# МГУ ВМК

## Лекция 6

# *Сортировка данных с точки зрения МВС*

***Параллельные алгоритмы***

Якобовский Михаил Владимирович
проф., д.ф.-м.н.
Институт прикладной математики
им. М.В.Келдыша РАН, Москва

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

# К вопросу о

- Наилучшем последовательном алгоритме
- Медленном последовательном алгоритме
- Высокой степени внутреннего параллелизма

# 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 A

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

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

выполнялось неравенство

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

# Задача B

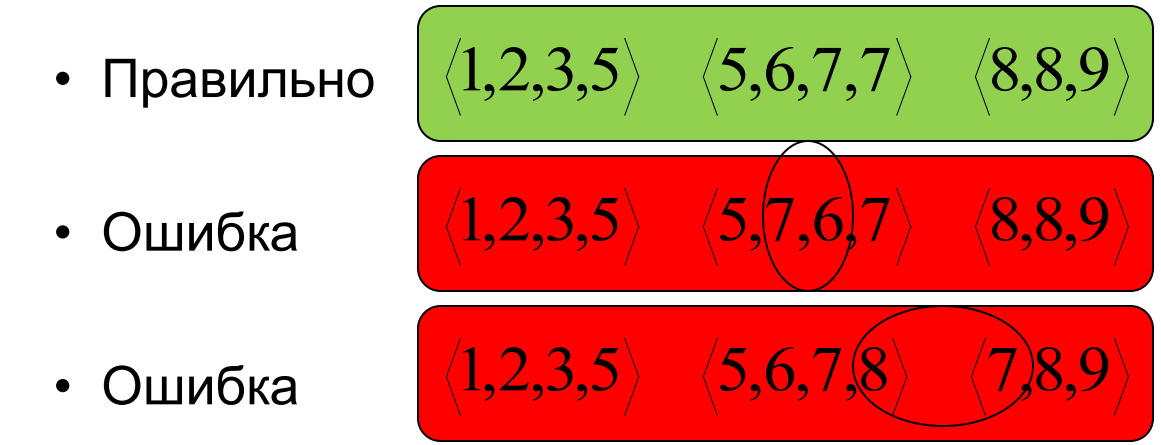
- Пусть массив можно разместить на *p* процессорах.
- Пусть на процессоре с номером *rank* размещено $n^{rank}$ элементов массива $a^{rank}$.

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

- 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




$N = 11$

$p = 3$

# 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)

# Предлагаемая стратегия: Этапы сортировки

- Упорядочивание фрагментов массива на каждом из процессоров
- Перераспределение элементов массива между процессорами

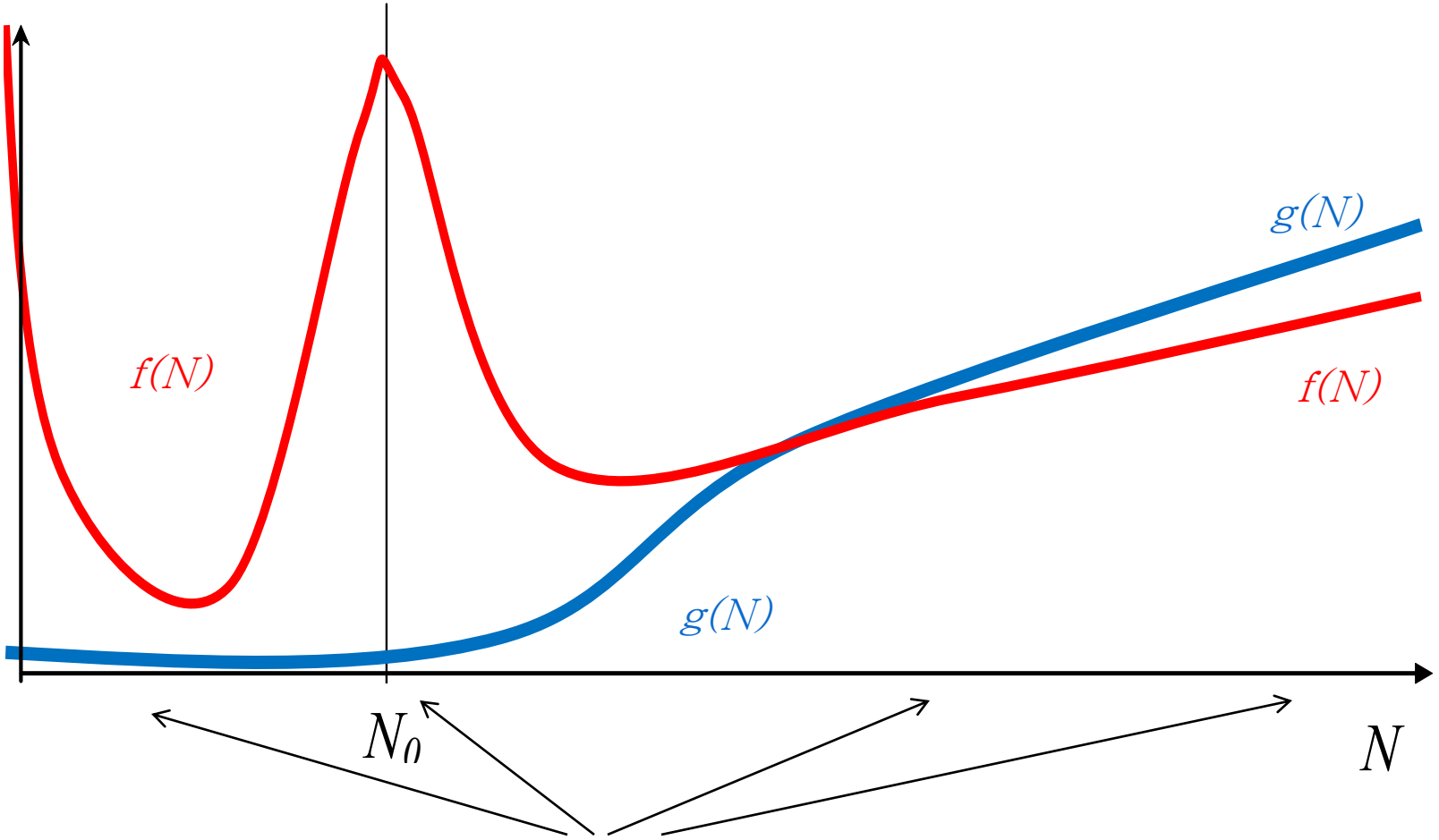
# Конструирование наилучшего последовательного алгоритма

*Сравнение последовательных 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***) | $10\ \ n \log_2 n$ | $O(n \log_2 n)$ |

