ITIVITI

LET IT TRADE

Алгоритмы и структуры данных

Косяков Михаил Сергеевич
к.т.н., доцент кафедры ВТ

... и многие другие

УНИВЕРСИТЕТ ИТМО

Умножение чисел

- Требуется перемножить два n -значных числа
- Разобьем каждое число на две $p = n/2$ –значные половины:
  - $num_1 = x_1 * 10^p + x_0$
  - $num_2 = y_1 * 10^p + y_0$
- $num_1 * num_2 =$

$$x_1 * y_1 * 10^{2p} + (x_1 * y_0 + x_0 * y_1) * 10^p + x_0 * y_0$$

- $T(N) = 4T(N/2) + O(N)$
- Оценка сложности: $\Theta(N^2)$

Умножение чисел

- Вместо четырех подзадач требуется решить три
- $C_2 = x_1 * y_1$
- $C_0 = x_0 * y_0$
- $C_1 = (x_1 * y_0 + x_0 * y_1) = (x_1 + x_0)(y_0 + y_1) - x_1 * y_1 - x_0 * y_0$
- $num_1 * num_2 = C_2 * 10^{2p} + C_1 * 10^p + C_0$

- $T(N) = 3T(N/2) + O(N)$
- Оценка сложности: $\Theta(N^{\log 3})$

Умножение полиномов

- $A(x) = \sum_{k=0}^{n} a_k x^k$ $\quad\quad$ $B(x) = \sum_{k=0}^{n} b_k x^k$
- $A(x) = A_1 x^{\frac{n}{2}} + A_0$ $\quad\quad$ $B(x) = B_1 x^{\frac{n}{2}} + B_0$

- Например $1 + 3x + x^2 + 7x^3 = (1 + 3x) + x^2(1 + 7x)$

- $A(x)B(x) = C_2 x^n + C_1 x^{\frac{n}{2}} + C_0$
- $C_2 = A_1 B_1$
- $C_1 = (A_0 + A_1)(B_0 + B_1) - A_0 B_0 - A_1 B_1$
- $C_0 = A_0 B_0$

Содержание курса

- Введение в теорию алгоритмов
- **Алгоритмы сортировок**
- Структуры данных
  - Линейные структуры
  - Бинарные деревья поиска
  - Хеши и хеш-функции
- Алгоритмы на графах
  - Обходы графов в ширину и глубину
  - Минимальные остовные деревья
  - Поиск кратчайших путей в графе

Мотивация

ITIVITI

LET IT TRADE

- Одна из самых естественных и распространенных задач:
  - Словарь – слова расположены в алфавитном порядке
  - Рейтинг студентов – у кого больше баллов, тот выше в списке
  - Расписание поездов, самолетов – по времени прибытия / отправления
  - Многие алгоритмы требуют отсортированные входные данные: различные геометрические задачи, быстрое объединение структур данных (в т.ч. natural join в СУБД) и т.д.

Бинарный поиск

ITIVITI

- Найдем идентификаторы всех таймзон:

```
TimeZoneIdentifier using_linear_str(const char * timezone_name)
{
    if (::strcmp(timezone_name, "America/Porto_Acre") == 0)
        { return TZ_STD_AMERICA_PORTO_ACRE; }
    if (::strcmp(timezone_name, "Australia/Perth") == 0)
        { return TZ_STD_AUSTRALIA_PERTH; }
    if (::strcmp(timezone_name, "Europe/Madrid") == 0)
        { return TZ_DST_EUROPE_MADRID; }
...
```

- Повторим эксперимент тысячу раз:

```
                          #       50%         95%         99.9%
=============================================================
linear_str_begin -
linear_str_end            1000    3831 µs     4062 µs     4530 µs
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Бинарный поиск

ITIVITI

- Найдем идентификаторы всех таймзон:

```
TimeZoneIdentifier using_linear(const char * timezone_name)
{
    const uint32_t hash_key = hash(timezone_name);

    if (0x013129B9 == hash_key) { return  TZ_STD_AMERICA_PORTO_ACRE; }
    if (0x014A1BF1 == hash_key) { return  TZ_STD_AUSTRALIA_PERTH; }
    if (0x03C9E0D7 == hash_key) { return  TZ_DST_EUROPE_MADRID; }
...
```

- Повторим эксперимент тысячу раз:

```
                          #       50%         95%        99.9%
=============================================================
linear_begin -
linear_end                1000    122 µs      140 µs     439 µs
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Бинарный поиск

ITIVITI

- Найдем идентификаторы всех таймзон:

```
TimeZoneIdentifier using_switch(const char * timezone_name)
{
    const uint32_t hash_key = hash(timezone_name);
    switch(hash_key)
    {
        case 0x013129B9: return  TZ_STD_AMERICA_PORTO_ACRE;
        case 0x014A1BF1: return  TZ_STD_AUSTRALIA_PERTH;
        case 0x03C9E0D7: return  TZ_DST_EUROPE_MADRID;
...
```

- Повторим эксперимент тысячу раз:

```
                        #       50%         95%         99.9%
=============================================================
switch_begin -
switch_end              1000    25 µs       28 µs       89 µs
```

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Бинарный поиск и сортировка

- Бинарный поиск:
  - $T(N) = O(1) + T(N/2)$
  - Число атомов во Вселенной ~10^{80}, найдем за ~260 шагов
  - ? $x_0 : f(x_0) = c$, $f(x)$ монотонно возрастает
- Сортировка:
  - Отношение порядка $<$ на множестве ключей:
  - Трихотомия: $\forall\, a, b : (a < b) \vee (b = a) \vee (b < a)$
  - Транзитивность: $(a < b) \wedge (b < c) \Rightarrow (a < c)$
  - Игра «камень, ножницы, бумага»
- Задача сортировки - получить перестановку, в которой ключи расположены в порядке неубывания

Характеристики сортировок

- Оценка эффективности алгоритма сортировки:
  - Время работы алгоритма в лучшем, среднем и худшем случае
  - Объем используемой дополнительной памяти
- Свойства алгоритма сортировки:
  - Рекурсивность
  - Стабильность
  - Адаптивность
  - Тип (на основе сравнения, подсчета, …)

Стабильность

По времени	->По фамилии (нестабильная)	->По фамилии (стабильная)
Петров 09.00	Беляев 10.00	Беляев 10.00
Петров 10.00	Иванов 12.00	Иванов 11.00
Беляев 10.00	Иванов 11.00	Иванов 12.00
Торопов 11.00	Косяков 13.00	Косяков 13.00
Иванов 11.00	Петров 10.00	Петров 09.00
Иванов 12.00	Петров 09.00	Петров 10.00
Косяков 13.00	Торопов 11.00	Торопов 11.00

