Рекурсия

Пример бесконечной рекурсии

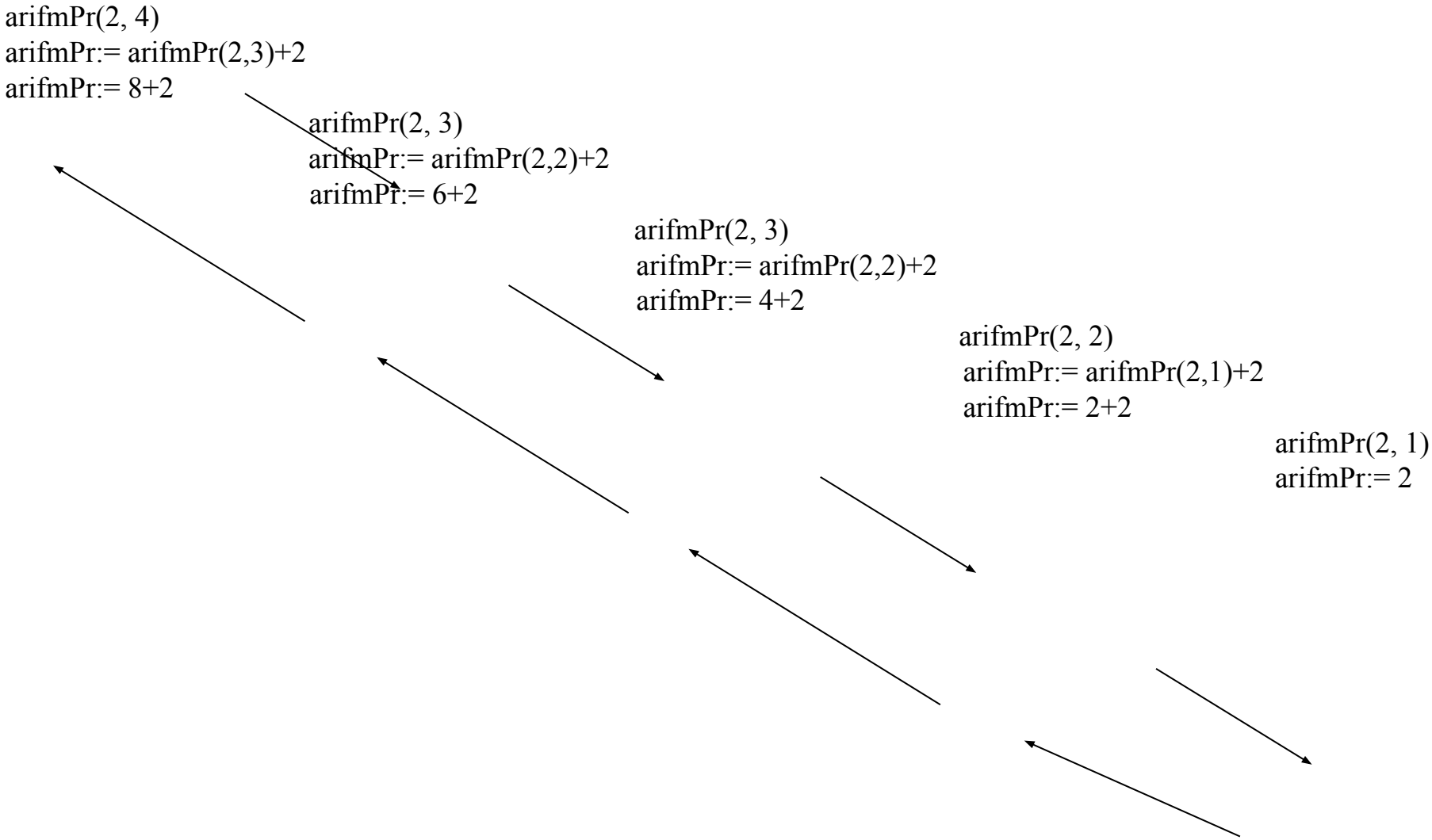
У попа была собака, он её любил,
Она съела кусок мяса, он её убил,
В землю закопал,
Надпись написал:

> У попа была собака, он её любил,
> Она съела кусок мяса, он её убил,
> В землю закопал,
> Надпись написал:

Рекурсивная функция

```
function arifmPr(base, iter: integer): integer;
begin
  if iter = 1 then
    arifmPr:= base
  else
    arifmPr:= arifmPr(base, iter - 1) + base;
end;
```

Рекурсивная функция



Сортировка слиянием
(Mergesort)

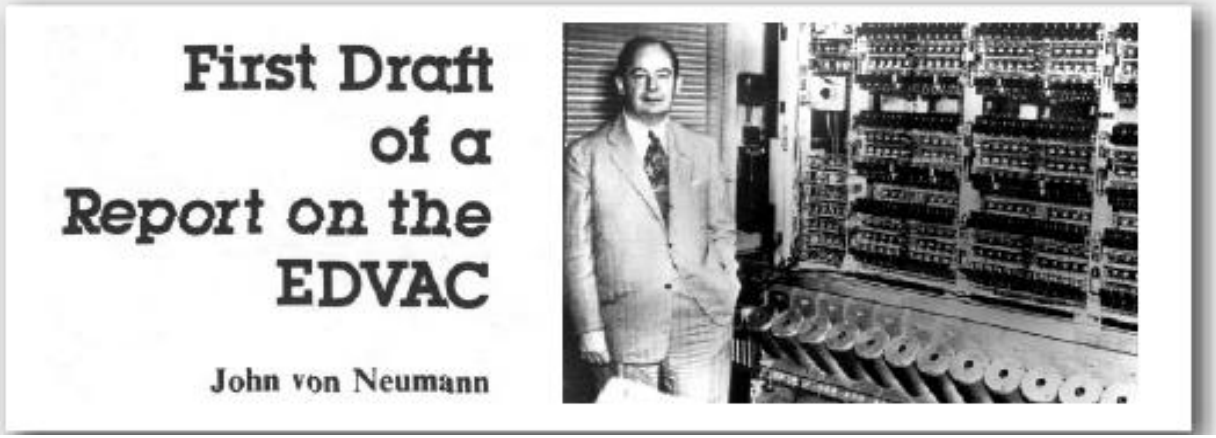
Два классических алгоритма сортировки

- Критические компоненты в мировой вычислительной инфраструктуре
  - Понимание научных основ этих алгоритмов даст нам возможность разрабатывать прикладные системы для решения реальных задач
  - Быстрая сортировка входит в десятку самых полезных алгоритмов, разработанных в 20-м веке