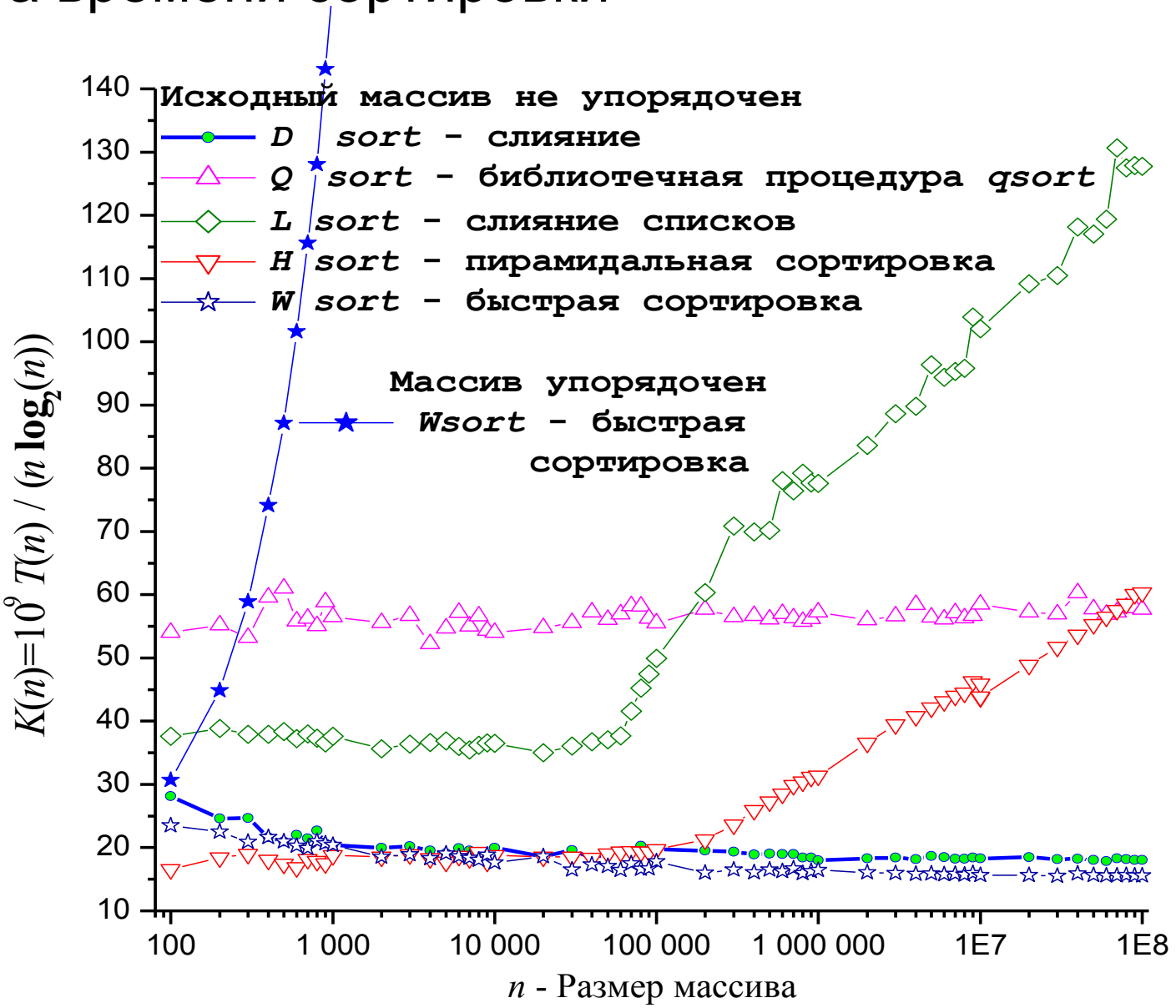
# Пусть $f(N)<C\cdot g(N)$, ну и что?



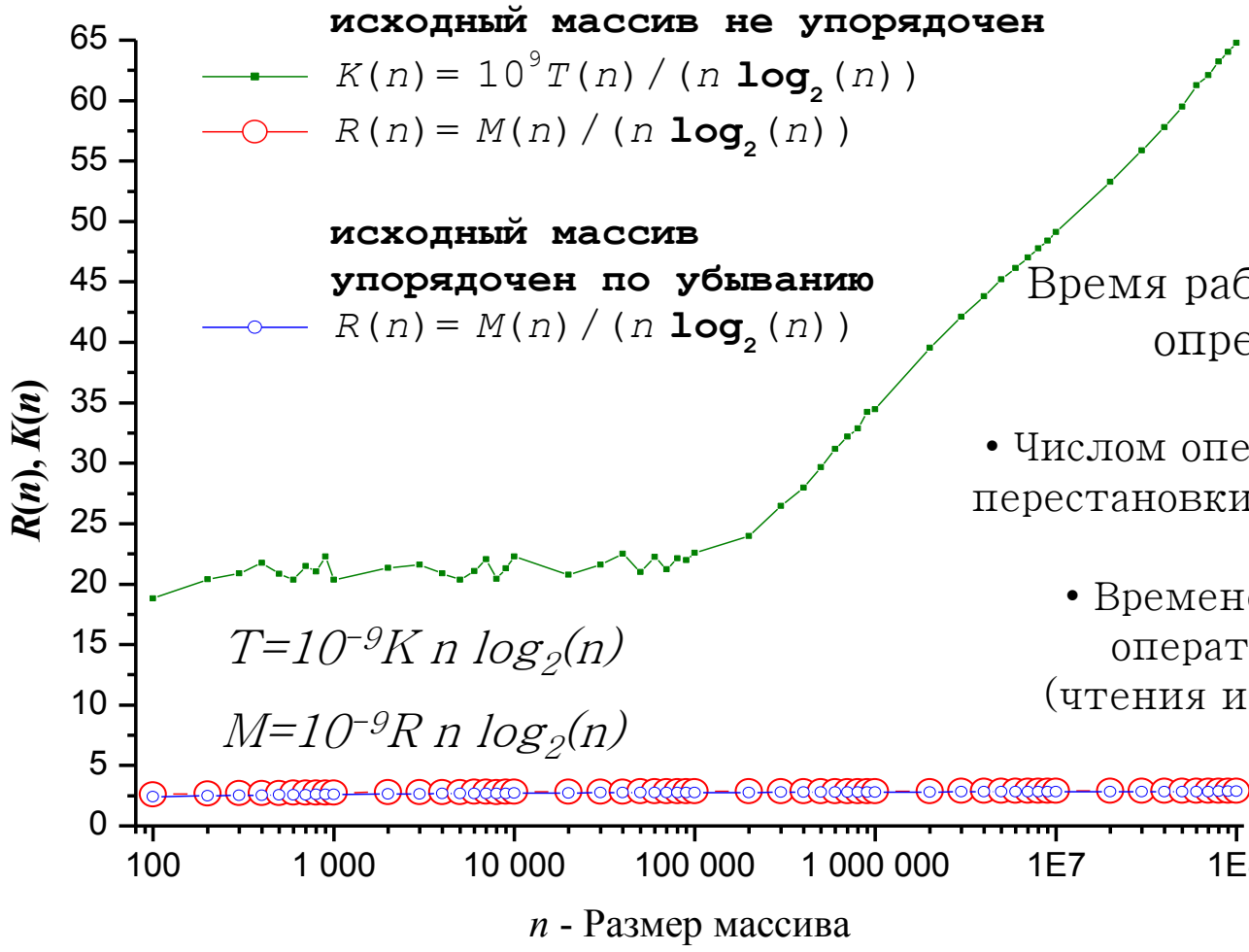

- Где тут наши 2 Гигобайта оперативной памяти???