Сортировка выбором

- Самый простой алгоритм сортировки:

```
1.  void selection_sort(int * a, int n)
2.  {
3.      for(int i = 0; i < n - 1; ++i) {
4.          for(int j = i + 1; j < n; ++j) {
5.              if (a[j] < a[i]) {
6.                  int t = a[i];
7.                  a[i] = a[j];
8.                  a[j] = t;
9.              }
10.         }
11.     }
12. }
```

- Можно через рекурсию: $T(N) = O(N) + T(N-1)$

Сортировка выбором

```
1.  void selection_sort(int * a, int n) {
2.      for(int i = 0; i < n - 1; ++i) {
3.          int min = i;
4.          for(int j = i + 1; j < n; ++j) {
5.              if (a[j] < a[min]) {
6.                  min = j;
7.              }
8.          }
9.          int t = a[i];
10.         a[i] = a[min];
11.         a[min] = t;
12.     }
13. }
```

- Оценка сложности: $\Theta(N^2)$
- Дополнительная память: $O(1)$

Сортировка пузырьком

```
1. void bubble_sort(int * a, int n) {
2.     bool swap = true;
3.     while(swap) {
4.         swap = false; --n;
5.         for(int i = 0; i < n; ++i) {
6.             if (a[i + 1] < a[i]) {
7.                 int t = a[i];
8.                 a[i] = a[i + 1];
9.                 a[i + 1] = t;
10.                swap = true;
11.            }
12.        }
13.    }
14. }
```

- Оценка сложности: $O(N^2)$
- Дополнительная память: $O(1)$

Сортировка слиянием

ITIVITI

LET IT TRADE

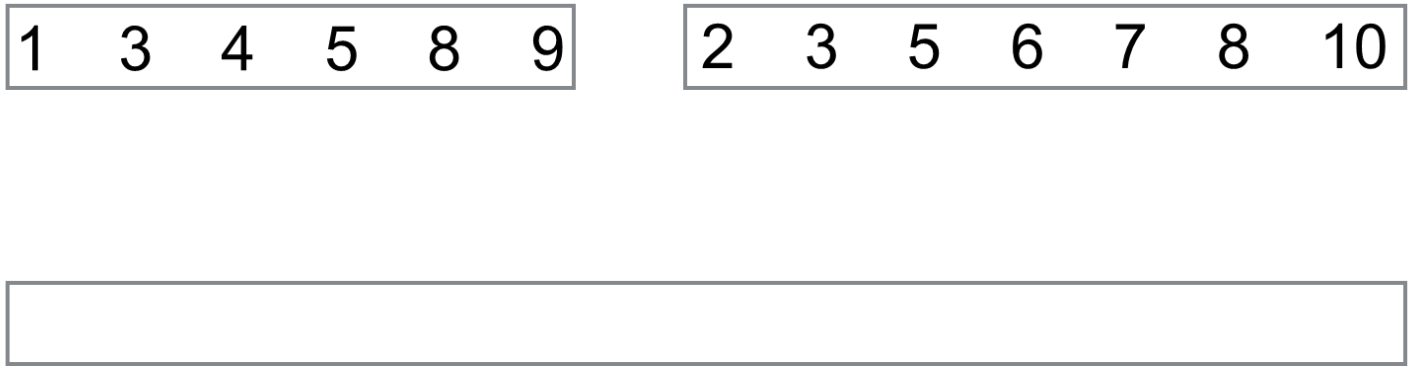
- Метод «разделяй и властвуй» (divide-and-conquer):
  - 1) разделим задачу на подзадачи
  - 2) решаем подзадачи с использованием рекурсии
  - 3) объединяем решения подзадач в решение задачи

Сортировка слиянием

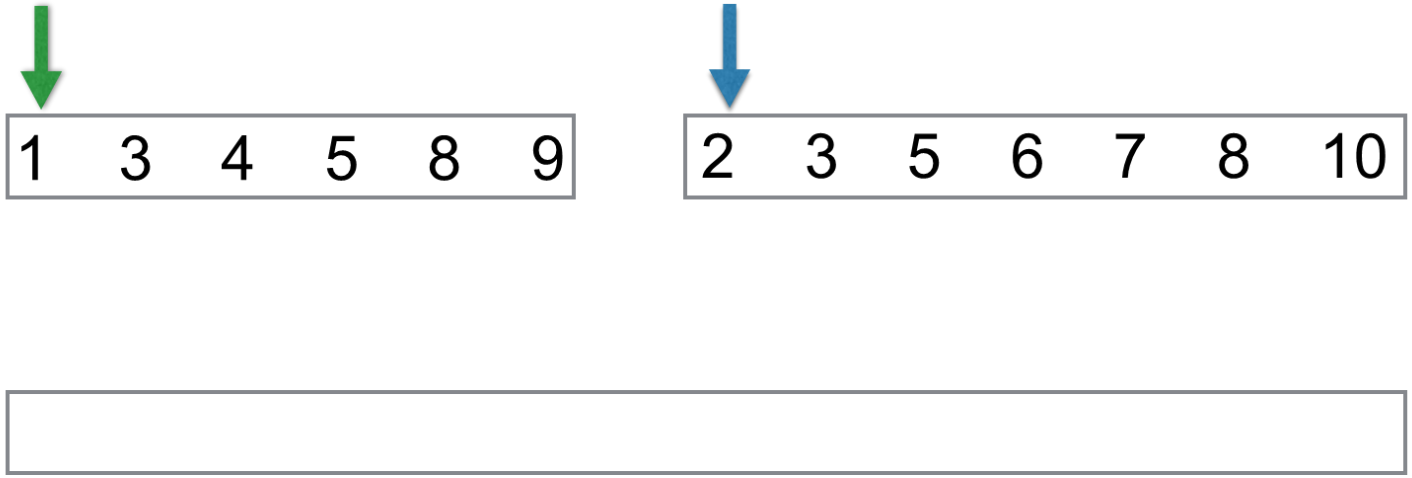
```
1.  void merge_sort(int * a, int * aux, int l, int r) {
2.      if (l < r) {
3.          int m = (l + r) / 2;
4.          merge_sort(a, aux, l, m);
5.          merge_sort(a, aux, m + 1, r);
6.          merge(a, aux, l, m, r);
7.      }
8.  }
```

- Для заданного *N* делим задачу на подзадачи всегда одинаковым образом (в отличии от быстрой сортировки)
- Время работы алгоритма не зависит от входных данных (только от размера задачи)
- Основная работа – после выполнения рекурсивных вызовов при слиянии