Сортировка слиянием

- Основной план
  - Разделить массив на две половины
  - Рекурсивно отсортировать каждую половину
  -

Mergesort overview



Сортировка слиянием

- Цель. Получить два отсортированных подмассива от a[lo] до a[mid] и от a[mid+1] до a[hi]
- Видео 1

Слияние: реализация на Java

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
   assert isSorted(a, lo, mid);     // precondition: a[lo..mid]   sorted
   assert isSorted(a, mid+1, hi);   // precondition: a[mid+1..hi] sorted

   for (int k = lo; k <= hi; k++)                                          copy
      aux[k] = a[k];

   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)                                          merge
   {
      if      (i > mid)                a[k] = aux[j++];
      else if (j > hi)                 a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                             a[k] = aux[i++];
   }

   assert isSorted(a, lo, hi);      // postcondition: a[lo..hi] sorted
}
```

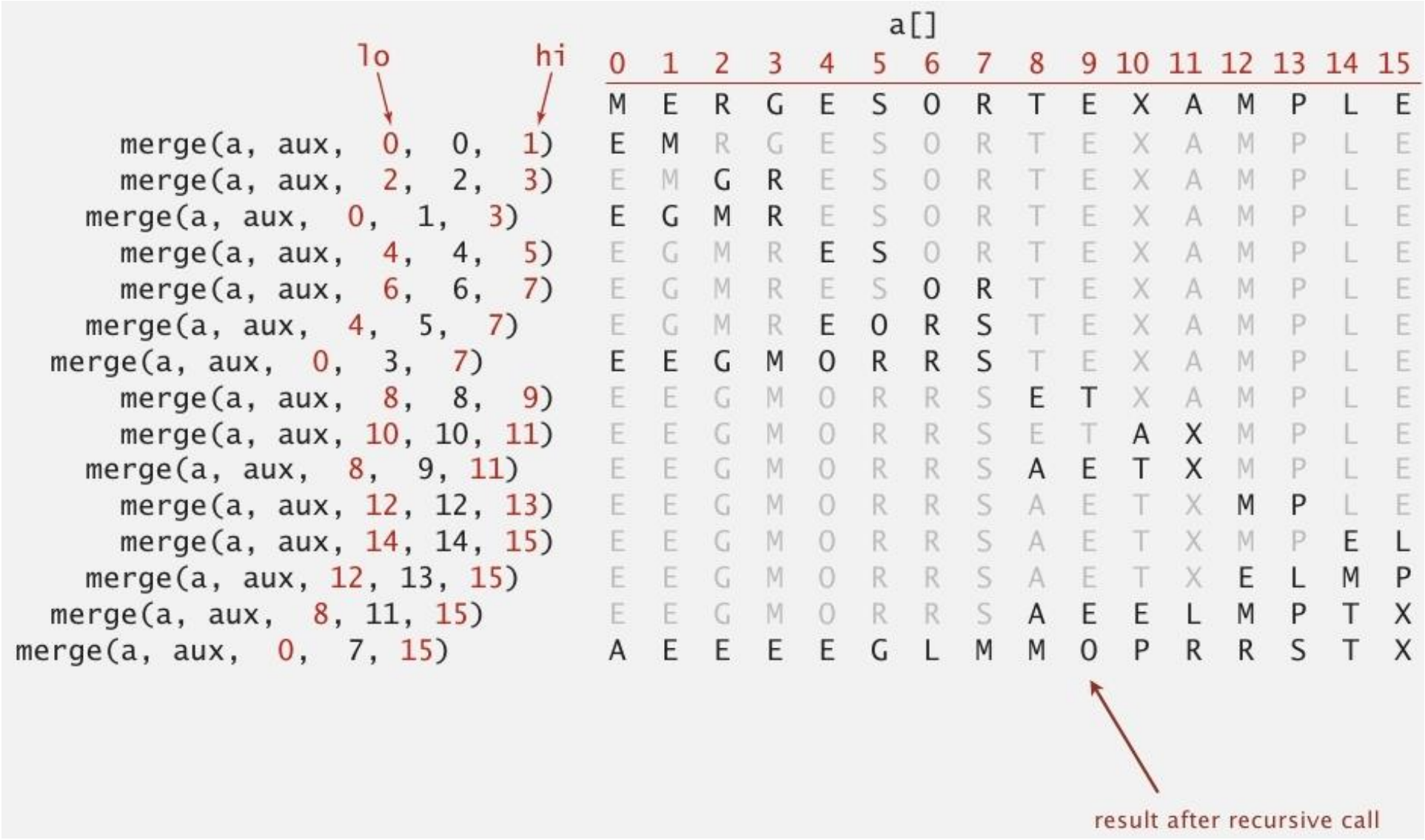
Assertions

- Assertion. Оператор для тестирования программы
  - Помогает обнаружить логические ошибки
  - Документирует код
- Java assert оператор. Генерирует ис
```
assert isSorted(a, lo, hi);
```
о, если условие не верно
- Можно включать и выключать в процессе работы => не влияет на производительность в процессе р
```
java -ea MyProgram   // enable assertions
java -da MyProgram   // disable assertions (default)
```
- Лучшие варианты использования. Использовать для проверки инвариантов. Отключать в отлаженном коде

Сортировка слиянием: реализация на Java

```
public class Merge
{
   private static void merge(...)
   {  /* as before */  }

   private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
   {
      if (hi <= lo) return;
      int mid = lo + (hi - lo) / 2;
      sort(a, aux, lo, mid);
      sort(a, aux, mid+1, hi);
      merge(a, aux, lo, mid, hi);
   }