# Константа времени сортировки

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

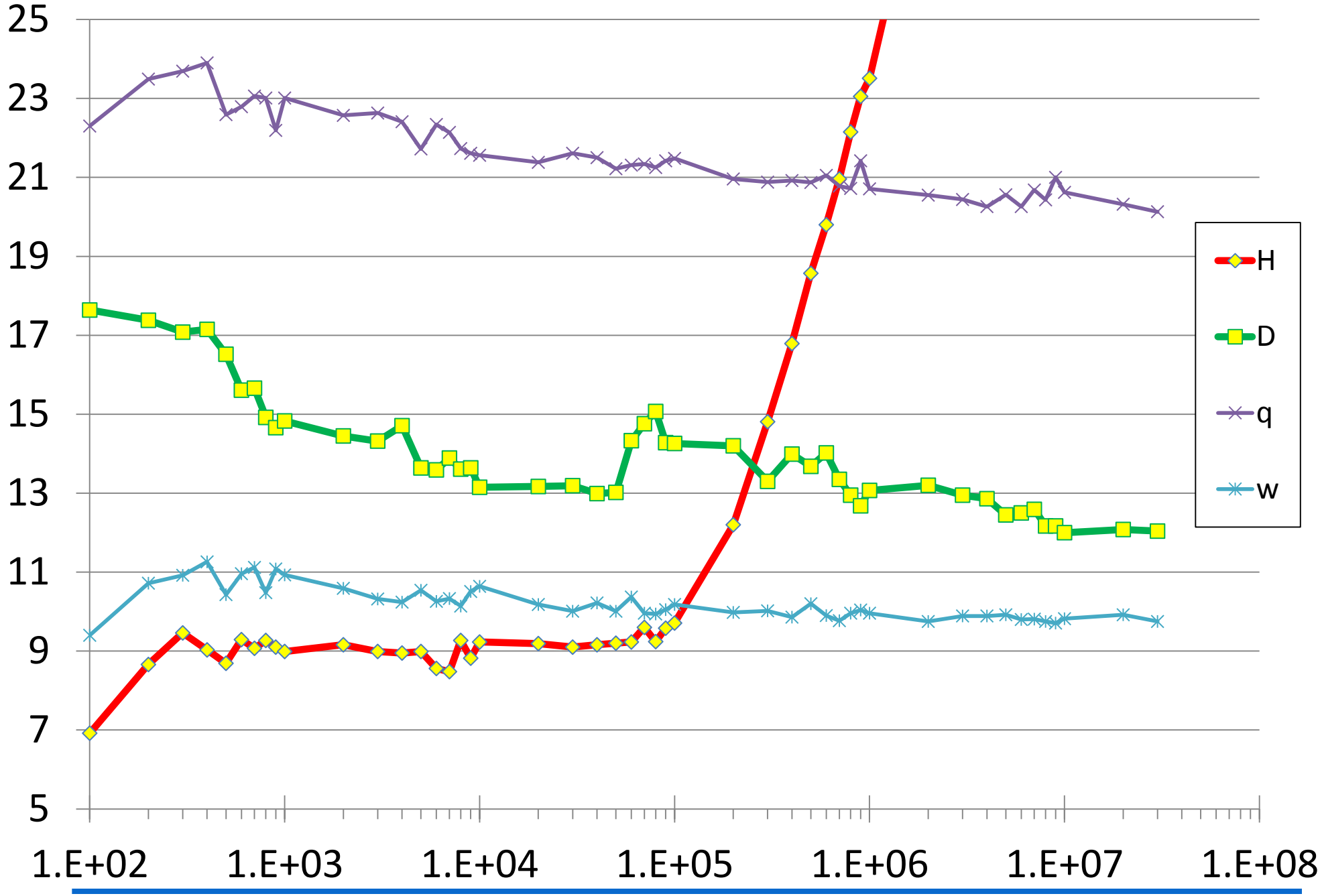


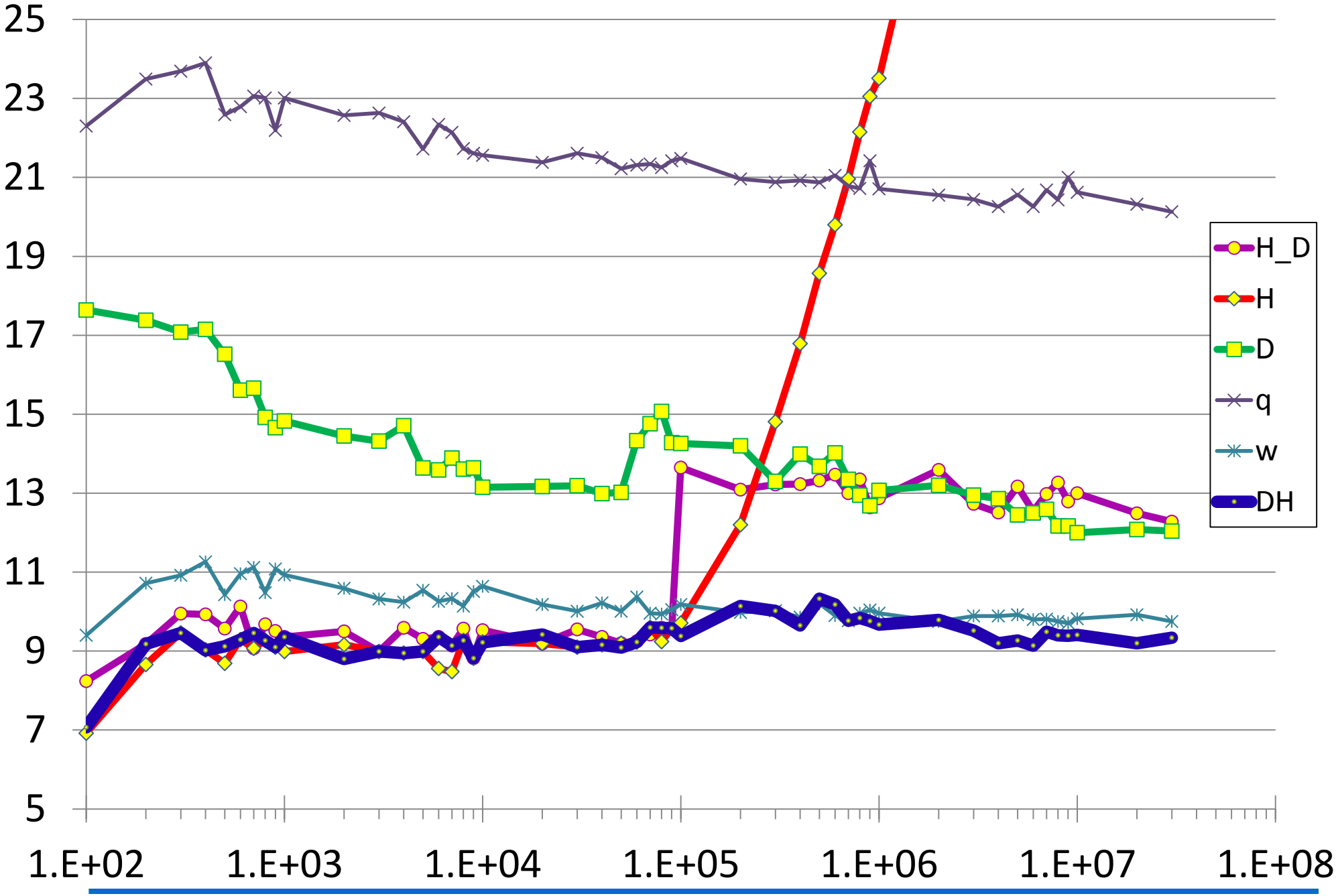