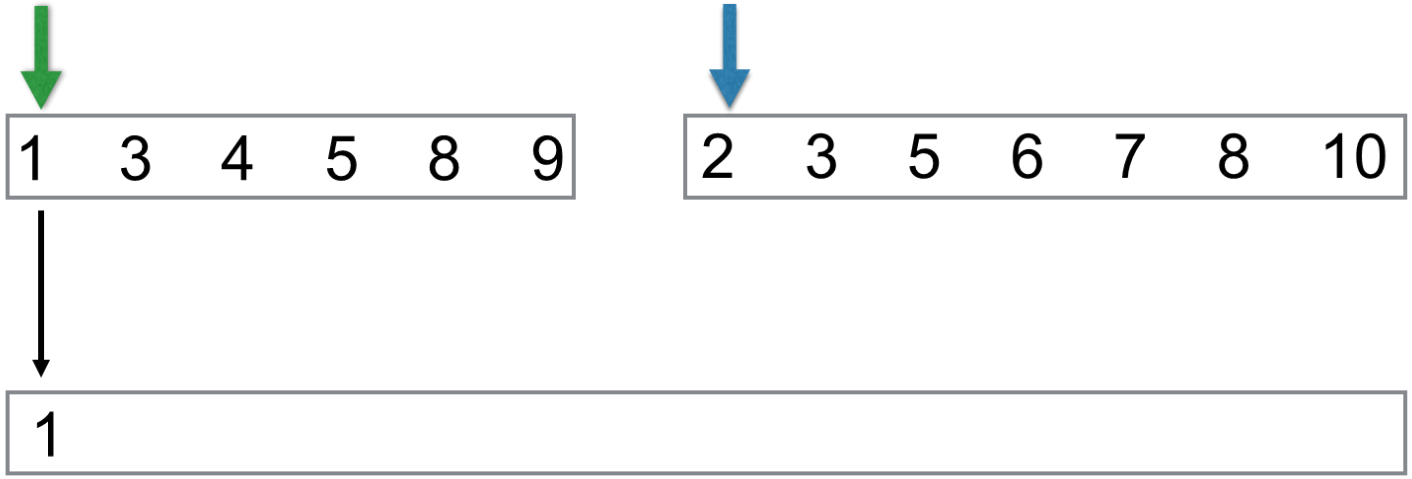
Сортировка слиянием



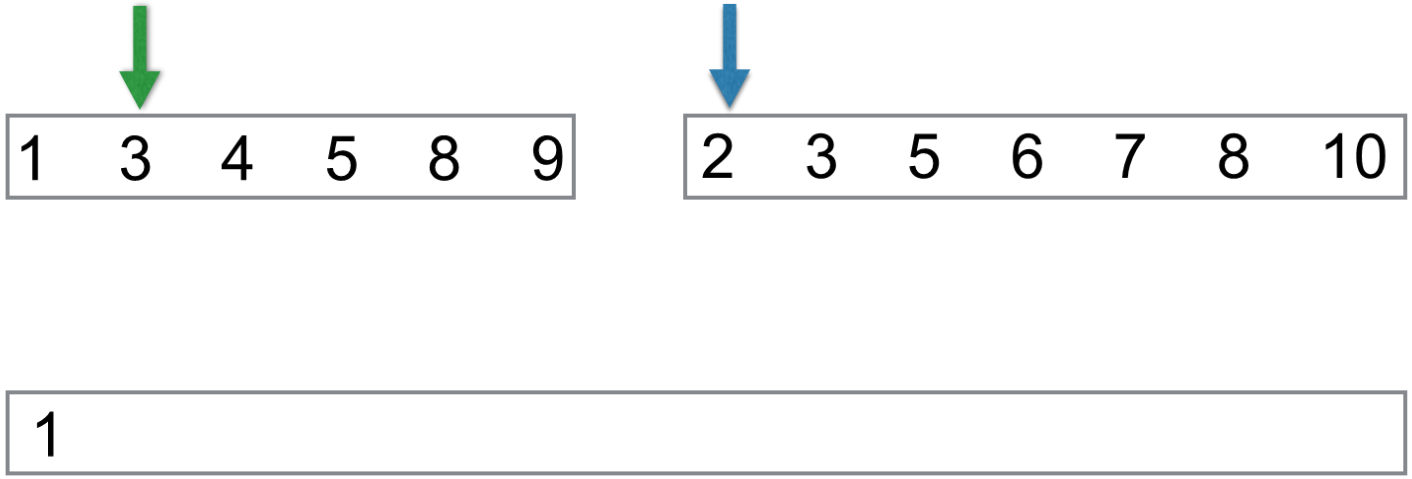
Сортировка слиянием



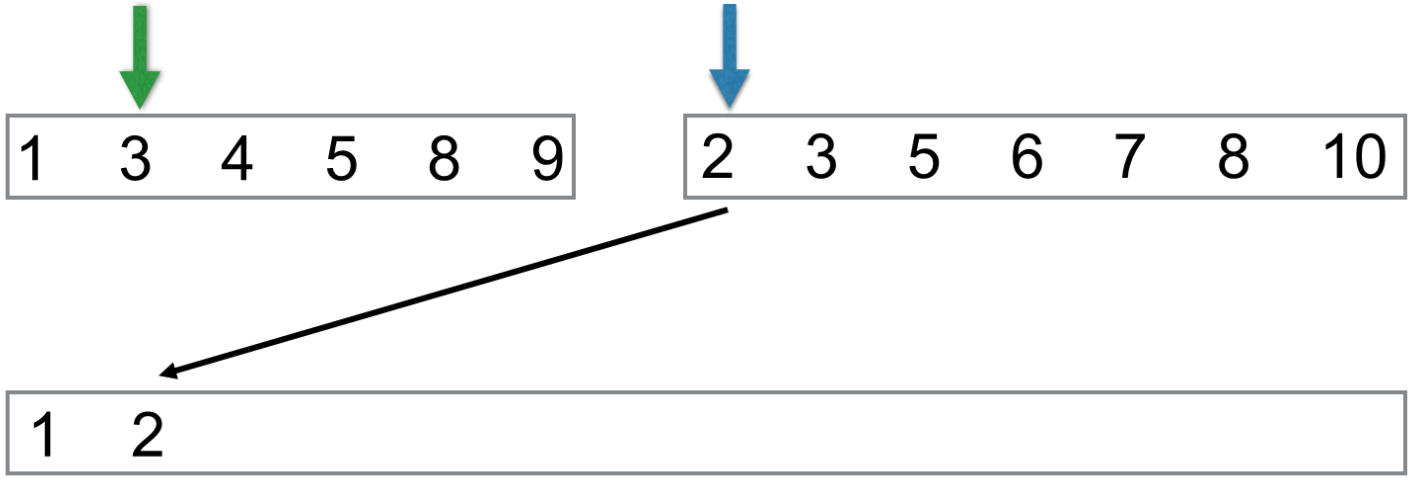
Сортировка слиянием



