# Интернет Университет Суперкомпьютерных технологий

## Учебный курс

# *Введение в параллельные алгоритмы*

## Лекция 3

# Сортировка данных с точки зрения МВС (начало)

Якобовский М.В., д.ф.-м.н.
Институт математического моделирования РАН, Москва

# ОСНОВНАЯ

Расположить в порядке неубывания *N* элементов массива чисел, используя *p* процессоров

ЦЕЛЬ

# Две задачи сортировки массива чисел

A. Объём оперативной памяти одного процессорного узла ***достаточен*** для одновременного размещения в ней всех элементов массива

B. Объём оперативной памяти одного процессорного узла ***мал*** для одновременного размещения в ней всех элементов массива

# Задача А

- Расположить $N$ элементов массива *a* таким образом, чтобы для любого

$$i = 0, \ldots, N-2$$

выполнялось неравенство

$$a_i \le a_{i+1}$$

# Задача B

- Пусть массив можно разместить на *p* процессорах.
- Пусть на процессоре с номером *rank* размещено $n^{rank}$ элементов массива $a^{rank}$.

$$N = \sum_{rank=0}^{rank<p} n^{rank}$$

- Расположить $N$ элементов массивов $a^{rank}$ таким образом, чтобы:
  - для любых $rank = 0, \ldots, (p-1)$ и $i = 0, \ldots, (n^{rank} - 2)$ выполнялось неравенство $a_i^{rank} \le a_{i+1}^{rank}$
  - для любого $rank = 0, \ldots, (p-2)$
  - выполнялось неравенство $a_{n^{rank}-1}^{rank} \le a_0^{rank+1}$

# Задача B

- Части массива хранятся на нескольких процессорах
  - Каждая часть массива должна быть упорядочена
  - На процессорах с б***о***льшими номерами должны быть размещены элементы массива с б***о***льшими значениями

- Правильно $\langle 1,2,3,5 \rangle \quad \langle 5,6,7,7 \rangle \quad \langle 8,8,9 \rangle$
- Ошибка $\langle 1,2,3,5 \rangle \quad \langle 5,7,6,7 \rangle \quad \langle 8,8,9 \rangle$
- Ошибка $\langle 1,2,3,5 \rangle \quad \langle 5,6,7,8 \rangle \quad \langle 7,8,9 \rangle$

$$N = 11$$

$$p = 3$$

# Задача B

- Будем рассматривать только процесс упорядочивания элементов:
  - Перед началом сортировки на каждом из процессоров уже есть часть элементов массива
  - После окончания сортировки на каждом из процессоров должно остаться столько элементов, сколько их было в начале (но, это уже могут быть другие элементы, расположенные ранее на других процессорах)

# Этапы сортировки

– Упорядочивание фрагментов массива на каждом из процессоров ?

– Перераспределение элементов массива между процессорами

– Упорядочивание фрагментов массива на каждом из процессоров ?

# Конструирование наилучшего последовательного алгоритма

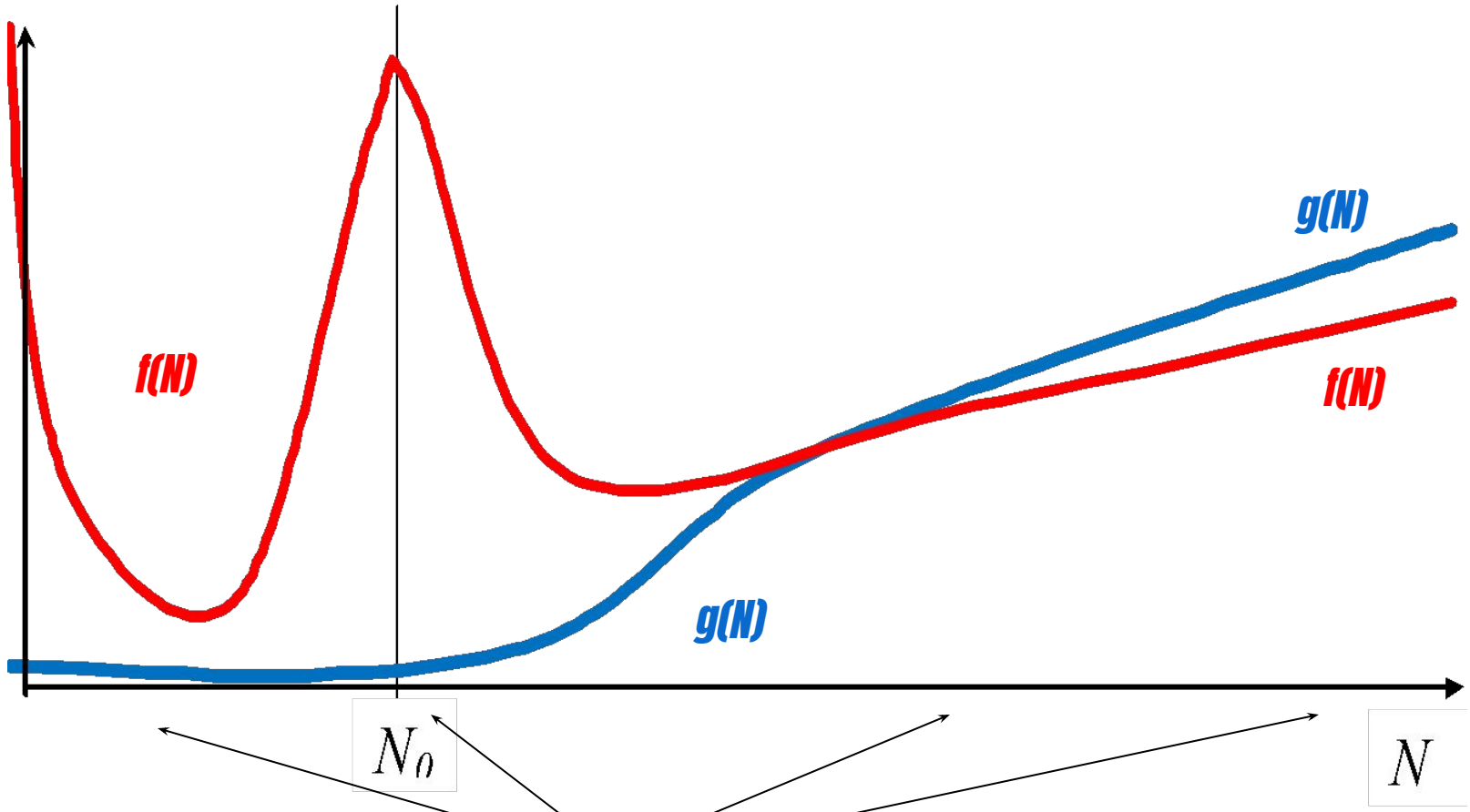
?