# Пирамидальная сортировка: константы времени и числа операций



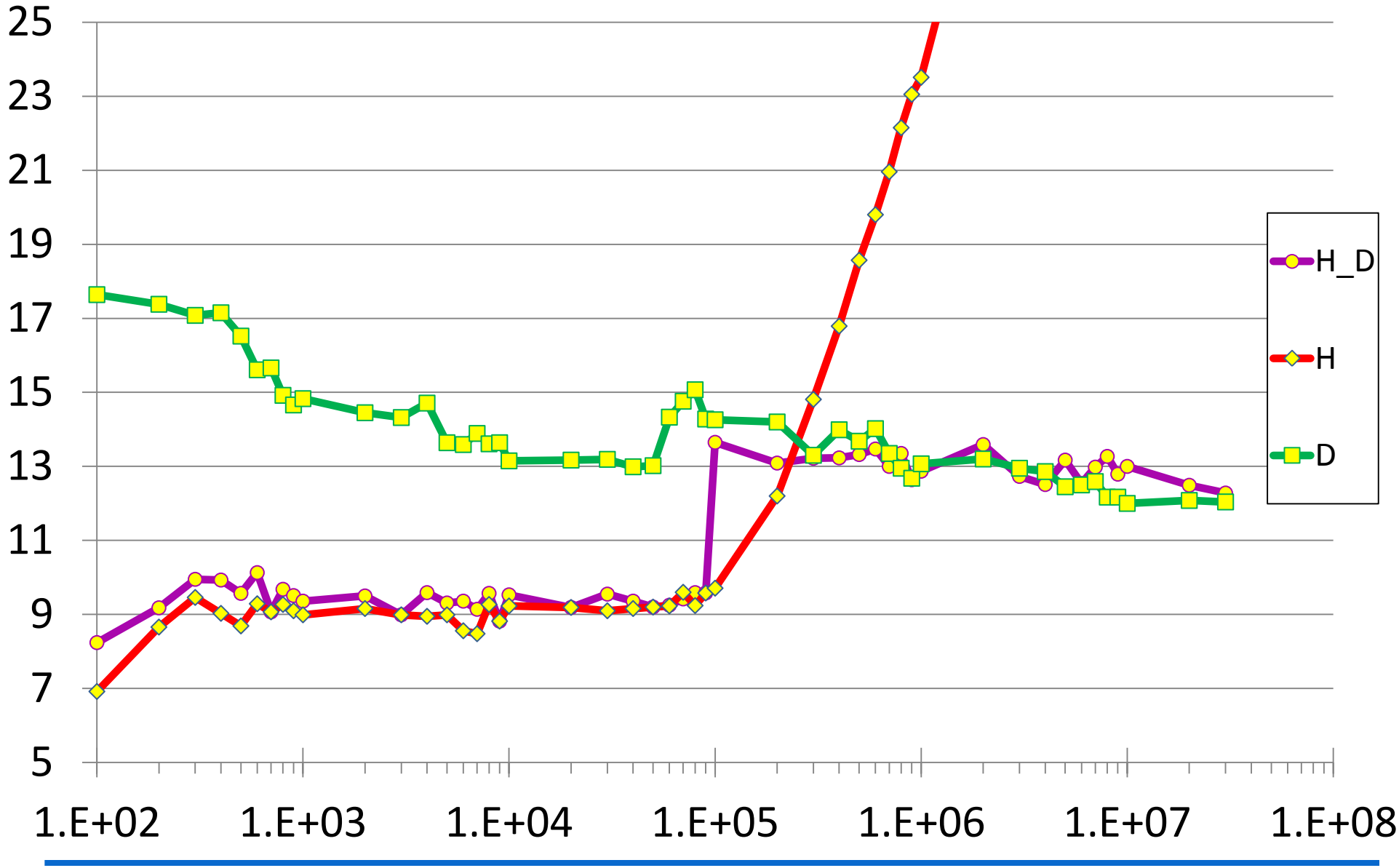

Время работы алгоритма определяется:

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

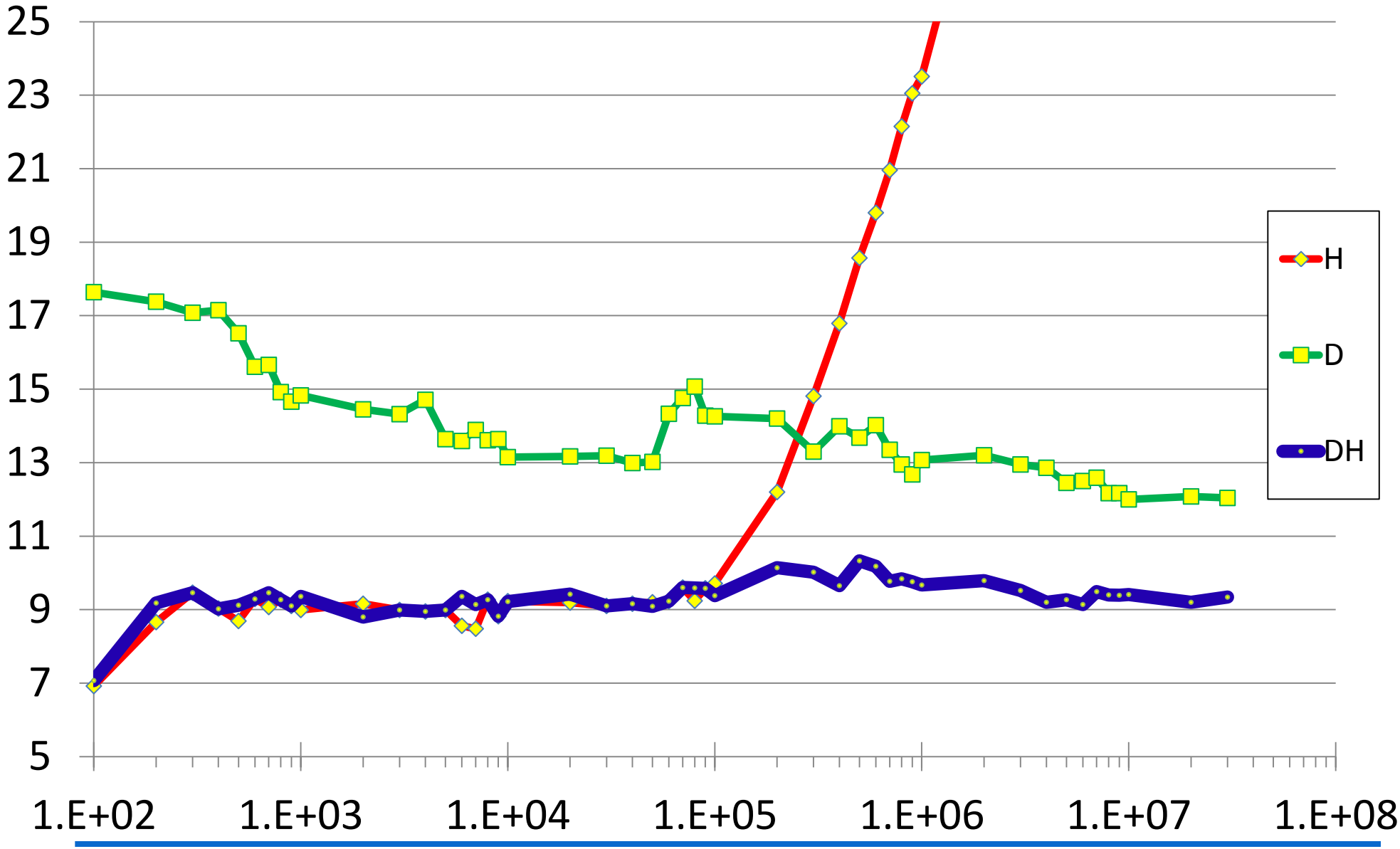

25
23
21
19
17
15
13
11
9
7
5
1.E+02
1.E+03
1.E+04
1.E+05
1.E+06
1.E+07
1.E+08
H
D
q
w

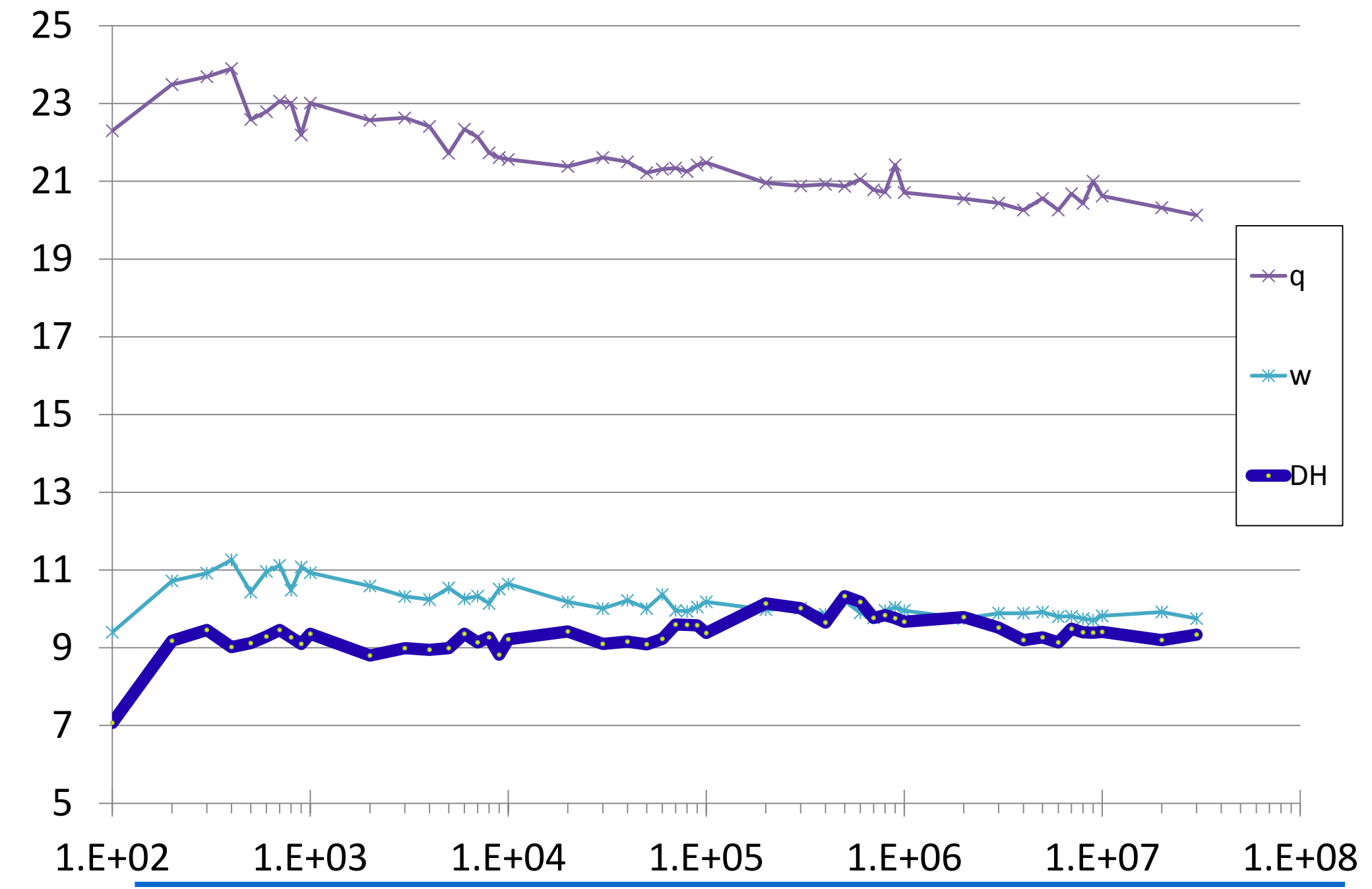

25
23
21
19
17
15
13
11
9
7
5
1.E+02