   public static void sort(Comparable[] a)
   {
      aux = new Comparable[a.length];
      sort(a, aux, 0, a.length - 1);
   }
}
```

Сортировка слиянием: трассировка



- Видео 2

Сортировка слиянием: эмпирический анализ

- Оценка времени выполнения:
  - На ПК 10^8 сравнений/секунду
  - На суперкомпьютере 10^{12} сравнений/секунду

	insertion sort (N^2)			mergesort (N log N)		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

- Вывод. Хорошие алгоритмы лучше, чем суперкомпьютер

Сортировка слиянием: количество сравнений и обращений к массиву

- Утверждение. Сортировка слиянием использует NlogN сравнений и 6 NlogN обращений для сортировки массива размером N
- Доказательство. C(N) — количество сравнений, A(N) — количество обращений

$$C(N) \le C(\lceil N/2 \rceil) + C(\lfloor N/2 \rfloor) + N \quad \text{for } N > 1, \text{ with } C(1) = 0.$$

left half — right half — merge

$$A(N) \le A(\lceil N/2 \rceil) + A(\lfloor N/2 \rfloor) + 6N \quad \text{for } N > 1, \text{ with } A(1) = 0.$$

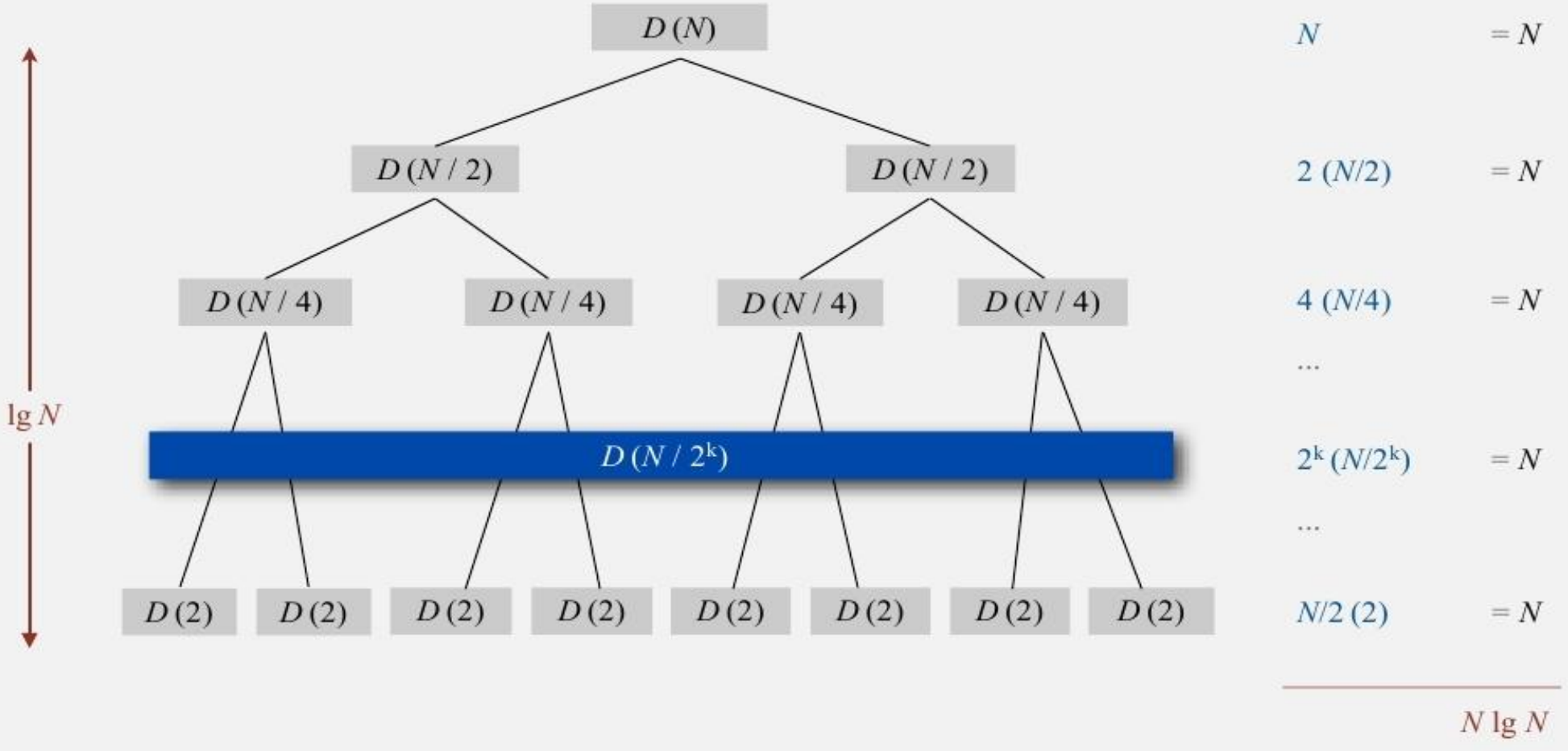
- Решим рекуррентность для N. N — степень двойки

$$D(N) = 2\,D(N/2) + N, \text{ for } N > 1, \text{ with } D(1) = 0.$$

Разделяй и властвуй

- Утверждение. Если D(N) удовлетворяет D(N) = 2D(N/2) + N, где N > 1 и D(1) = 0, то D(N) =

Pf 1. [assuming N is a power of 2]



Разделяй и властвуй

- Утверждение. Если D(N) удовлетворяет D(N) = 2D(N/2) + N, где N > 1 и D(1) = 0, то D(N) = Nlo

Pf 2. [assuming N is a power of 2]

$D(N) = 2\,D(N/2) + N$	given
$D(N) / N = 2\,D(N/2) / N + 1$	divide both sides by N
$= D(N/2) / (N/2) + 1$	algebra
$= D(N/4) / (N/4) + 1 + 1$	apply to first term
$= D(N/8) / (N/8) + 1 + 1 + 1$	apply to first term again
$\ldots$	
$= D(N/N) / (N/N) + 1 + 1 + \ldots + 1$	stop applying, D(1) = 0
$= \lg N$	

Разделяй и властвуй

- Утверждение. Если D(N) удовлетворяет D(N) = 2 D(N/2) + N, где N > 1 и D(1) = 0, то D(N) = $N\log_2 N$

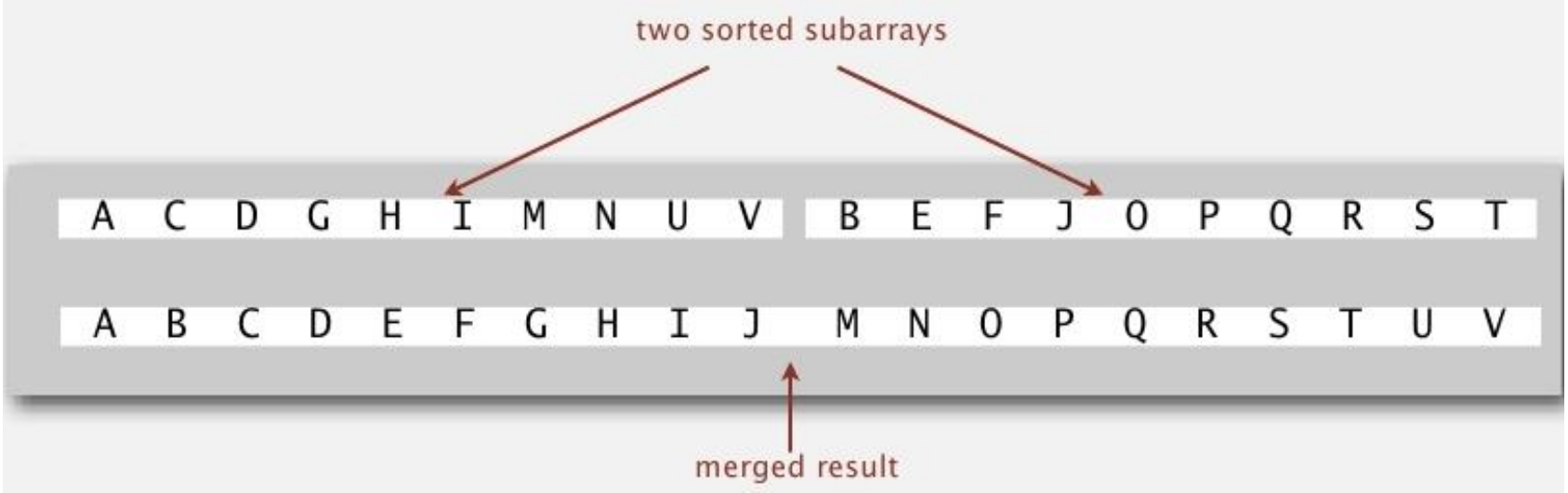
Pf 3. [assuming N is a power of 2]

- Base case: $N = 1$.
- Inductive hypothesis: $D(N) = N \lg N$.
- Goal: show that $D(2N) = (2N) \lg (2N)$.

$D(2N) = 2\,D(N) + 2N$	given
$= 2\,N \lg N + 2N$	inductive hypothesis
$= 2\,N\,(\lg (2N) - 1) + 2N$	algebra
$= 2\,N \lg (2N)$	QED

Анализ сортировки слиянием: память

- Утверждение. Сортировка слиянием использует дополнительную память пропорциональную N
- Массиву aux[] нужен N ячеек для последнего слияния



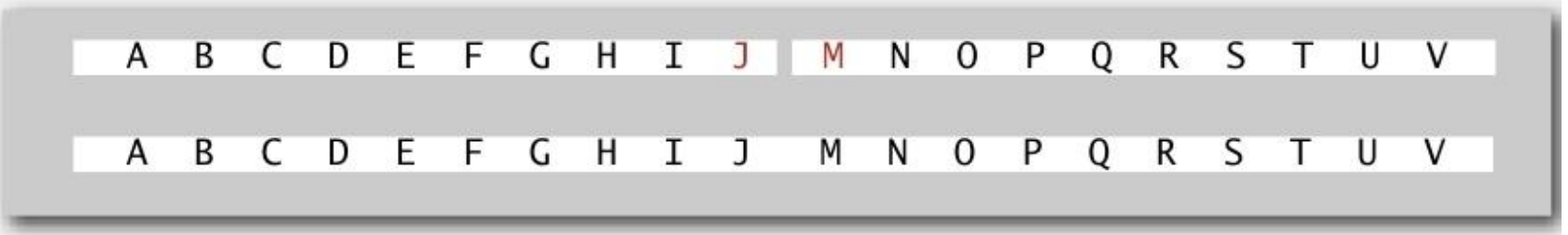
Сортировка слиянием: реализация

- Используйте сортировку вставками для маленьких подмассивов
  - Сортировка слиянием очень дорогая для маленьких подмассивов
  - Подмассивы для сортировки вставками ~ 7

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
   if (hi <= lo + CUTOFF - 1)
   {
      Insertion.sort(a, lo, hi);
      return;
   }
   int mid = lo + (hi - lo) / 2;
   sort (a, aux, lo, mid);
   sort (a, aux, mid+1, hi);
   merge(a, aux, lo, mid, hi);
}
```