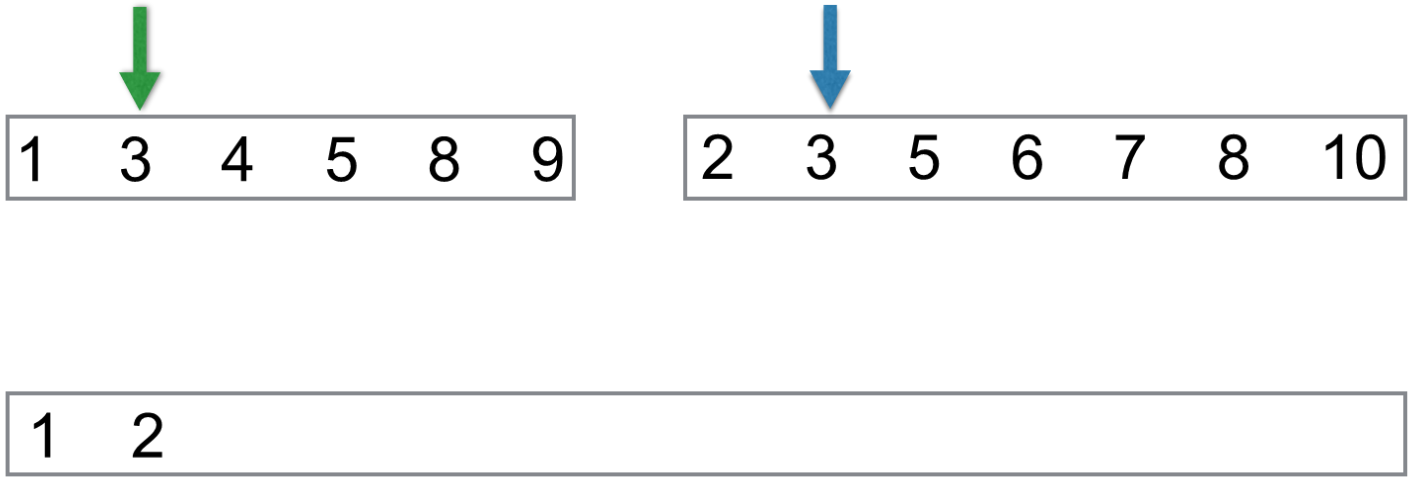
Сортировка слиянием



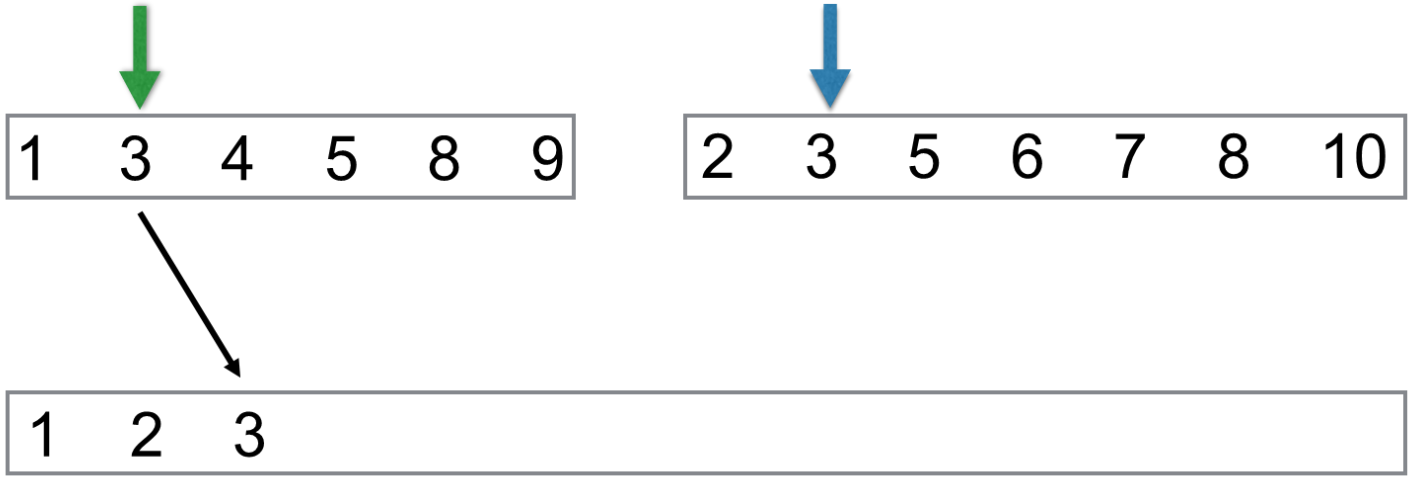
Сортировка слиянием



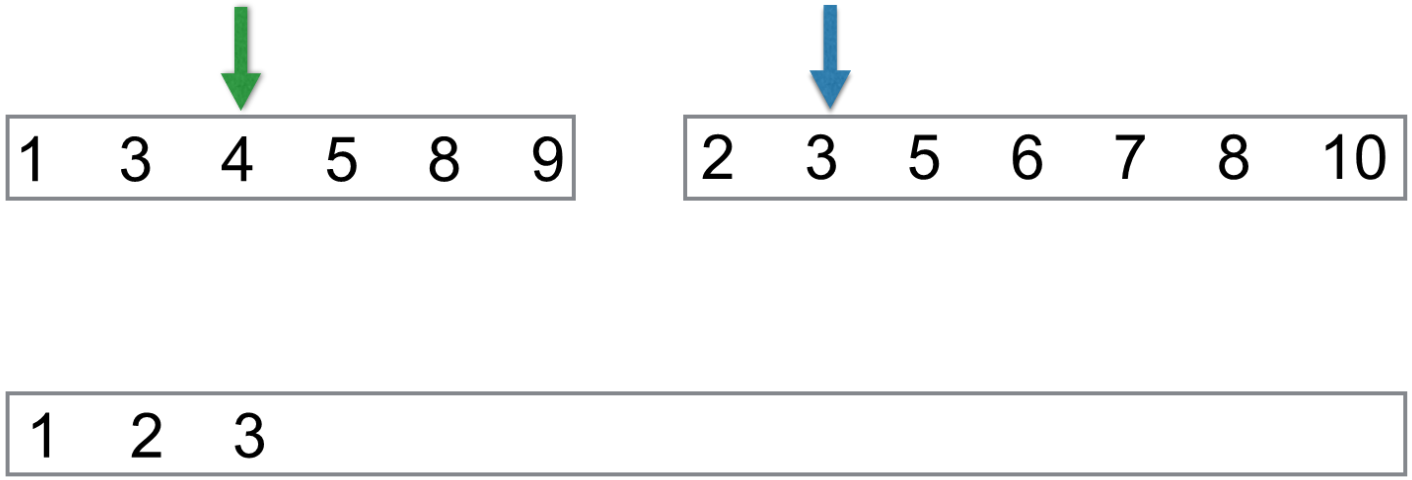
Сортировка слиянием



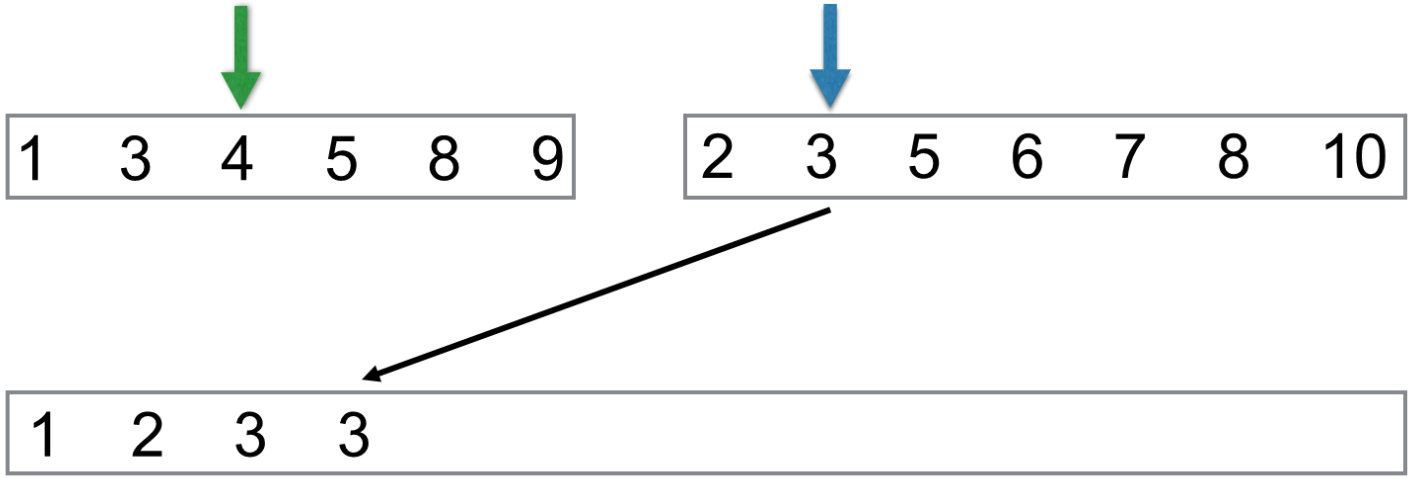