# Сравнение алгоритмов сортировки

$$M(n) < Cn^2$$

| Алгоритм сортировки | Среднее число операций | Максимальное число операций |
|---|---|---|
| Быстрая (***qsort***) | $11.7\, n \log_2 n$ | $O(n^2)$ |
| Пирамидальная (***hsort***) | $16\, n \log_2 n$ | $18\, n \log_2 n + 38n$ |
| Слияние списков (***lsort***) | $10\, n \log_2 n$ | $O(n \log_2 n)$ |

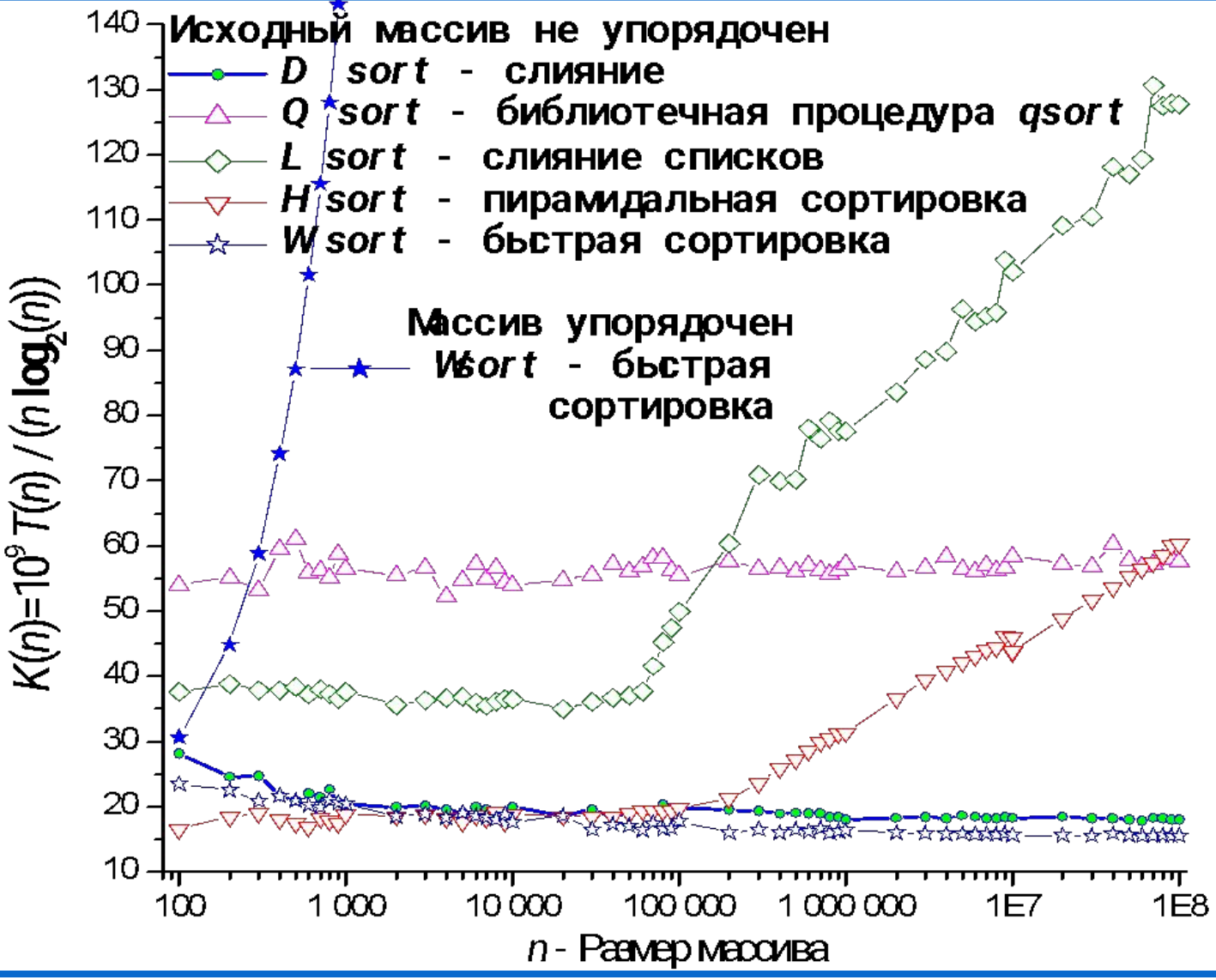
# Пусть $f(N)<C\cdot g(N)$, ну и что?



