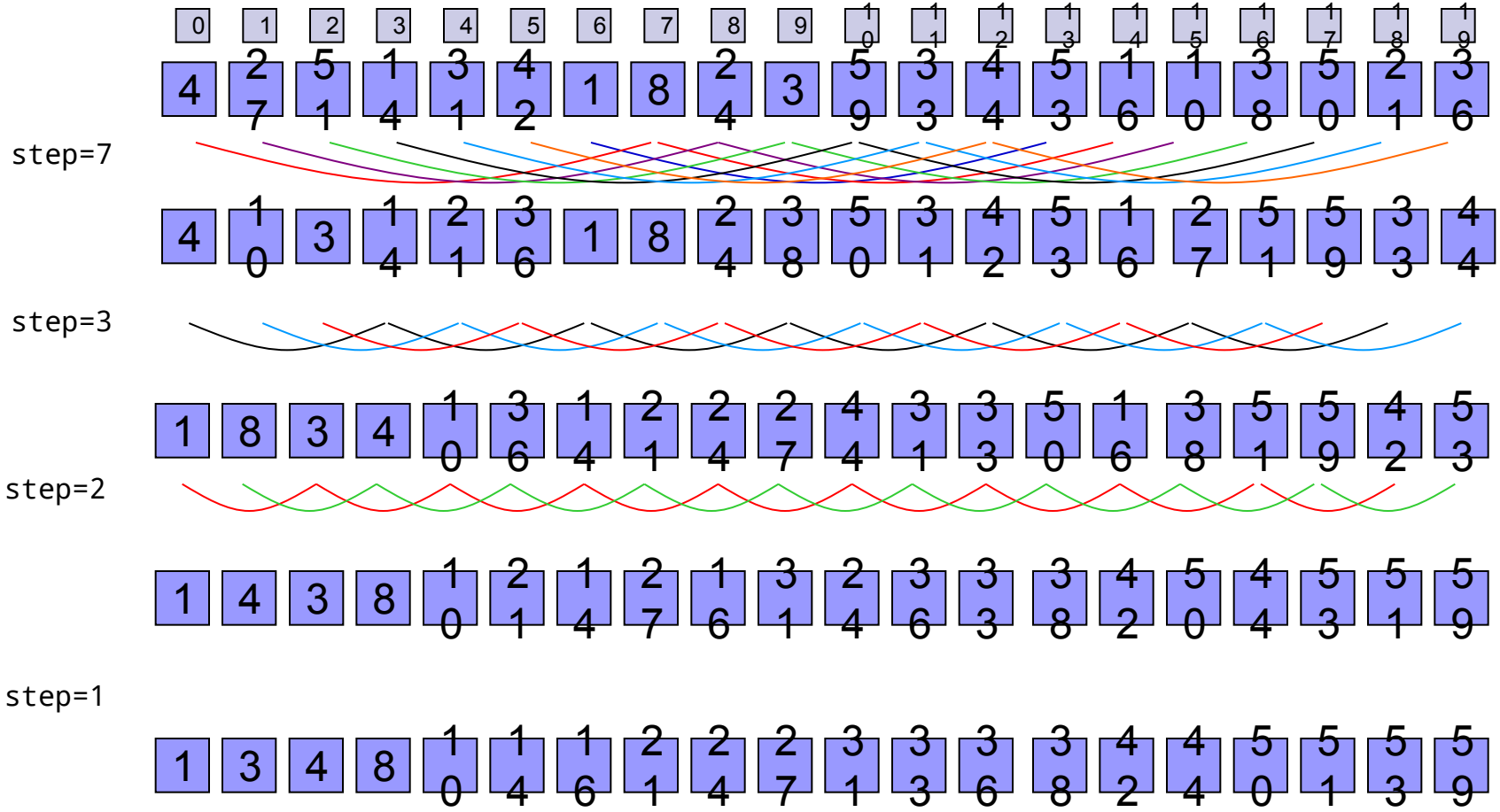
Сортировка Шелла

- Идея алгоритма
- Анализ алгоритма

Сортировка Шелла
(Д.Л.Шелл, 1959)