Сортировка слиянием



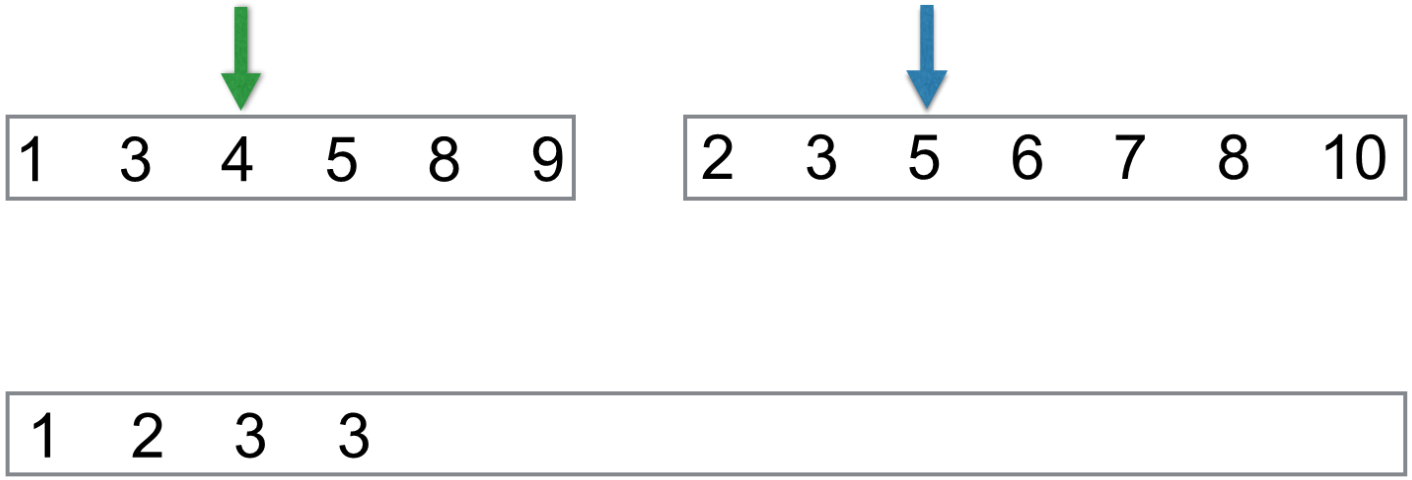
Сортировка слиянием



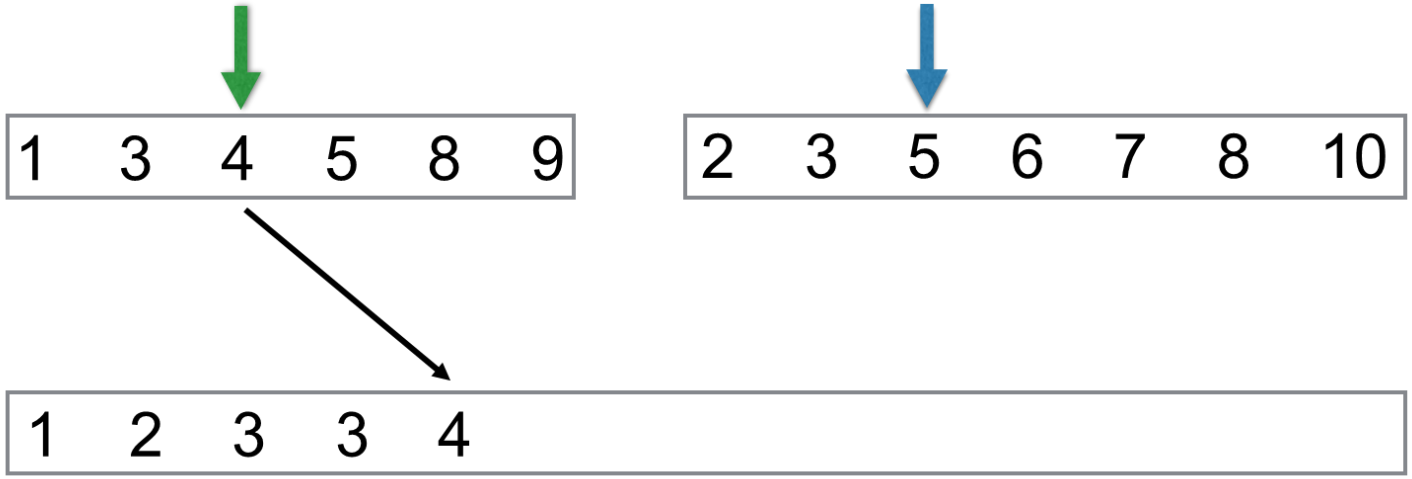
Сортировка слиянием



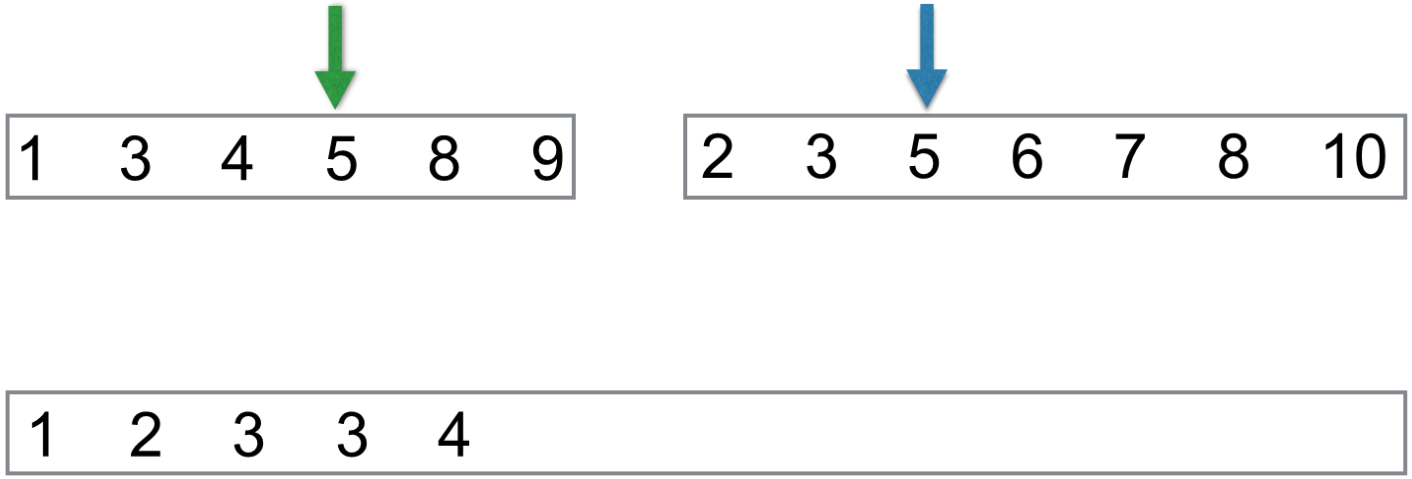