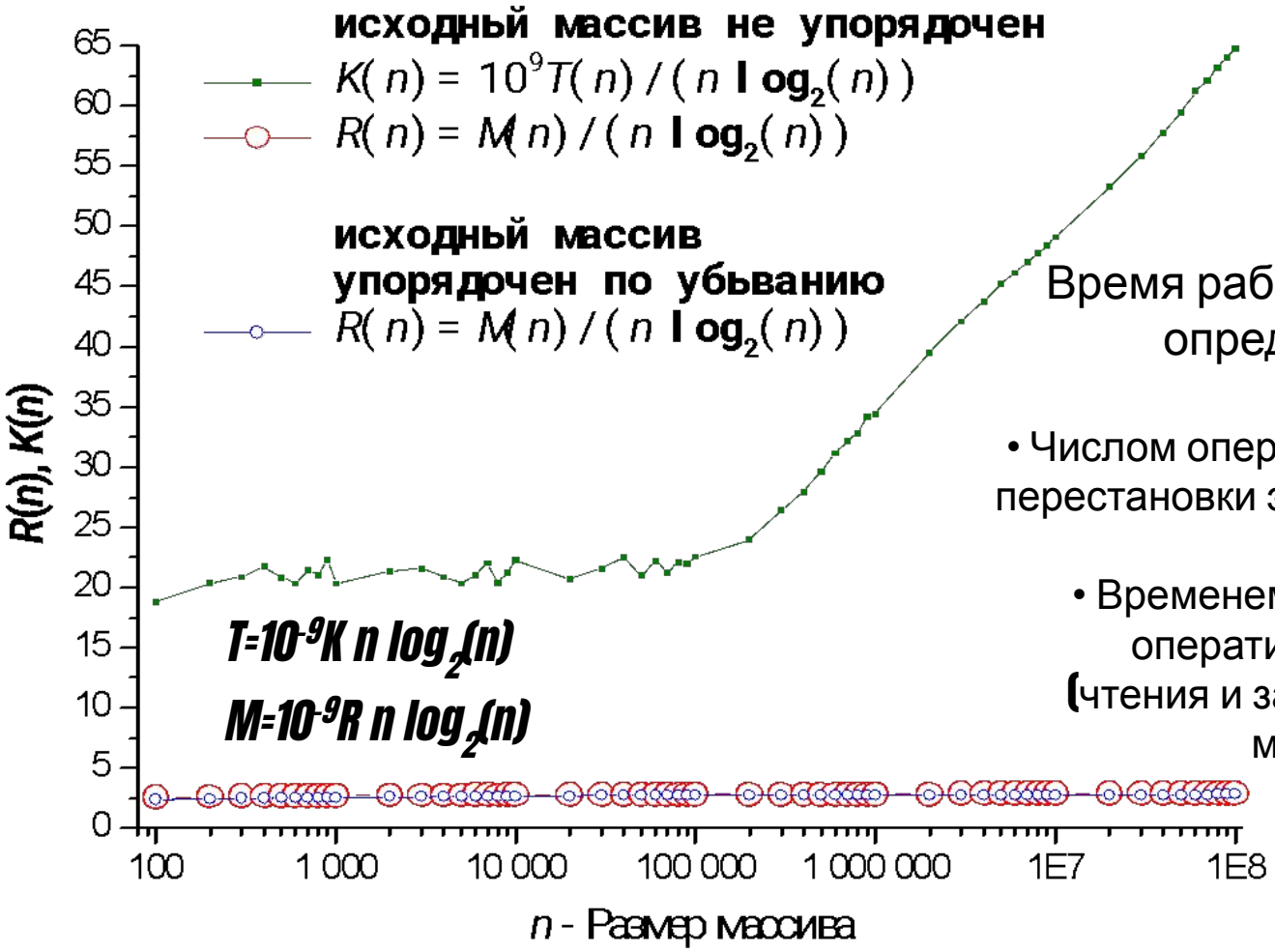

- Где тут наши 2 Гигобайта оперативной памяти???