1.E+03
1.E+04
1.E+05
1.E+06
1.E+07
1.E+08
H_D
H
D
q
w
DH

# Меньше 10^5 - пирамидальная, больше - слияние

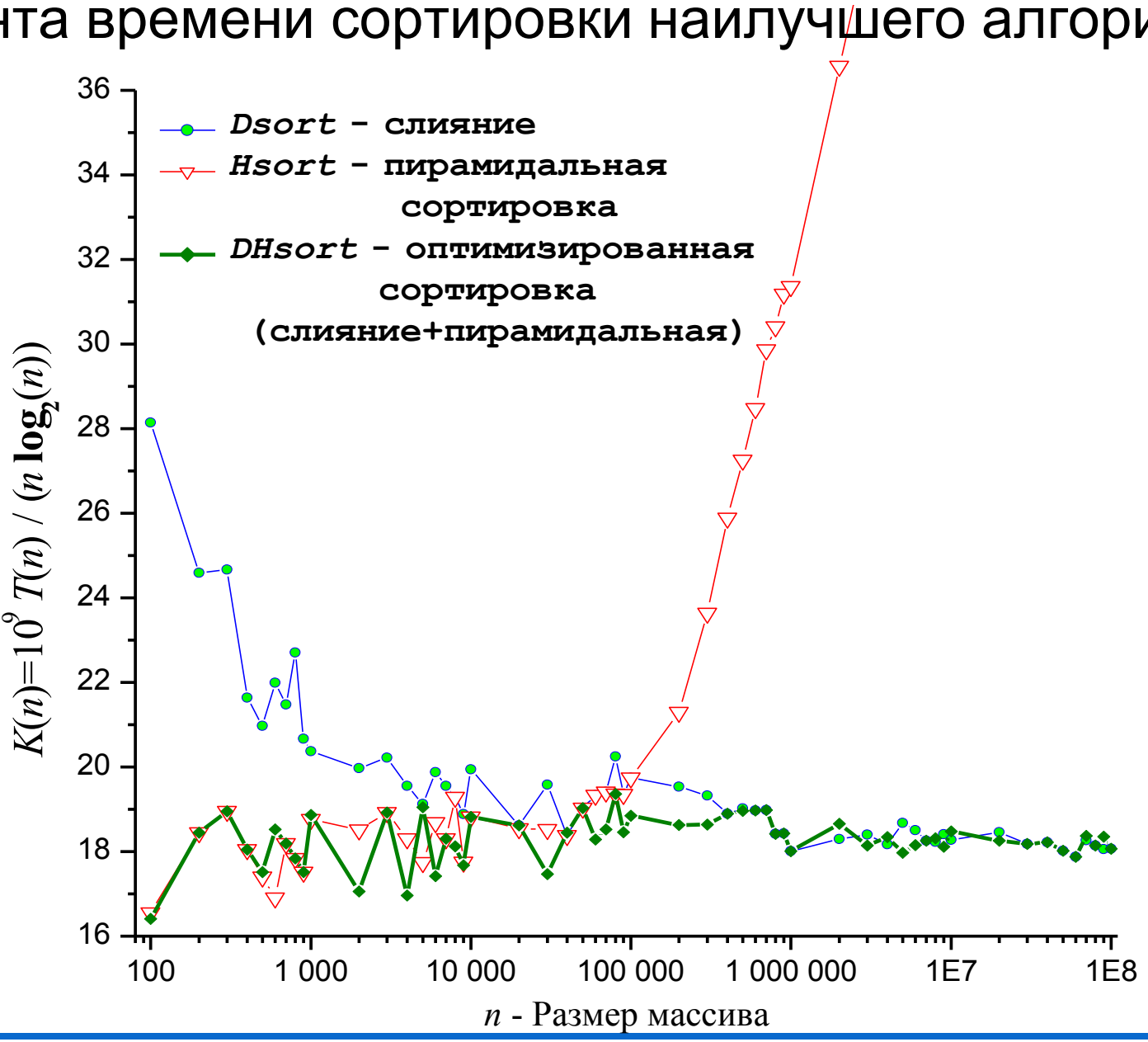


Меньше 100 000 элементов – пирамидальная,
иначе – (пирамидальная над фрагментами + слияние упорядоченных фрагментов)