Сортировка слиянием: реализация

- Остановка если все отсортировано
  - Если самый большой элемент первой половины <= самому маленькому элементу второй половины, то подмассив отсортирован
  -




```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
   if (hi <= lo) return;
   int mid = lo + (hi - lo) / 2;
   sort (a, aux, lo, mid);
   sort (a, aux, mid+1, hi);
   if (!less(a[mid+1], a[mid])) return;
   merge(a, aux, lo, mid, hi);
}
```

Сортировка слиянием: реализация

- Ограничить копирование во вспомогательный массив

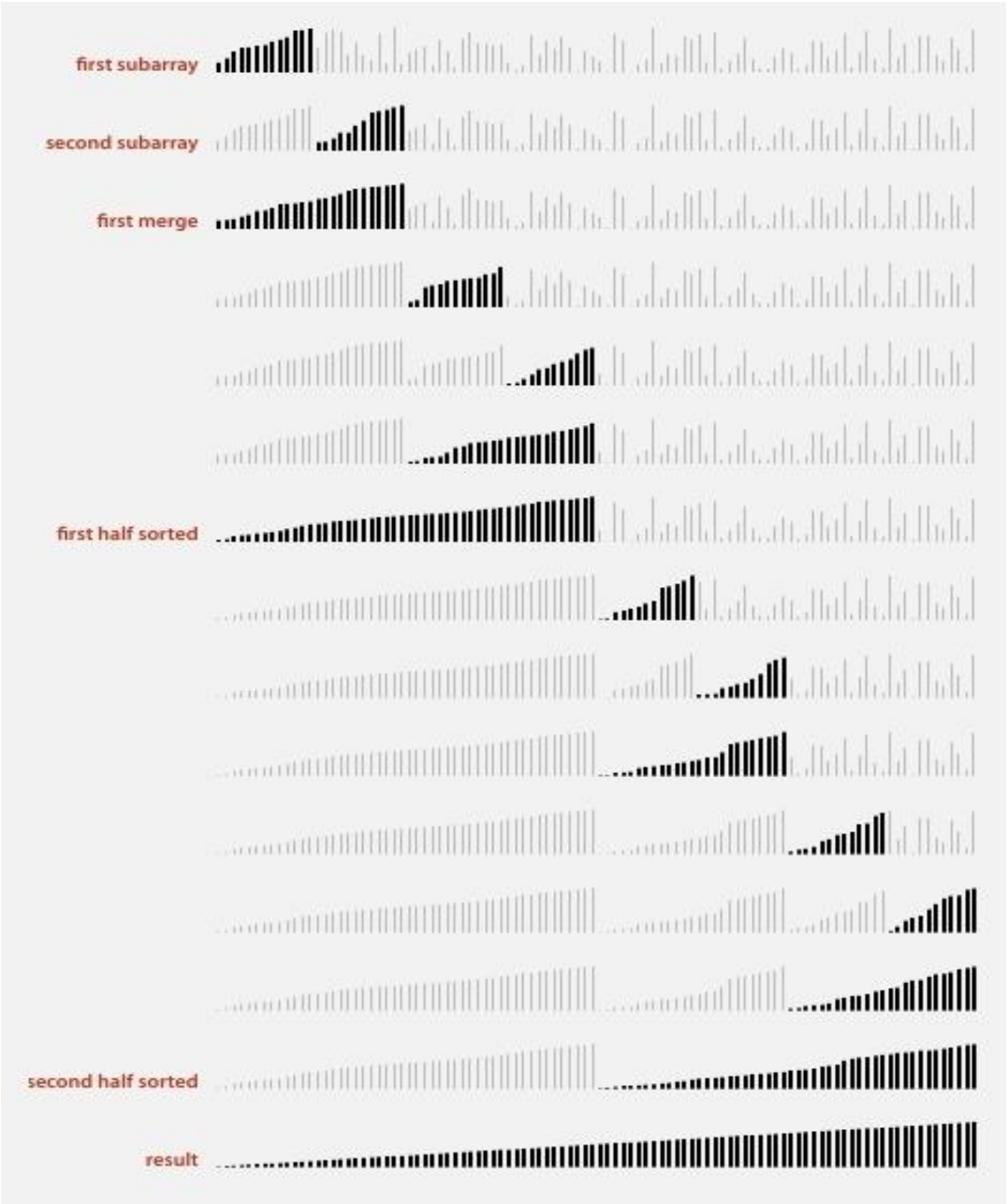
```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)
   {
      if      (i > mid)           aux[k] = a[j++];
      else if (j > hi)            aux[k] = a[i++];
      else if (less(a[j], a[i])) aux[k] = a[j++];
      else                        aux[k] = a[i++];
   }
}

private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
   if (hi <= lo) return;
   int mid = lo + (hi - lo) / 2;
   sort (aux, a, lo, mid);
   sort (aux, a, mid+1, hi);
   merge(a, aux, lo, mid, hi);
}
```