# Константа времени сортировки

$T=10^{-9}K\ N\ \log_2(N)$



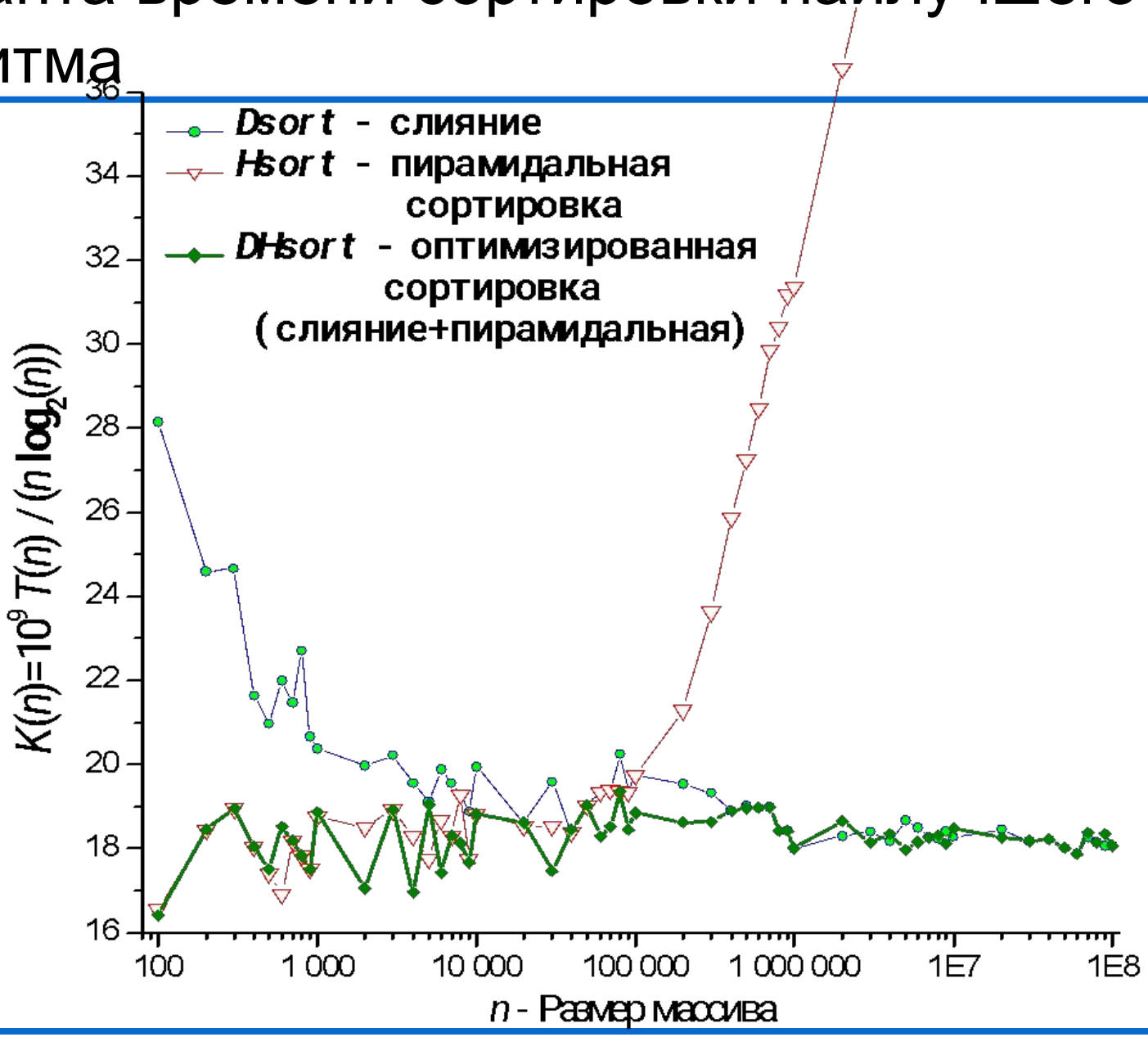
# Пирамидальная сортировка: константы времени и числа операций




Время работы алгоритма определяется:

- Числом операций сравнения и перестановки элементов массива
- Временем обращения к оперативной памяти (чтения и записи элементов массива)

# Константа времени сортировки наилучшего алгоритма



# *Изящный* алгоритм сортировки массива слиянием

```
сортировать ( массив mas, число элементов n )
  {
  если (n > 1)
    {
    // сортировка первой половины массива
    сортировать ( mas, n/2);
    // сортировка второй половины массива
    сортировать ( mas+n/2, n-n/2);
    // слияние отсортированных половинок массива
    слияние ( mas, n/2, mas+n/2,n-n/2);
    }
  }
```