- Элементы разбиваются на подмножества для некоторого $h>1$
  - $a_1, a_{1+h}, a_{1+2h}, a_{1+3h}, \ldots$
  - $a_2, a_{2+h}, a_{2+2h}, a_{2+3h}, \ldots$
  - …
  - $a_t, a_{t+h}, a_{t+2h}, a_{t+3h}, \ldots$
- Сортировка проводится методом вставок для каждого подмножества
- h уменьшается и процедура повторяется, пока $h>0$

Сортировка Шелла



Сортировка Шелла

- Анализ алгоритма
  - Анализ приводит к сложным математическим задачам
  - Для эффективной сортировки соседние значения не должны быть кратными
    - Иначе массив распадается на непересекающиеся цепочки
    - Требуется, чтобы цепочки взаимодействовали как можно чаще
  - Д. Кнут предлагает выбирать h так (порядок обратный):
    - 1, 4, 13, 40, 121, …, т.е. $h_{k-1} = 3h_k+1$, $h_t=1$, $t = [\log_3 N] - 1$
    - 1, 3, 7, 15, 31, …, т.е. $h_{k-1} = 2h_k+1$, $h_t=1$, $t = [\log_2 N] - 1$

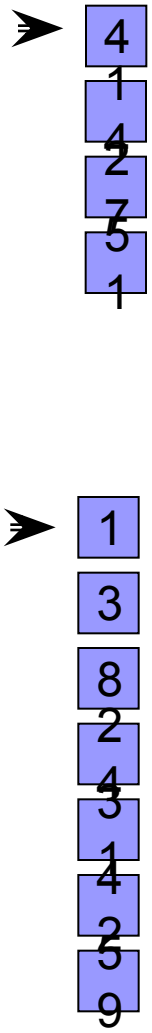
Количество перестановок элементов M
(по результатам экспериментов со случайными массивами)

	$N = 25$	$N = 1000$	$N = 100000$
Сортировка Шелла	50	7700	2 100 000
Сортировка простыми вставками	150	240 000	2.5 млрд.

Сортировка слиянием

- Идея алгоритма
- Временная сложность алгоритма

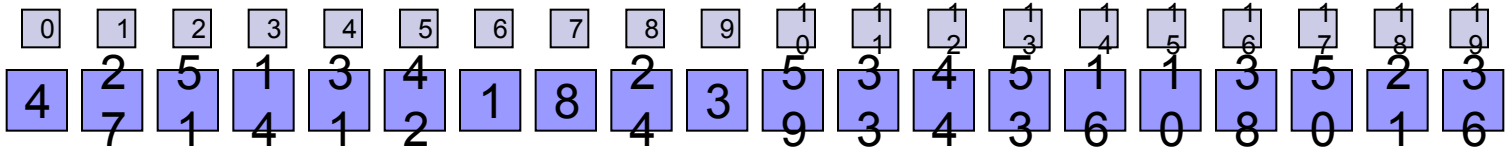
Слияние упорядоченных массивов



- Java-код

```
int[] merge(int[] a, int[] b) {
    int na = a.length,
        nb = b.length,
        nc;
    int[] c = new int[nc = na + nb];
    int ia = 0,
        ib = 0,
        ic = 0;
    while (ia < na && ib < nb) {
        if (a[ia] < b[ib])
            c[ic++] = a[ia++];
        else
            c[ic++] = b[ib++];
    }
    while (ia < na) c[ic++] = a[ia++];
    while (ib < nb) c[ic++] = b[ib++];
    return c;
}
```

Сортировка слиянием (фон Неймана)



И так далее…

Алгоритм сортировки слиянием (фон Неймана)

- Псевдокод

```
// Merge() сливает два упорядоченных подмассива
// в единый подмассив

MergeSort(A, left, right) {
   if (left < right) {
    mid = floor((left + right) / 2); // середина
    MergeSort(A, left, mid);
    MergeSort(A, mid+1, right);
    Merge(A, left, mid, right);
   }
}
```