merge from a[] to aux[]

Note: sort(a) initializes aux[] and sets aux[i] = a[i] for each i.

switch roles of aux[] and a[]

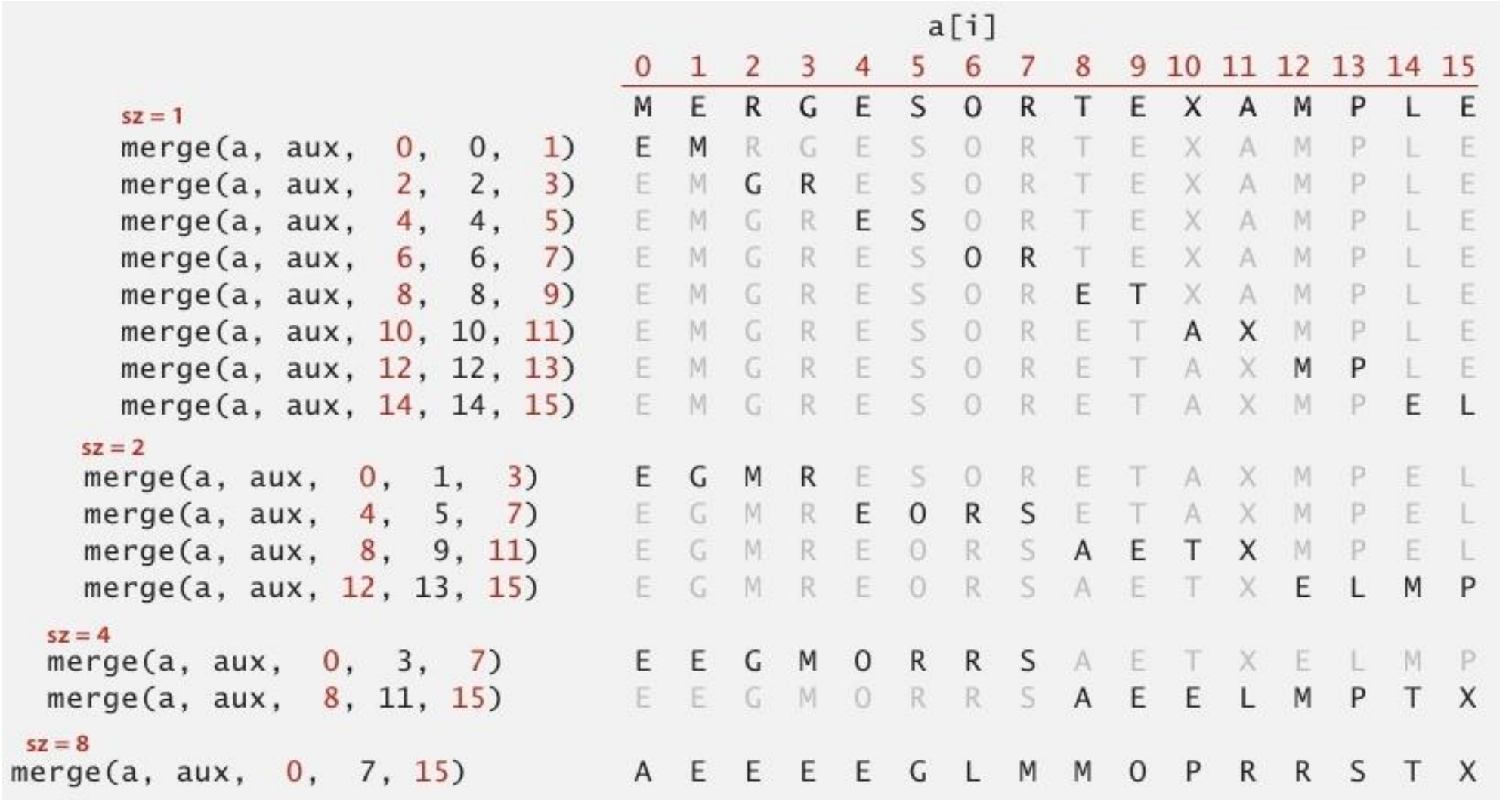
Сортировка слиянием: визуализация



Восходящая сортировка слиянием (Bottom-up mergesort)

Восходящая сортировка слиянием

- Начинаем со слияния подмассивов с размером 1
- Повторяем для подмассивов с размерами 2, 4, 8,