Сортировка слиянием



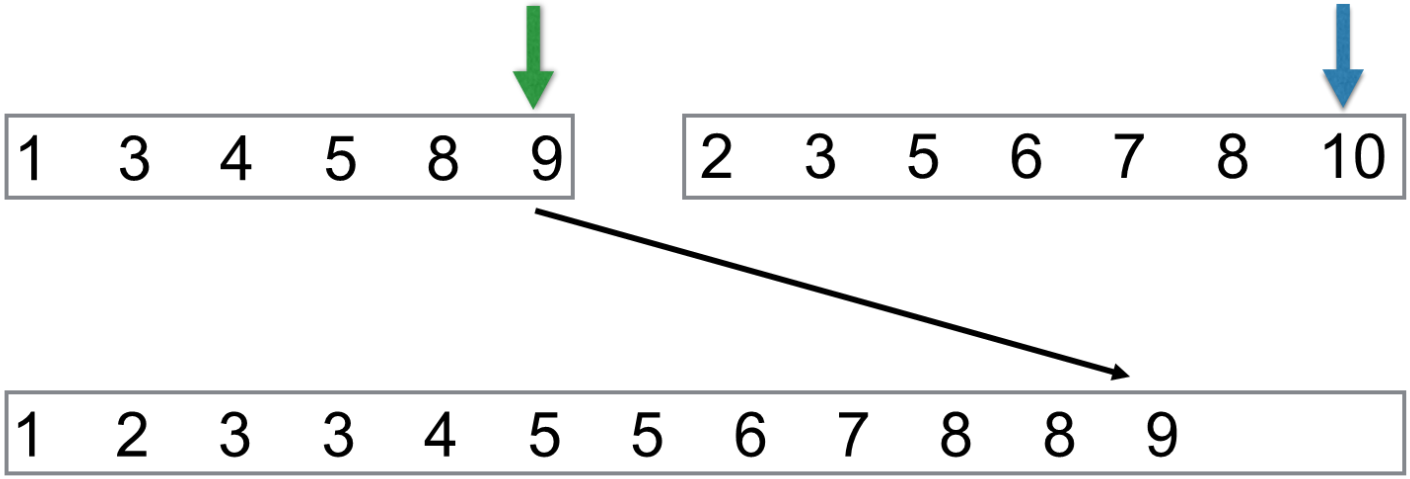
Сортировка слиянием



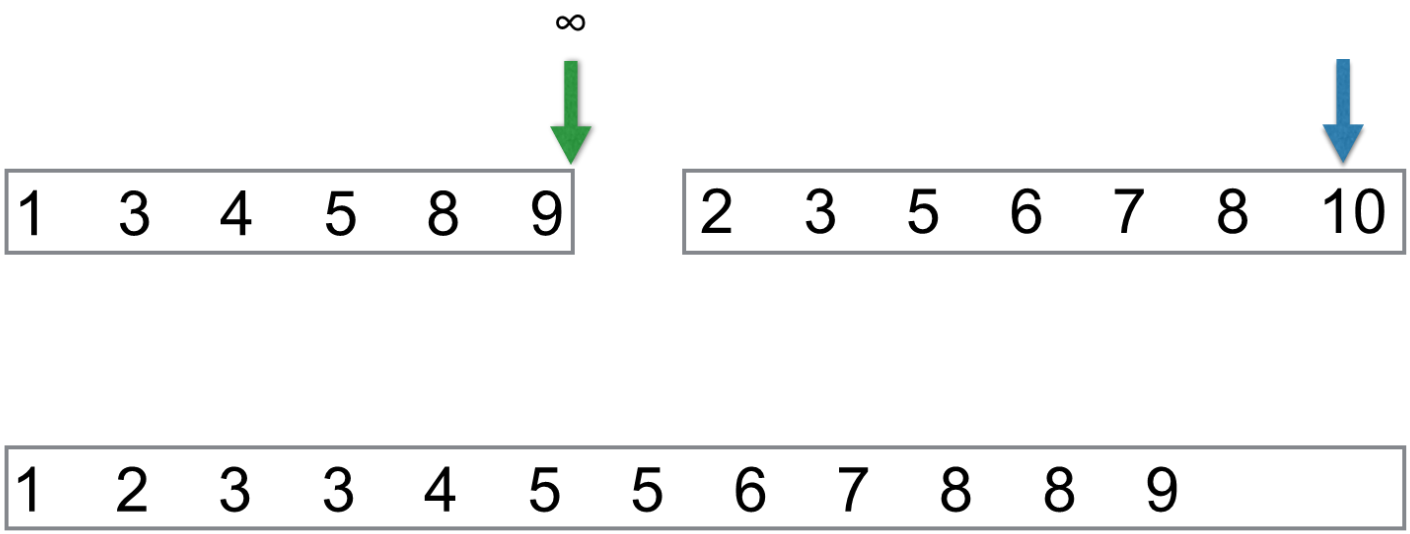
Сортировка слиянием



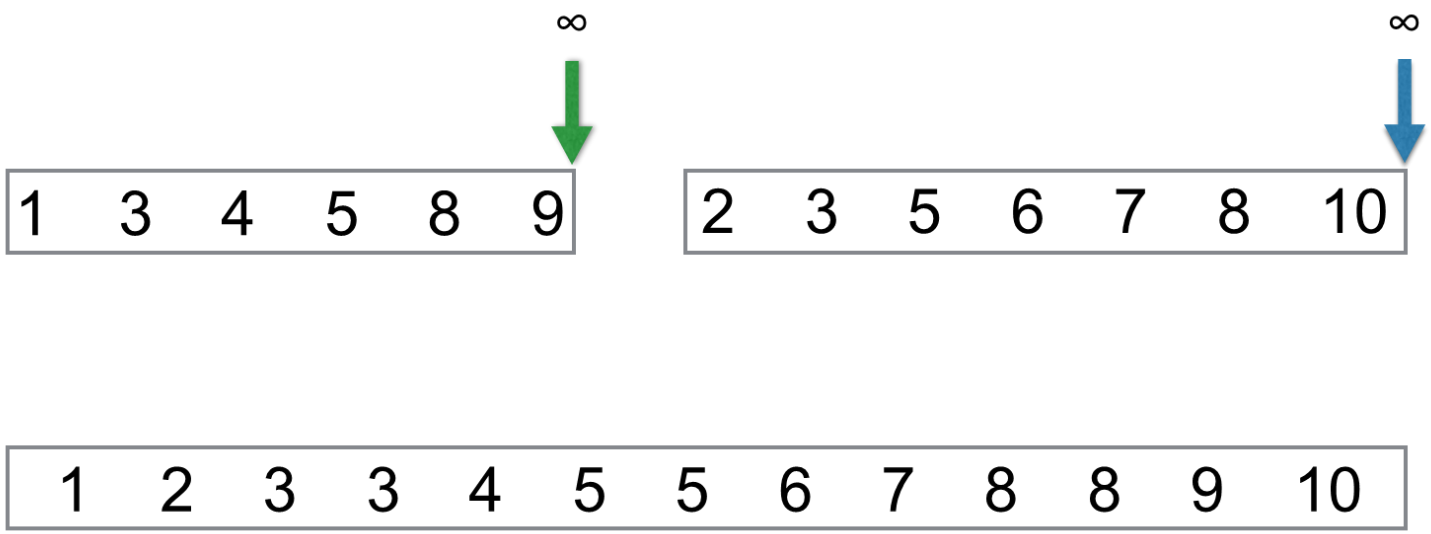
Сортировка слиянием