Алгоритм сортировки слиянием (фон Неймана)

```
#include <stdio.h>
#include <stdlib.h>

void merge (int *a, int n, int m) {
    int i, j, k;
    int *x = malloc(n * sizeof (int));
    for (i = 0, j = m, k = 0; k < n; k++)
    {
   x[k] = j == n ? a[i++]: i == m ? a[j++]: a[j] < a[i] ?a[j++]: a[i++];
    }
   for (i = 0; i < n; i++) {
       a[i] = x[i];
   }
   free(x);
}
```

```
void merge_sort (int *a, int n) {
if (n < 2)
return;
int m = n / 2;
merge_sort(a, m);
merge_sort(a + m, n - m);
merge(a, n, m);
}

int main () {
int a[] = {4, 65, 2, -31, 0, 99, 2, 83, 782, 1};
int n = sizeof a / sizeof a[0];
int i;
for (i = 0; i < n; i++)
printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
merge_sort(a, n);
for (i = 0; i < n; i++)
printf("%d%s", a[i], i == n - 1 ? "\n" : " ");
return 0;
}
```

Output:

```
4 65 2 -31 0 99 2 83 782 1
-31 0 1 2 2 4 65 83 99 782
```

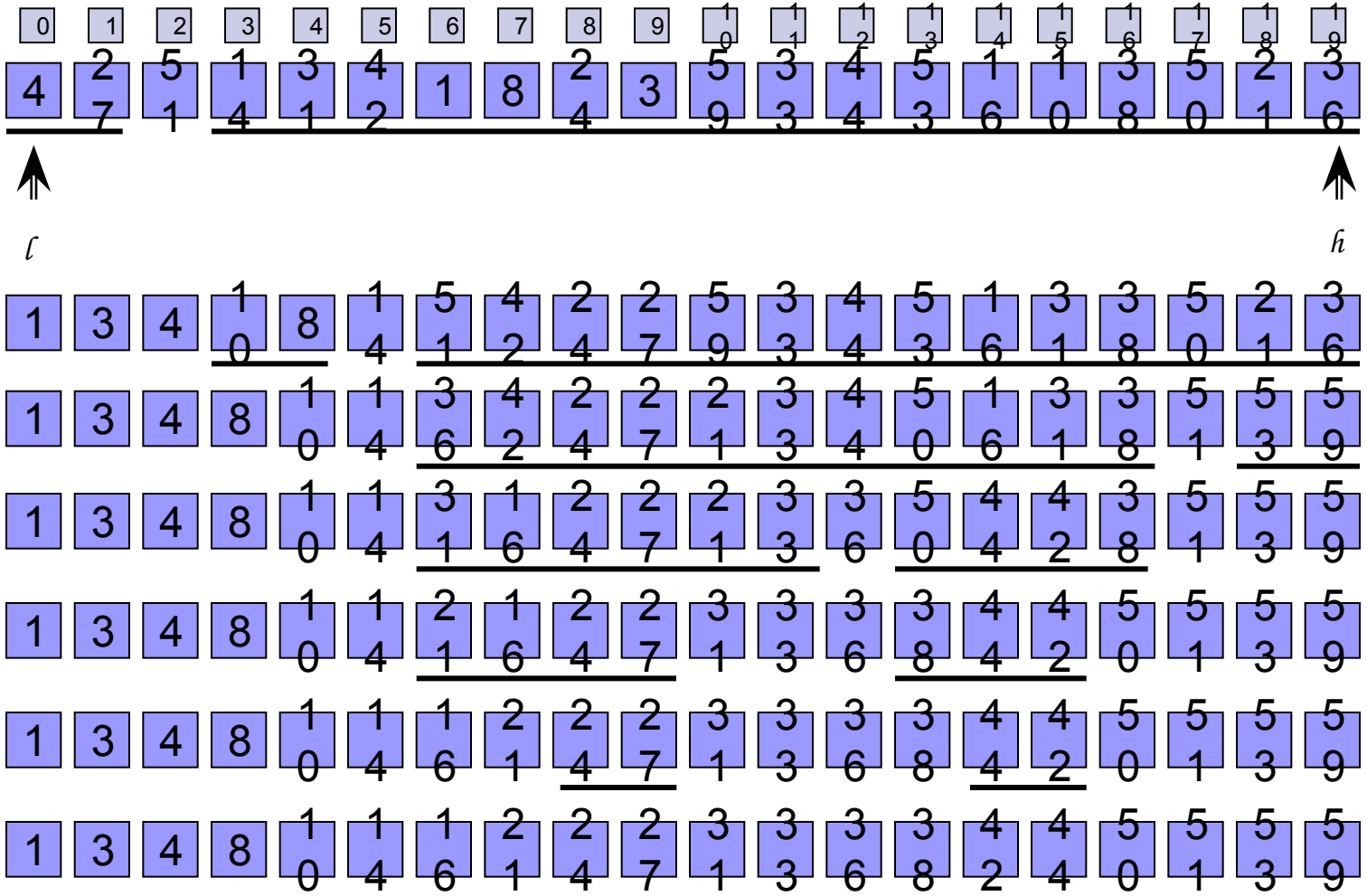
Сортировка слиянием

- Анализ алгоритма
  - Анализ приводит к сложным математическим задачам
  - Асимптотическая сложность – O($N \log N$)

Быстрая сортировка

- Идея алгоритма
- Временная сложность алгоритма

Быстрая сортировка (Хоара)



Быстрая сортировка

- Псевдокод

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4      do if A[j] ⩽ x
5             then i ← i + 1
6                  Обменять A[i] ↔ A[j]
7  Обменять A[i + 1] ↔ A[r]
8  return i + 1
```

```
QUICKSORT(A, p, r)
1  if p < r
2     then q ← PARTITION(A, p, r)
3          QUICKSORT(A, p, q − 1)
4          QUICKSORT(A, q + 1, r)
```

Чтобы выполнить сортировку всего массива A, вызов процедуры должен иметь вид QUICKSORT$(A, 1, lehgth\,[A])$.

Пример программы

```
int a[100];
void quickSort(int l, int r)
{
   int x = a[l + (r - l) / 2];
   //запись эквивалентна (l+r)/2,
   //но не вызвает переполнения на больших