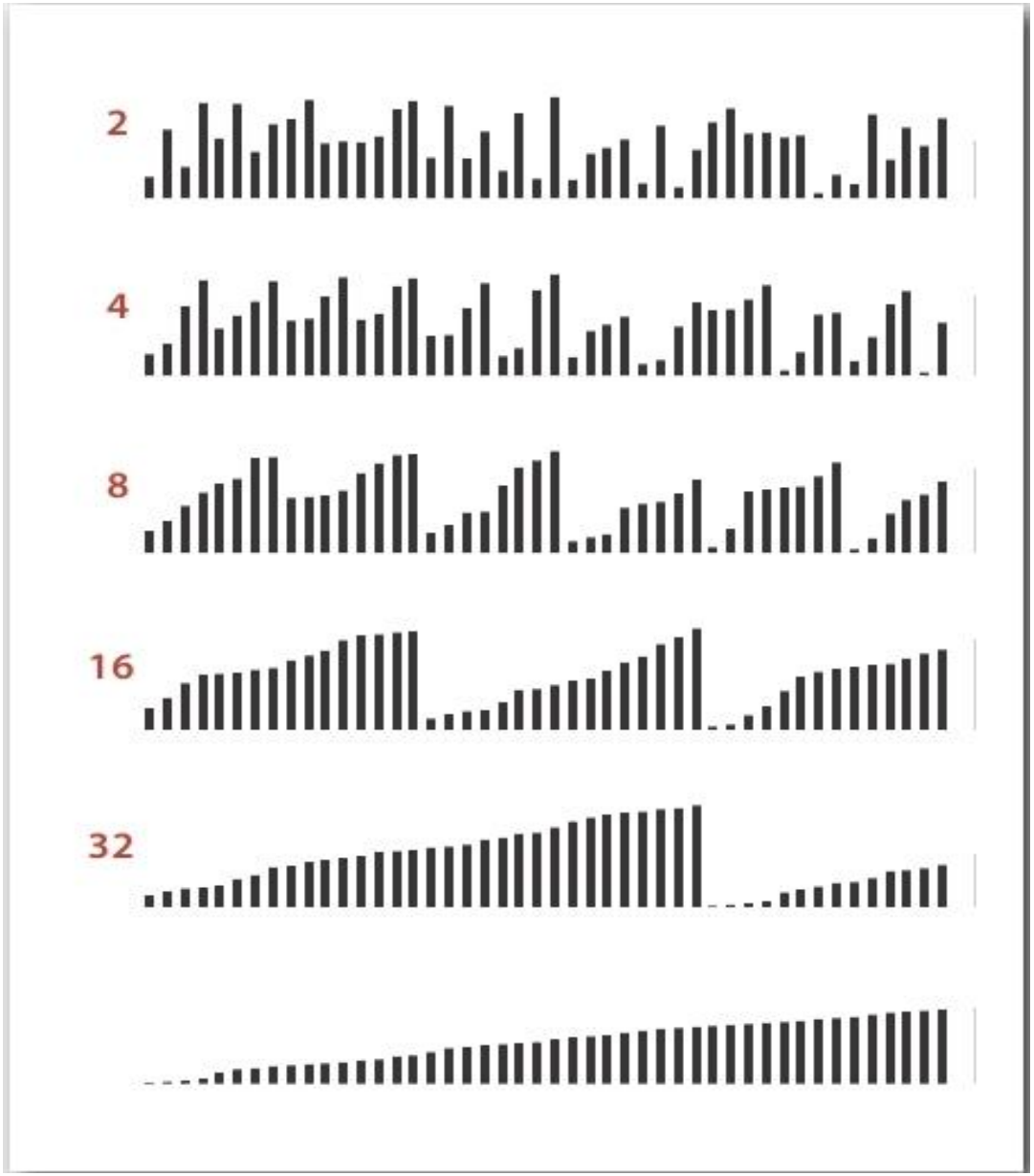


Восходящая сортировка слиянием: реализация на Java

```
public class MergeBU
{
   private static void merge(...)
   {  /* as before */  }

   public static void sort(Comparable[] a)
   {
      int N = a.length;
      Comparable[] aux = new Comparable[N];
      for (int sz = 1; sz < N; sz = sz+sz)
         for (int lo = 0; lo < N-sz; lo += sz+sz)
            merge(a, aux, lo, lo+sz-1, Math.min(lo+sz+sz-1, N-1));
   }
}
```