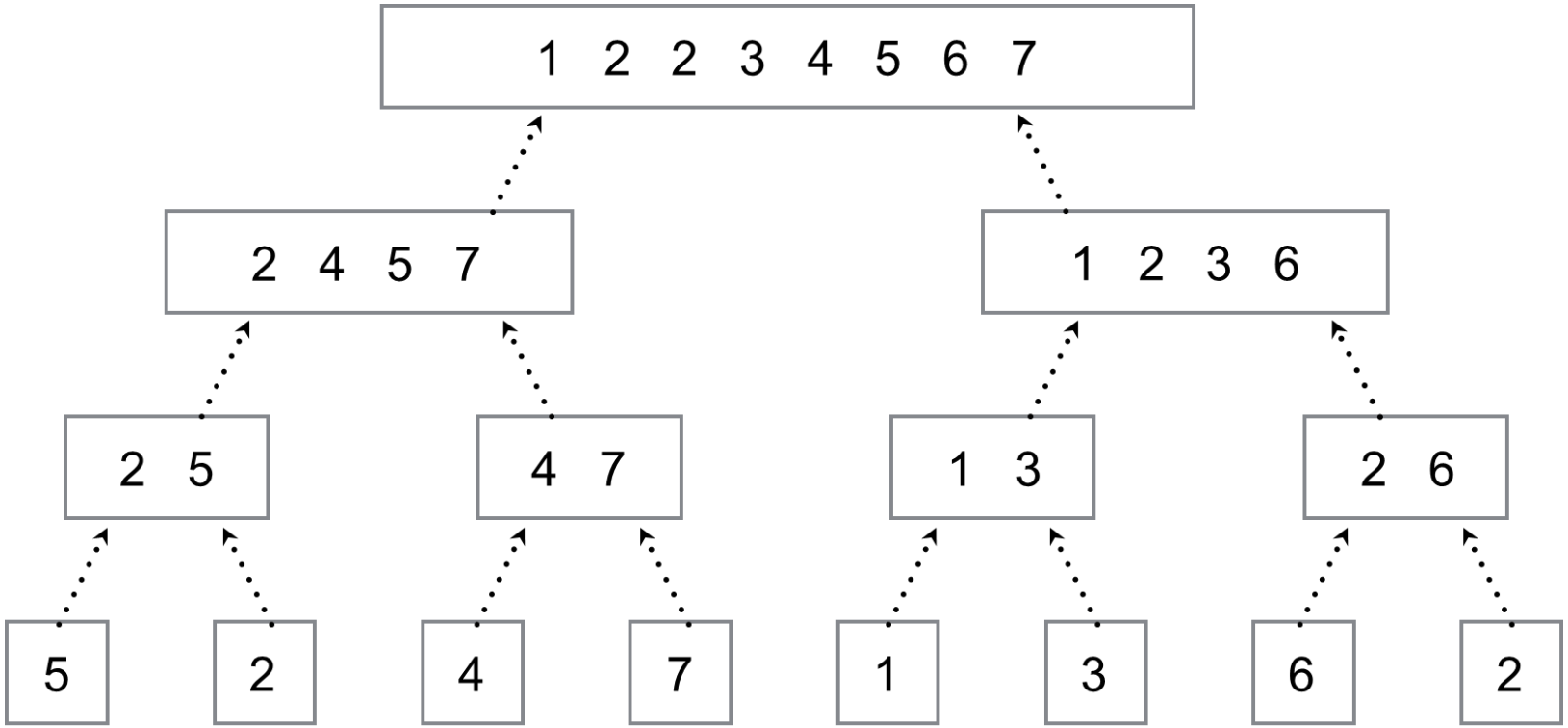
Сортировка слиянием



Сортировка слиянием



Сортировка слиянием



- Можно считать, что $N = 2^k$, т.е. $k = \log N$; почему?
- $T(N) = \Theta(N) + 2T(N/2)$