# Алгоритм сортировки массива слиянием

```
Dsort(intsort *array, int n)
{
 a=array; // сортируемый массив
 b=array_second; // вспомогательный массив

 for(i=1;i<n;i=i*2) // размер объединяемых фрагментов
   {
   for(j=0;j<n;j=j+2*i) // начало первого из объединяемых
                        // фрагментов
    {
    r=j+i; // начало второго из объединяемых фрагментов
    n1=max(min(i,n-j), 0);
    n2=max(min(i,n-r), 0);

    // слияние упорядоченных фрагментов
                b = a[r…r+n1] & a[j…j+n2]
    }
   c=a;a=b;b=c;
   }
```

# Слияние упорядоченных фрагментов

```
for(ia=0,ib=0,k=0;k<n1+n2;k++)
   {
   if(ia>=n1) b[j+k]=a[r+ib++];
   else
   if(ib>=n2) b[j+k]=a[j+ia++];
   else
   if(a[j+ia]<a[r+ib]) b[j+k]=a[j+ia++];
   else                b[j+k]=a[r+ib++];
   }
```

# Сортировка слиянием методом сдваивания

- Требуется 2 + 4 + 8 + 16 тактов (8 процессоров)

Для просмотра анимации возможно требуется установить свободно распространяемый Swiff Point Player: http://www.globfx.com/products/swfpoint/

# Ускорение при методе сдваивания

- $k_1$ – сортировка, $k_2$ – передача данных

$$S(n,p)=\frac{T(n,1)}{T(n,p)}=\frac{k_1 n\log_2 n}{\frac{n}{p}\left[k_1\left(\log_2\frac{n}{p}+2p-1\right)+k_2(p-1)\right]}$$

$$S(10^9,4)\approx\frac{4}{1.13+\frac{1}{30}\frac{k_2}{k_1}}<3.5$$

$$S(10^9,32)=\frac{32}{1+\frac{1}{30}\left(56+31\frac{k_2}{k_1}\right)}\approx\frac{32}{3+\frac{k_2}{k_1}}<11$$

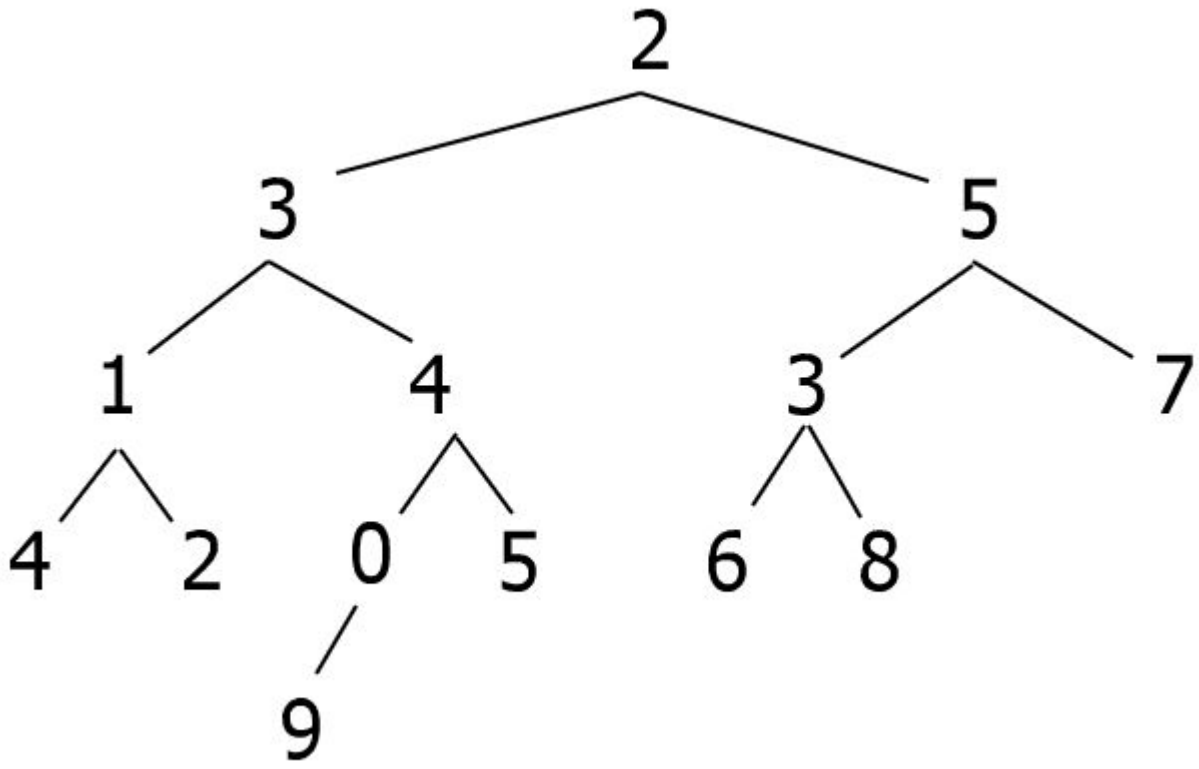
# Слияние двух массивов двумя процессорами