- Итог. Простая и не рекурсивная версия сортировки слиянием (работает на 10%

Восходящая сортировка слиянием: визуализация



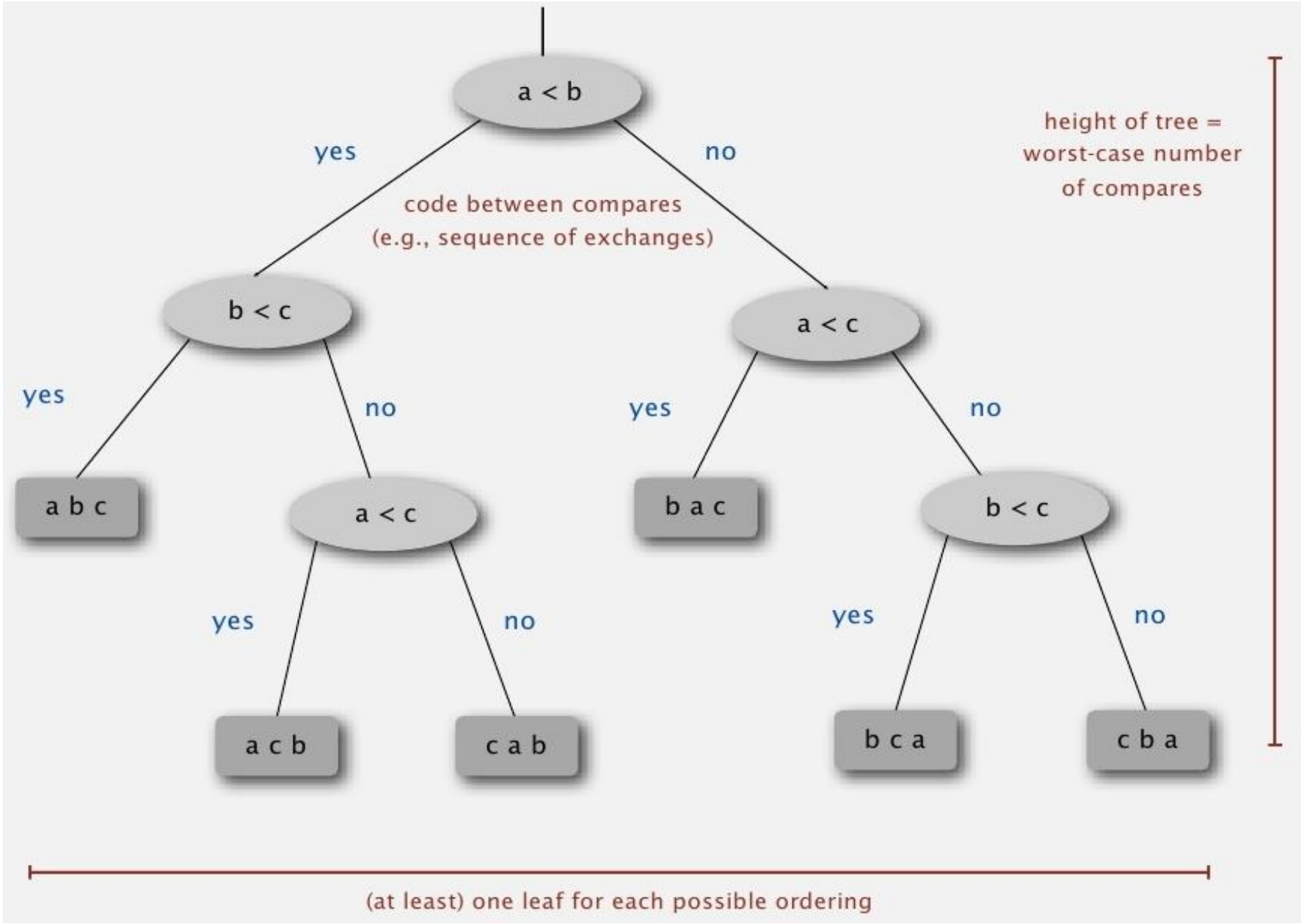
Сложность сортировки

Сложность сортировки

- Вычислительная сложность - основа для обучения эффективным алгоритмам для решения конкреной проблемы X

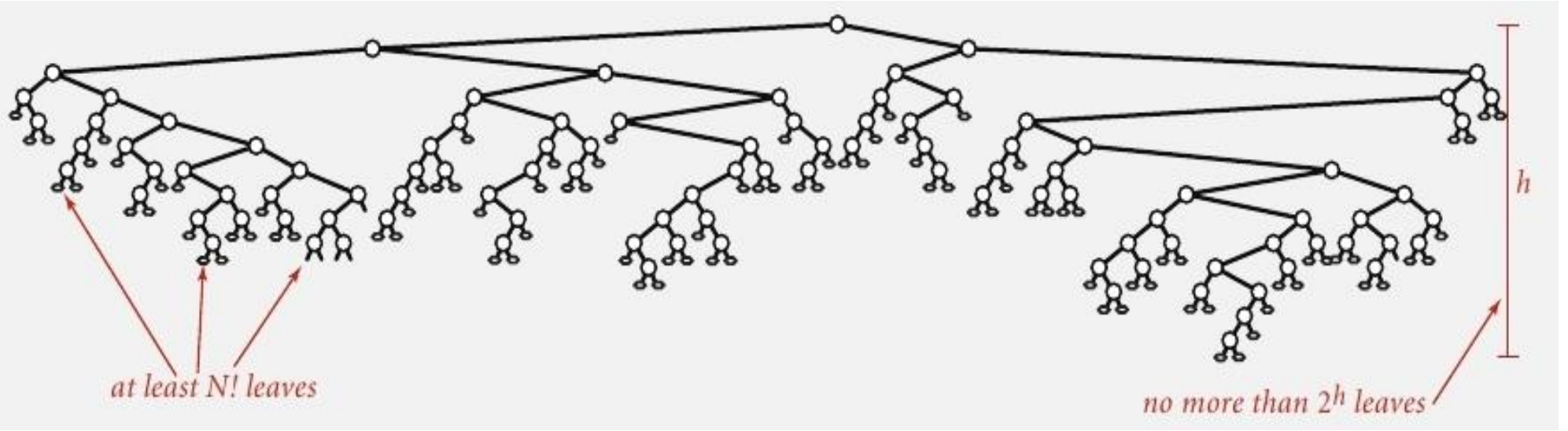
- Вычислительная модель. Возможные операции
- Стоимость модели. Количество операций
- Верхняя граница. Стоимость, гарантированная определенным алгоритмам для задачи X
- Нижняя граница. Доказанное ограничение стоимости для всех алгоритмов для задачи X
- Оптимальный алгоритм. Алгоритм с лучшей

Дерево принятия решений (для 3-х элементов)



Нижняя граница для сортировки на основе сравнений

- Утверждение. Ни один алгоритм сортировки, основанный на сравнениях, не может гарантировать упорядочить N элементов, выполнив менее $\log_2(N!) \sim N\log_2(N)$ сравнений
- Доказательство
  - Все элементы массива различны



Нижняя граница для сортировки на основе сравнений

- Утверждение. Ни один алгоритм сортировки, основанный на сравнениях, не может гарантировать упорядочить N элементов, выполнив менее $\log_2(N!) \sim N\log_2(N)$ сравнений
- Доказательство
  - Все элементы массива различны
  - В худшем [illegible] дет h
  - Бинарное [illegible] максимум 2^h листьев
  - N! <= количество листьев <= 2^n

$$2^h \geq \#\,\text{leaves} \geq N!$$

$$\Rightarrow h \geq \lg(N!) \sim N \lg N$$

Stirling's formula

Сложность сортировки

- Вычислительная модель. Возможные операции
- Стоимость модели. Количество операций
- Верхняя граница. Стоимость, гарантированная определенным алгоритмам для задачи X
- Нижняя граница. Доказанное ограничение стоимости для всех алгоритмов для задачи X
- Оптимальный алгоритм. Алгоритм с лучшей максимально возможной стоимостью для задачи X (когда верхняя и нижняя граница приблизительно совпадают)

Сложность сортировки

- Сравнения? Сортировка слиянием оптимальна по количеству сравнений
- Память? Сортировка слиянием не оптимальна по памяти

- Урок. Руководствуйся теорией
- Не пытайтесь создать алгоритм сортировки

Сложность сортировки

- Можно снизить нижнюю границу для сортировки если есть информация о:
  - Упорядоченности входных данных
  - Распределении значений ключей
  - Представлении ключей
- Частично-упорядоченный массив
- Дубликаты ключей. Зная распределение дубликатов во входных данных, мы можем отказаться от NlogN
- Представление ключей. Можем использовать цифровые/символьные сравнения ключей вместо сравнений номеров и строк

Устойчивость сортировок

Устойчивость

- Типичная задача. Отсортировать сначала по имени, а затем, по группам

```
Selection.sort(a, Student.BY_NAME);
```