Сортировка слиянием

- Слияние

```
1. void merge(int * a, int * aux, int l, int m, int r) {
2.     int i = l;
3.     int j = m + 1;
4.     for (int k = l; k <= r; ++k) aux[k] = a[k];
5.
6.     for (int k = l; k <= r; ++k) {
7.         if (i > m) { a[k] = aux[j++]; continue; }
8.         if (j > r) { a[k] = aux[i++]; continue; }
9.         if (aux[j] < aux[i]) { a[k] = aux[j++]; }
10.        else                 { a[k] = aux[i++]; }
11.    }
12.}
```

- Стабильная; в C++ std::stable_sort – обычно сортировка слиянием
- Оценка сложности: $\Theta(N \log N)$; доп. память: $O(N)$

Сортировка слиянием

- Слияние через битонический (bitonic) порядок

```
1. void merge_bitonic(int * a, int * aux, int l, int m, int r) {
2.     int i, j;
3.     for (i = m + 1; i > l; --i) aux[i-1] = a[i-1];
4.     for (j = m; j < r; ++j) aux[r+m-j] = a[j+1];
5.     for (int k = l; k <= r; ++k) {
6.         if (aux[j] < aux[i]) {
7.             a[k] = aux[j--];
8.         } else {
9.             a[k] = aux[i++];
10.        }
11.    }
12. }
```

- Стабильность?

Сортировка слиянием

- Устранение копирования при merge

```
1. void sort(int * a, int N) {
2.     int * aux = new int[N];
3.     for (int k = 0; k < N; ++k) aux[k] = a[k];
4.     merge_sort(a, aux, 0, N-1);
5.     delete[] aux;
6. }
```

```
1. void merge_sort(int * a, int * aux, int l, int r) {
2.     if (l < r) {
3.         int m = (l + r) / 2;
4.         merge_sort(aux, a, l, m);
5.         merge_sort(aux, a, m + 1, r);
6.         merge(a, aux, l, m, r);
7.     }
8.}
```

Bottom-Up сортировка слиянием

ITIVITI

- Альтернатива Top-Down
- Нерекурсивная версия
- Пройдемся по массиву и сольем подмассивы размера 1
- Продолжим далее для подмассивов размера 2, 4, 8, …
- Обход в обратном порядке vs Обход по уровням снизу вверх
- Дерево рекурсии и последовательность слияний для Top-Down при $N = 5$: 1-1; 2-1; 1-1; 3-2
- Последовательность слияний для Bottom-Up при $N = 5$: 1-1; 1-1; 2-2; 4-1
- Немного медленнее рекурсивной Top-Down версии; почему?

Bottom-Up сортировка слиянием


```
1. void merge_sort_BU(int * a, int * aux, int l, int r) {
2.     for (int sz = 1; sz <= r-l; sz = sz+sz)
3.         for (int i = l; i <= r-sz; i += sz+sz)
4.             merge(a, aux, i, i+sz-1, min(i+sz+sz-1, r));
5.}
```

- Последовательность слияний при $N = 5$

```
sz = 1
merge(a, aux, 0, 0, 1)
merge(a, aux, 2, 2, 3)

sz = 2
merge(a, aux, 0, 1, 3)

sz = 4
merge(a, aux, 0, 3, 4)
```

Обновление таблицы

ITIVITI

- «На проволоке» таблица лежит по строкам

	11	**22**	**33**
1	X	5	hello
2	Y	10	world
3	Z	15	string
4	X	11	string
5	X	13	itmo

2.11 = Z
1.22 = 0
2.33 = bye
1.22 = 1
1.11 = A

=>

1.11 = A
1.22 = 0
1.22 = 1
2.11 = Z
2.33 = bye

LET IT TRADE

УНИВЕРСИТЕТ ИТМО

Большие объекты

- Дан массив больших объектов:
  - $B = \langle \text{obj}_1, \text{obj}_2, \ldots, \text{obj}_N \rangle$
- Как лучше отсортировать его?

Большие объекты

- Дан массив больших объектов:
  - $B = \langle \text{obj}_1, \text{obj}_2, \ldots, \text{obj}_N \rangle$
- Можно отсортировать указатели на объекты или индексы:
  - $I = \langle 1, 2, \ldots, N \rangle$
  - Сортируем I так, чтобы выполнялось следующее условие:
  - $B[I[1]] \le B[I[2]] \le \ldots \le B[I[N]]$

Число инверсий

ITIVITI

LET IT TRADE

- Число инверсий в массиве A [] равно числу пар (i , j), таких, что $i < j$ и $A[i] > A[j]$
- Используется для оценки численной схожести нескольких списков с рейтингом (оценками) пользователей (numerical similarity measure between ranked lists)
- Сколько инверсий в массиве: 1, 2, 3, 4, 5, 6, 7, 8 ?
- Сколько инверсий в массиве: 1, 4, 5, 3, 2, 6, 7, 8 ?
  - Как вы считали? Какая сложность подсчета?
  - Число инверсий определяется числом пересечений отрезков, соединяющих одинаковые элементы

Число инверсий

- Метод «разделяй и властвуй»
- Хотим $\Theta(N \log N)$
- Используем сортировку слиянием
  - Левая и правая инверсия (можем посчитать рекурсивно)
  - Расщепленная инверсия (как посчитать за линейное время ?)
- Если в массиве $A[]$ отсутствуют расщепленные инверсии, как соотносятся левый и правый подмассивы $A[]$?
- Число расщепленных инверсий, включающих элемент a_j из правого подмассива, совпадает с числом элементов в левом подмассиве на момент копирования a_j

Коэффициент ранговой корреляции Кендалла

ITIVITI

- С помощью сортировки и нехитрой математики Алиса легко сможет выбрать, с кем пойти в кино

ID	Фильм	Алиса	Вася	Саша
1	Звездные войны	6	5	1
2	Терминатор	5	6	2
3	Аватар	2	2	5
4	Властелин колец	3	1	4
5	Гарри Поттер	4	4	3
6	Титаник	1	3	6

$$\tau = \frac{\text{число пар в одном порядке} - \text{число инверсий}}{\text{число всех пар}}$$

LET IT TRADE

ITIVITI

LET IT TRADE

Спасибо за внимание!

УНИВЕРСИТЕТ ИТМО