- Требуется 8 тактов

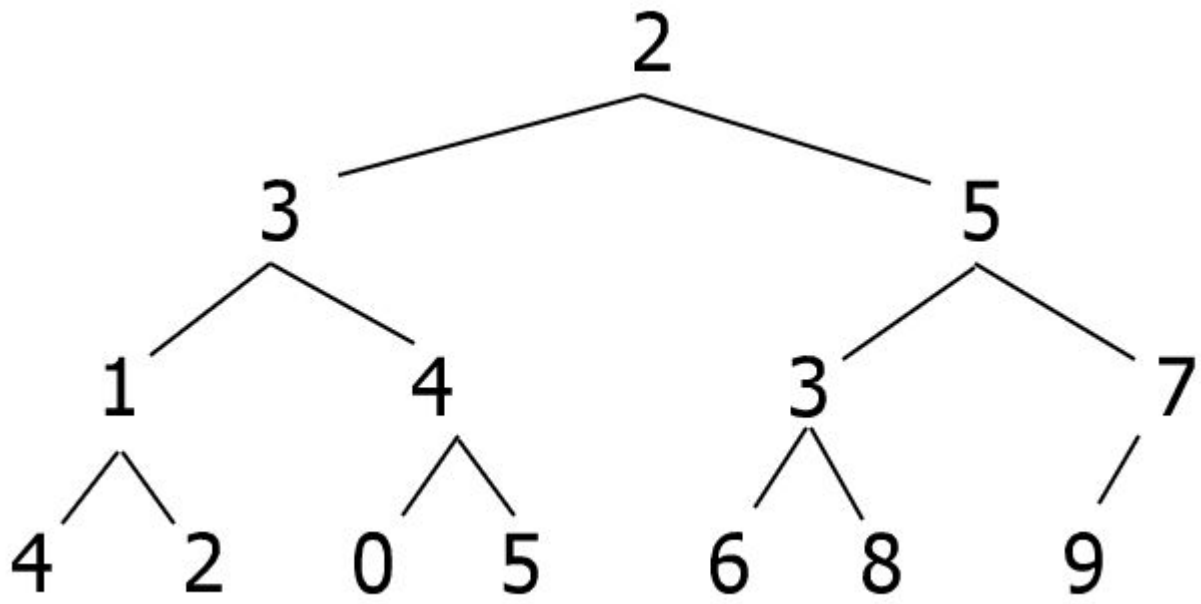
# Пирамиды

- Дерево называют сбалансированным, если потомки любого его корня отличаются по высоте не более чем на 1
- Пирамида – сбалансированное бинарное дерево в котором левый потомок любого узла не ниже правого потомка

# Не пирамида



# Пирамида



# Хранение пирамиды

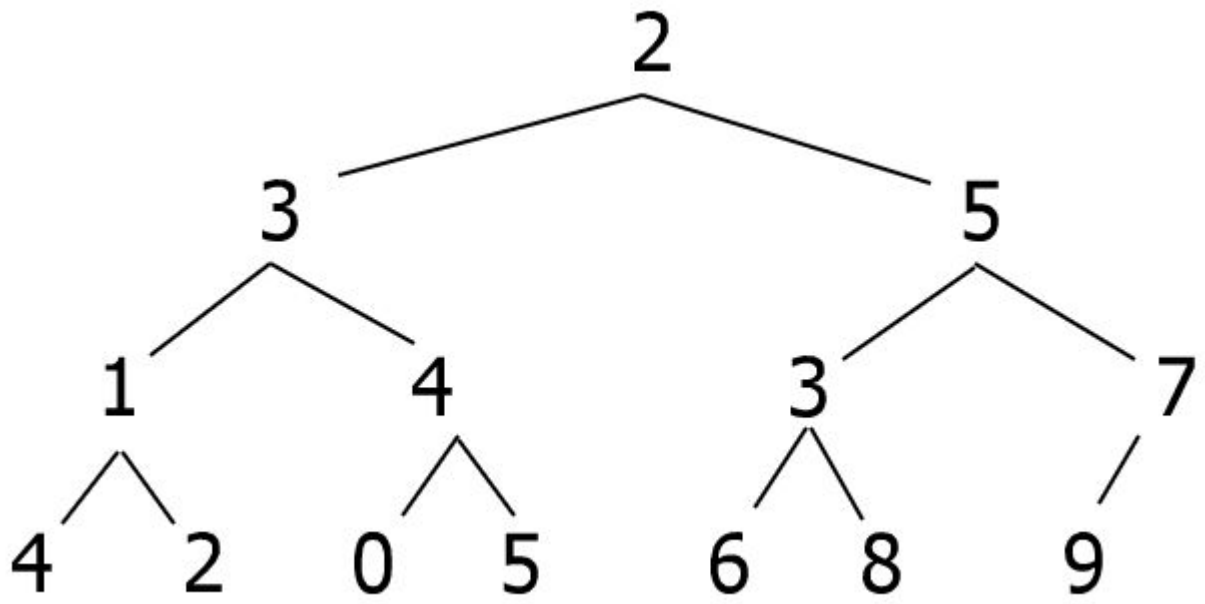
- В линейном массиве потомки вершины i хранятся в элементах 2i, 2i+1