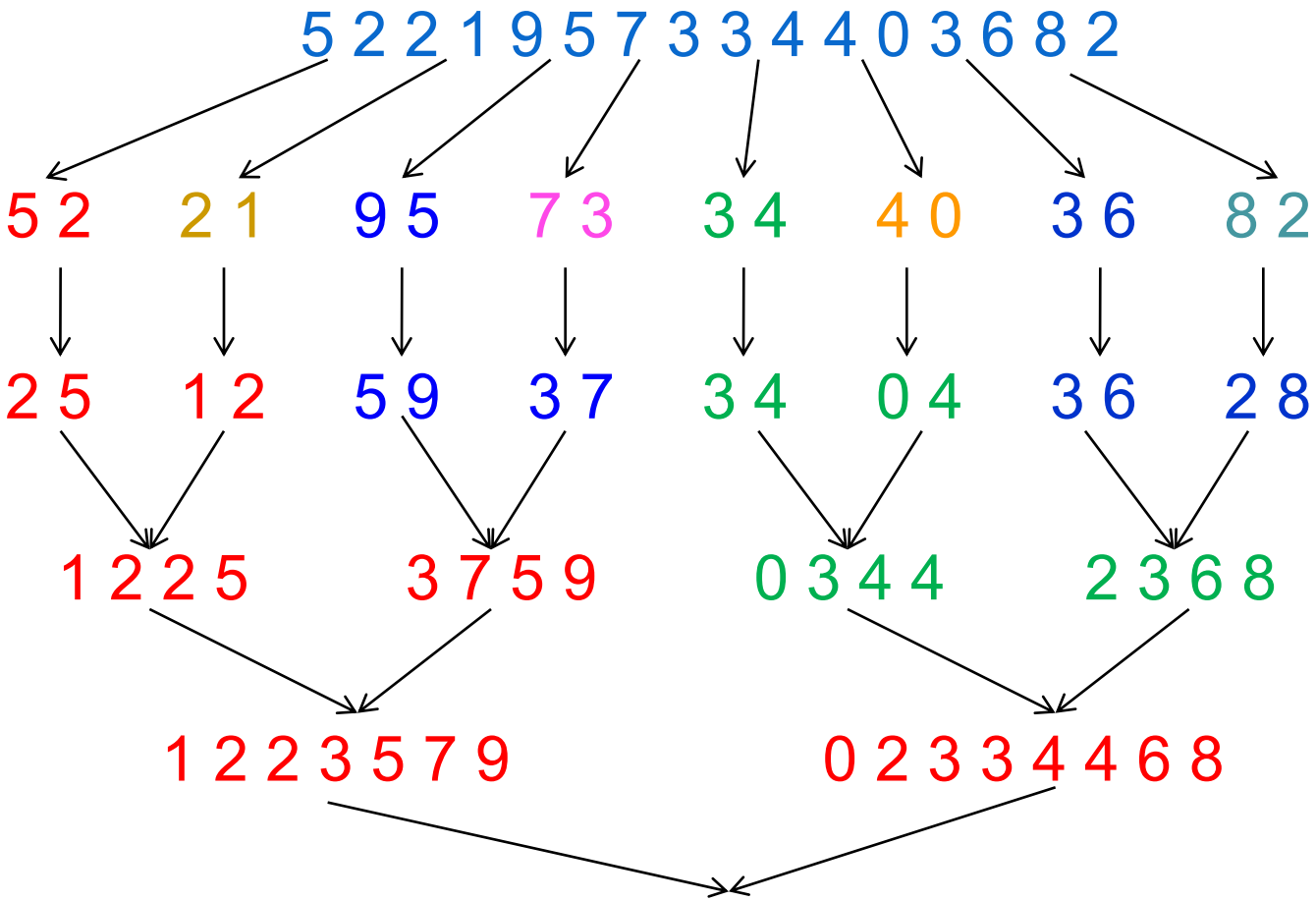



25
23
21
19
17
15
13
11
9
7
5
1.E+02
1.E+03
1.E+04
1.E+05
1.E+06
1.E+07
1.E+08
q
w
DH

# Константа времени сортировки наилучшего алгоритма



# Сортировка слиянием методом сдваивания
# Требуется 2 + 4 + 8 + 16 = 30 тактов (8 процессоров)




исходный массив

8 процессоров

2 такта

4 такта

8 тактов

16 тактов

ИТОГО 30 тактов

# *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);
    }
  }
```

# Нерекурсивный 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++];
      }
```

# Слияние одним процессором. Требуется 16 тактов

*Обращение к **последовательным** адресам памяти*

(1 2 2 3 5 5 7 9) (0 2 3 3 4 4 6 8)

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

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

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

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

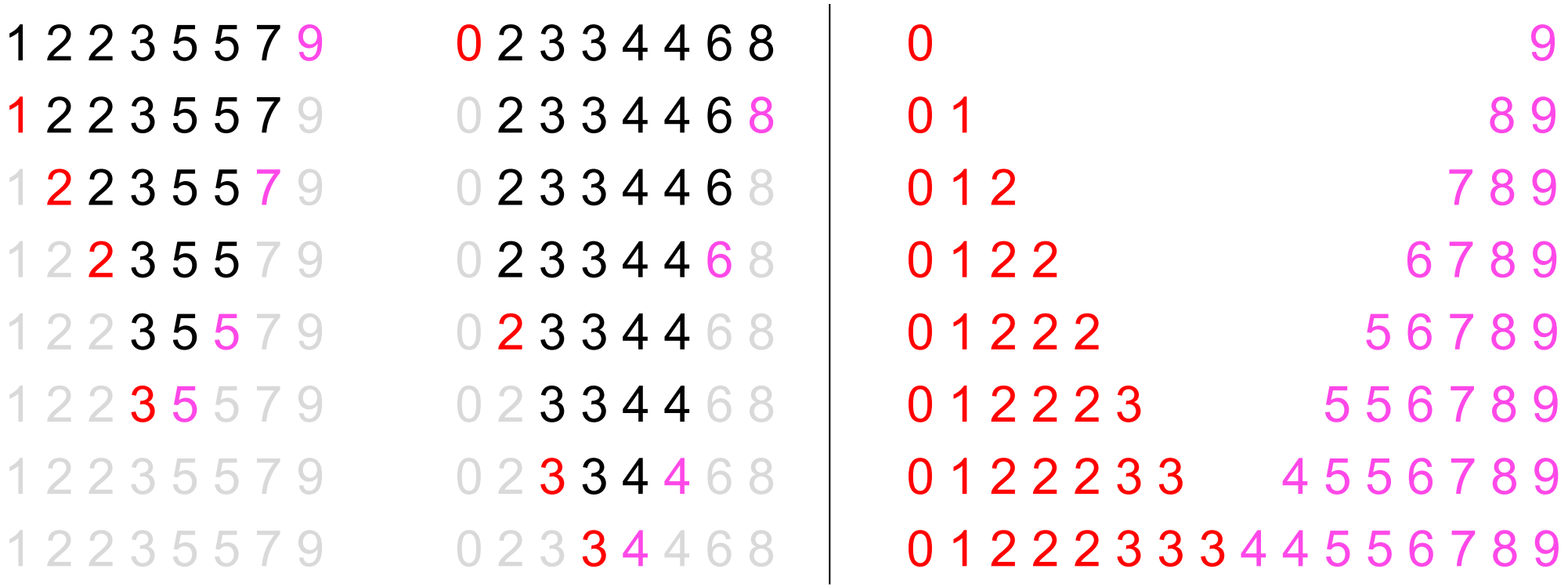
1 2 2 3 5 5 7 9 0 2 3 3 4 4 6 8 0 1 2 2 3