данных
   int i = l;
   int j = r;
   //код в while обычно выносят в процедуру particle
   while(i <= j)
   {
      while(a[i] < x) i++;
      while(a[j] > x) j--;
      if(i <= j)
      {
         swap(a[i], a[j]);
         i++;
         j--;
      }
   }
   if (i<r)
            quickSort(i, r);

   if (l<j)
      quickSort(l, j);
}
```

```
int main()
{
   int n;//количество элементов в массиве
   scanf("%d",n]);

   for(int i = 0; i < n; i++)
   {
      scanf("%d",a[i]);
   }
   quickSort(0, n-1);
   for(int i = 0; i < n; i++)
   {
      printf("%d ",a[i]);
   }
   return 0;
}
```

Улучшения алгоритма

- Первый элемент в сортируемом куске выбирается случайно и запоминается
- Участки, меньшие определенного размера, сортируются простыми способами
- Иногда исключение рекурсивных вызовов приводит к повышению эффективности

Быстрая сортировка

- Анализ алгоритма
  - Эффективность во многом зависит от сбалансированности разбиения на подмассивы
    - Наихудшее разбиение: 1 к (N–1) => $O(N^2)$
    - Лучшее разбиение: $N/2$ к $N/2$ => $O(N \log N)$
    - Средний случай: $O(N \log N)$

Вопросы?

- Сортировка: общие замечания
  - Задача сортировки
  - Внутренняя и внешняя сортировка
  - Устойчивость
  - Сортировка массива
- Сортировка прямыми вставками
  - Идея
  - Псевдокод
  - Анализ алгоритма
  - Сортировка бинарными вставками
- Сортировка прямым выбором
  - Идея алгоритма
  - Временная сложность алгоритма
- Сортировка прямыми обменами
  - Идея алгоритма
  - Временная сложность алгоритма
  - Улучшения алгоритма
  - Шейкерная сортировка
- Сортировка Шелла
  - Идея алгоритма
  - Временная сложность алгоритма
- Сортировка слияниями
  - Идея алгоритма
  - Временная сложность алгоритма
- Быстрая сортировка
  - Идея алгоритма
  - Временная сложность алгоритма