$$a_1 \; a_2 \; a_3 \; a_4 \; a_5 \; a_6 \; a_7 \; \ldots$$

$$a_1 \; a_i \; a_3 \; a_{2i} \; a_{2i+1} \; a_6 \; a_7 \; \ldots$$

# Пирамида

- 2 3 5 1 4 3 7 4 2 0 5 6 8 9



# Пирамидальная сортировка

- Пирамида называется упорядоченной если

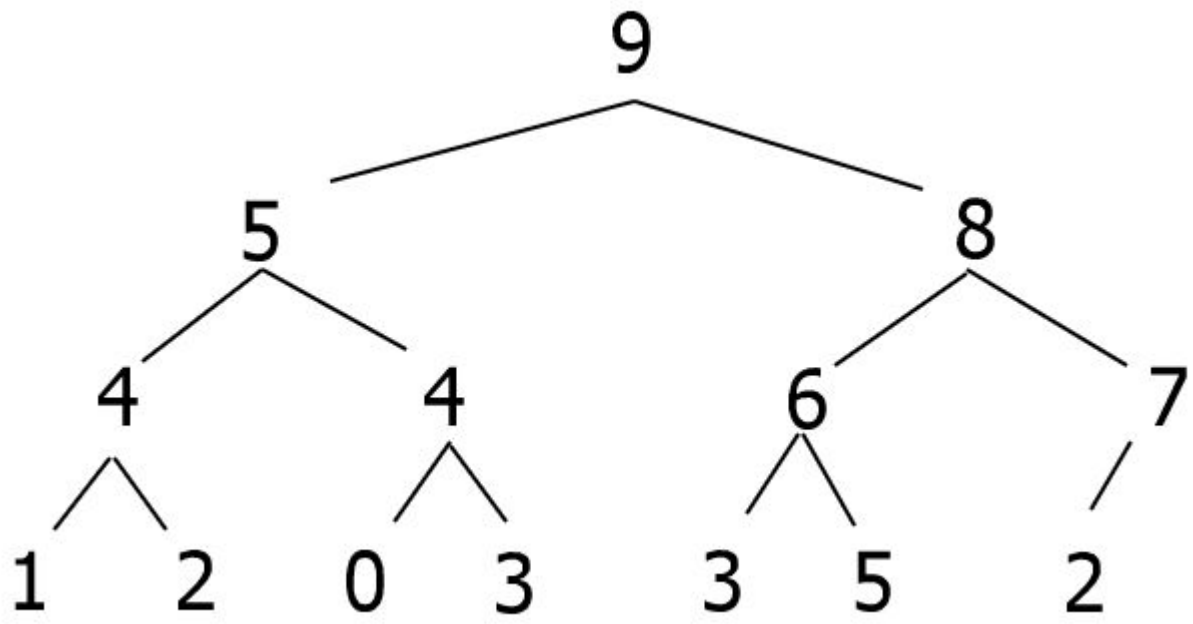
  для любого $i \geq 1 \quad a_{2i} \leq a_i \quad и \quad a_{2i+1} \leq a_i$

  (если такие элементы в массиве есть ☺)

$$a_{\lfloor i/2 \rfloor} \geq a_i, \quad при \quad 1 \leq \lfloor i/2 \rfloor < i \leq N$$

# Упорядоченная пирамида

- 9 5 8 4 4 6 7 1 2 0 3 3 5 2



# Пирамидальная сортировка

- 9 58 4467 1203352[
- 2 58 4467 120335[9
- 8 52 4467 120335[9
- 8 57 4462 120335[9

- 5 57 4462 12033[89
- 7 55 4462 12033[89
- 7 56 4452 12033[89

- 3 56 4452 1203[789
- 6 53 4452 1203[789
- 6 53 4452 1203[789
- 6 55 4432 1203[789