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

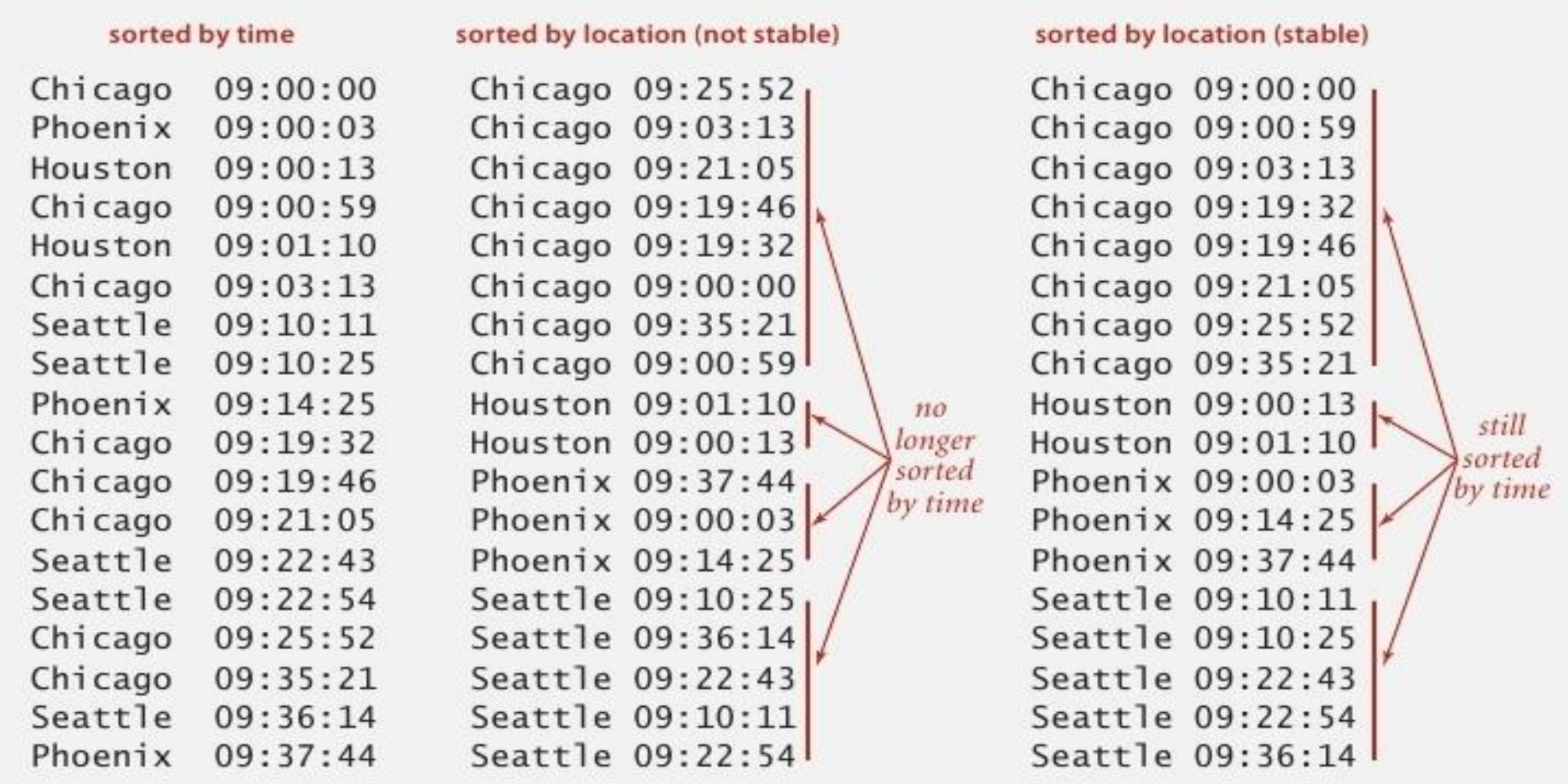
```
Selection.sort(a, Student.BY_SECTION);
```

Furia	1	A	766-093-9873	101 Brown
Rohde	2	A	232-343-5555	343 Forbes
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Andrews	3	A	664-480-0023	097 Little
Kanaga	3	B	898-122-9643	22 Brown
Gazsi	4	B	766-093-9873	101 Brown
Battle	4	C	874-088-1212	121 Whitman

- Студенты из группы 3 больше не отсортированы по именам

Устойчивость

- Сортировка вставками и сортировка слиянием устойчивы (но не выборочная и Шелла)



Устойчивость: сортировка вставками

- Сортировка вставками устойчива
- Равные элементы не передвигаются

```
public class Insertion
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
         for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
            exch(a, j, j-1);
   }
}
```

i	j	0	1	2	3	4
0	0	B_1	A_1	A_2	A_3	B_2
1	0	A_1	B_1	A_2	A_3	B_2
2	1	A_1	A_2	B_1	A_3	B_2
3	2	A_1	A_2	A_3	B_1	B_2
4	4	A_1	A_2	A_3	B_1	B_2
		A_1	A_2	A_3	B_1	B_2

Устойчивость: выборочная сортировка

- Выборочная сортировка не устойчива
- Передвижения элементов на большие расстояния может нарушить порядок

```
public class Selection
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
      {
         int min = i;
         for (int j = i+1; j < N; j++)
            if (less(a[j], a[min]))
               min = j;
         exch(a, i, min);
      }
   }
}
```

i	min	0	1	2
0	2	B_1	B_2	A
1	1	A	B_2	B_1
2	2	A	B_2	B_1
		A	B_2	B_1

Устойчивость: сортировка Шелла

- Сортировка Шелла не устойчива
- Передвижения элементов на большие расстояния

```
public class Shell
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      int h = 1;
      while (h < N/3) h = 3*h + 1;
      while (h >= 1)
      {
         for (int i = h; i < N; i++)
         {
            for (int j = i; j > h && less(a[j], a[j-h]); j -= h)
               exch(a, j, j-h);
         }
         h = h/3;
      }
   }
}
```

h	0	1	2	3	4
	B_1	B_2	B_3	B_4	A_1
4	A_1	B_2	B_3	B_4	B_1
1	A_1	B_2	B_3	B_4	B_1
	A_1	B_2	B_3	B_4	B_1

Устойчивость: сортировка слиянием

- Сортировка слиянием устойчива
- Если ключи равны, то берутся элементы из левог

```
private static void merge(...)
{
   for (int k = lo; k <= hi; k++)
      aux[k] = a[k];

   int i = lo, j = mid+1;
   for (int k = lo; k <= hi; k++)
   {
      if      (i > mid)                 a[k] = aux[j++];
      else if (j > hi)                  a[k] = aux[i++];
      else if (less(aux[j], aux[i])) a[k] = aux[j++];
      else                              a[k] = aux[i++];
   }
}
```

0	1	2	3	4	5	6	7	8	9	10
A_1	A_2	A_3	B	D	A_4	A_5	C	E	F	G