- 3 55 4432 120[6789
- 5 35 4432 120[6789
- 5 45 3432 120[6789

- 0 45 3432 12[56789
- 5 40 3432 12[56789
- 5 43 3402 12[56789

- 2 43 3402 1[556789
- 4 23 3402 1[556789
- 4 43 3202 1[556789

- 1 23 3402 [4556789

- 1 23 3402 [4556789
- 3 21 3402 [4556789
- 3 22 3401 [4556789

- 1 22 340 [34556789

# Пирамидальная сортировка

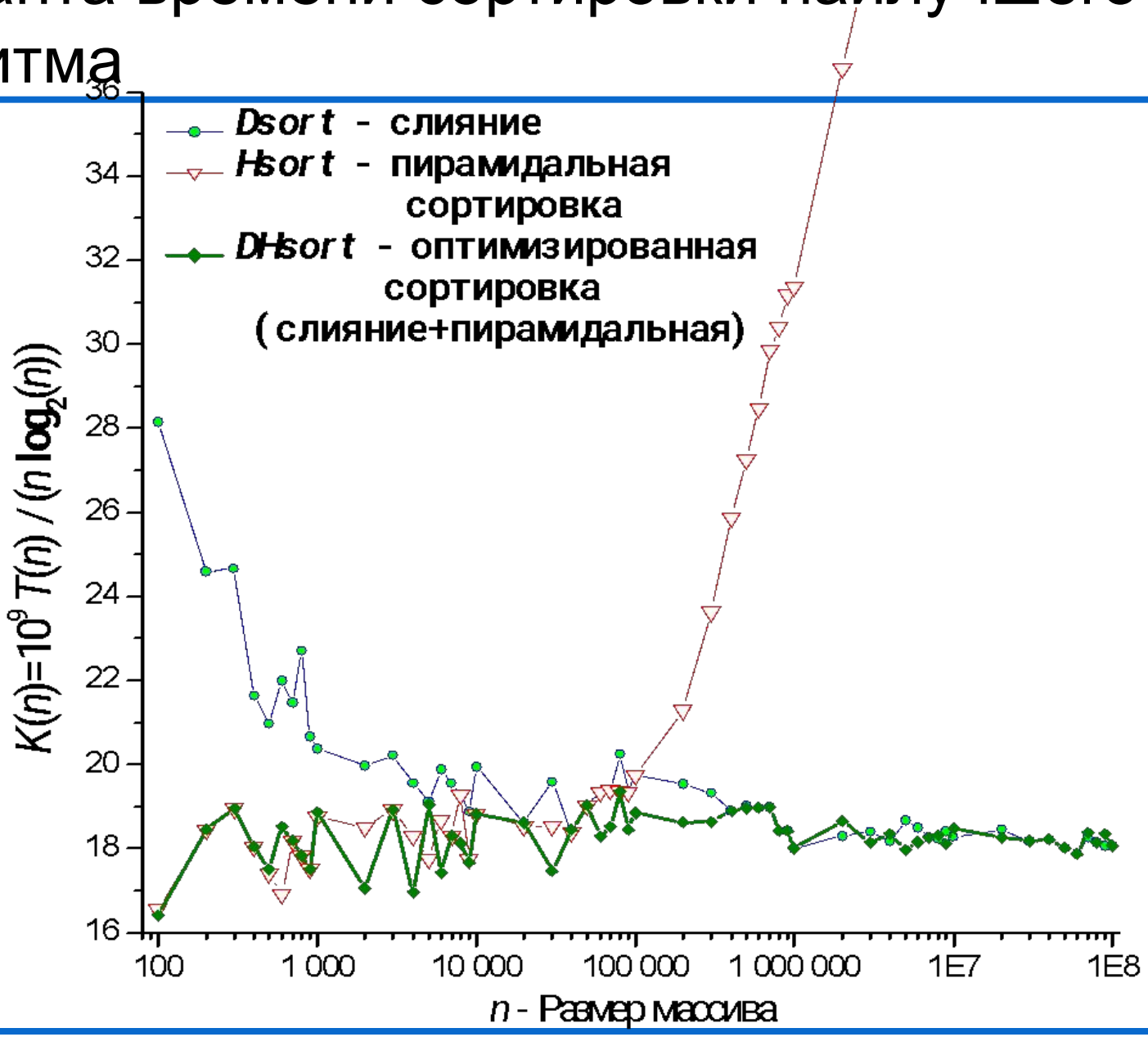
- 9 5 8 4 4 6 7 1 2 0 3 3 5 2 [
- 2 5 8 4 4 6 7 1 2 0 3 3 5 [ 9
- 8 5 2 4 4 6 7 1 2 0 3 3 5 [ 9
- 8 5 7 4 4 6 2 1 2 0 3 3 5 [ 9

- 5 5 7 4 4 6 2 1 2 0 3 3 [ 8 9
- 7 5 5 4 4 6 2 1 2 0 3 3 [ 8 9
- 7 5 6 4 4 5 2 1 2 0 3 3 [ 8 9

# Оптимальный алгоритм

- Оптимальна комбинация
- H алгоритма (пирамидальная ) в диапазоне
  - 10 - 50 000
- D алгоритма (слияние) в диапазоне
  - 50 000 - 100 000 000

# Константа времени сортировки наилучшего алгоритма



# Заключение

- Рассмотрен ряд методов сортировки массивов
- Проиллюстрирована разница между зависимостью от объема данных времени сортировки и числа выполняемых операций
- Построен «наилучший» последовательный алгоритм сортировки

# Вопросы для обсуждения

- В чем причина различия характера зависимости времени сортировки и числа выполняемых операций от числа элементов сортируемого массива?
- Какие еще можно предложить варианты сортировки, улучшающие использование кеш-памяти?

# Контакты

**Якобовский М.В.**

д.ф.-м.н.,

зав. сектором

«Программного обеспечения многопроцессорных систем и вычислительных сетей»

Института математического моделирования

Российской академии наук

mail: mail: lira@imamod.rumail: lira@imamod.ru

web: web: http://lira.imamod.ru