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

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

…

# Слияние двумя процессорами. Требуется 8 тактов



# Ускорение при методе сдваивания

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

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

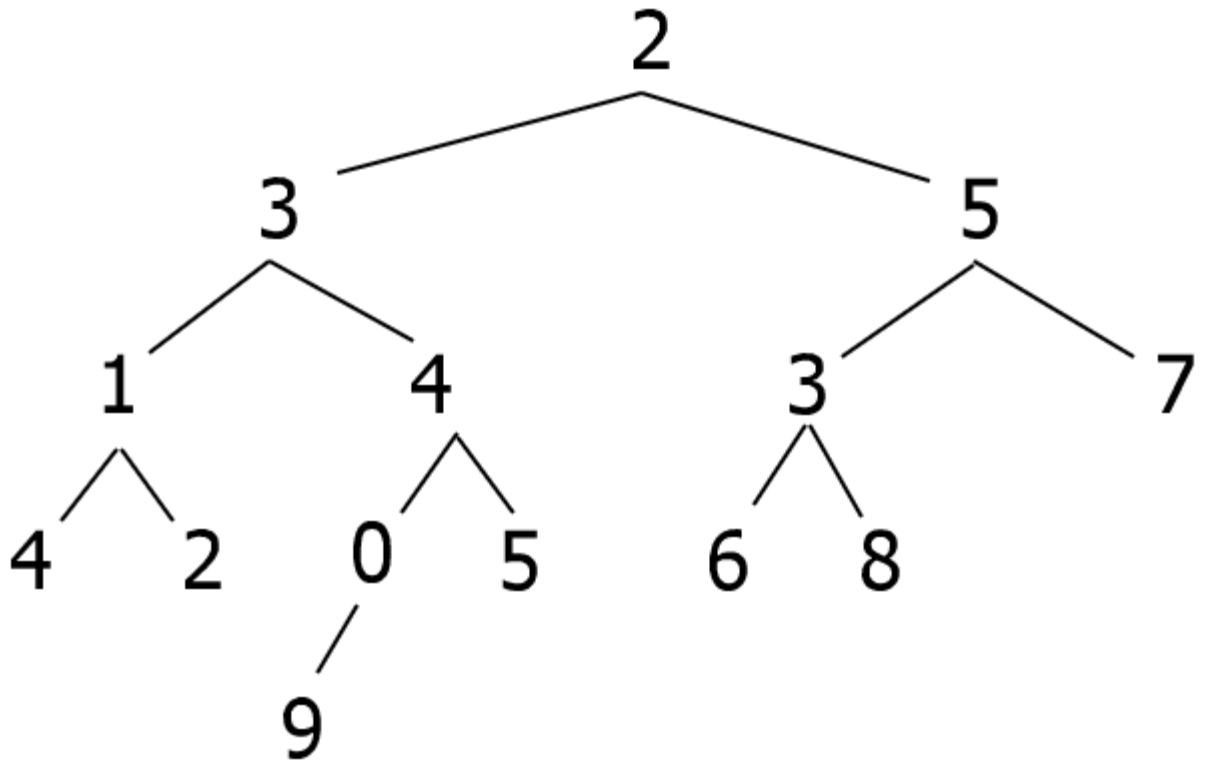
$$S\left(10^9,4\right)\approx\frac{4}{1.13+\frac{1}{30}\frac{k_2}{k_1}}<3.5$$

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

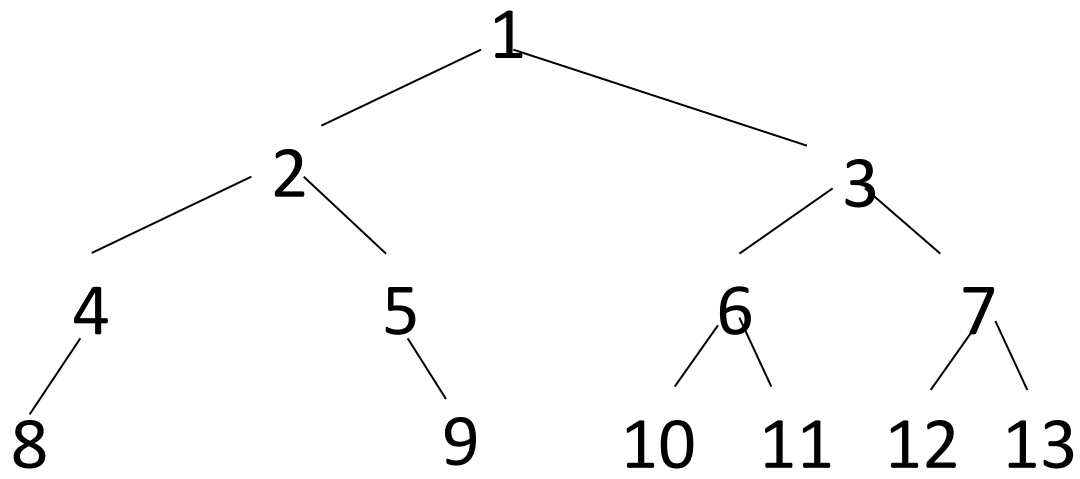
# Пирамиды

- 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

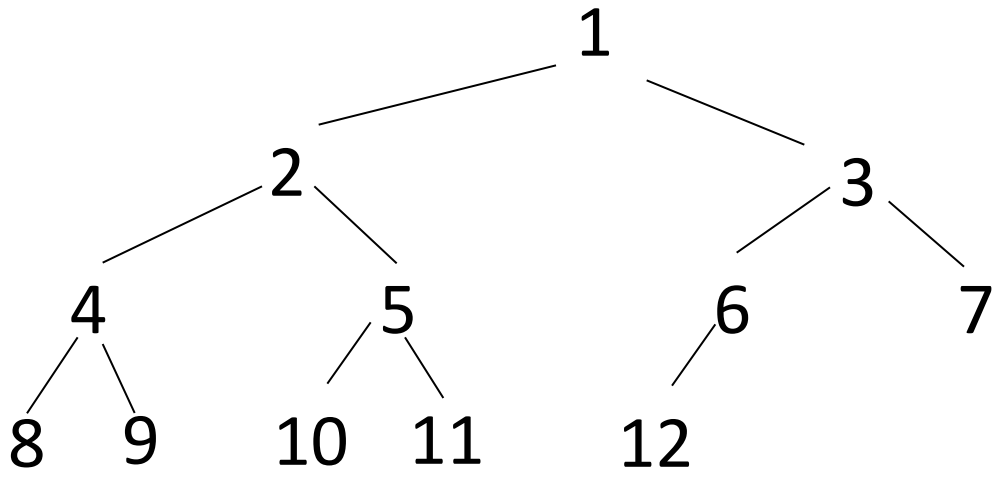
# Не сбалансированное дерево

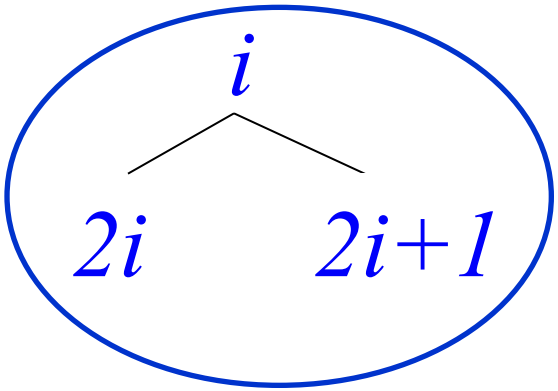


# Сбалансированное дерево, но не пирамида



# Пирамида



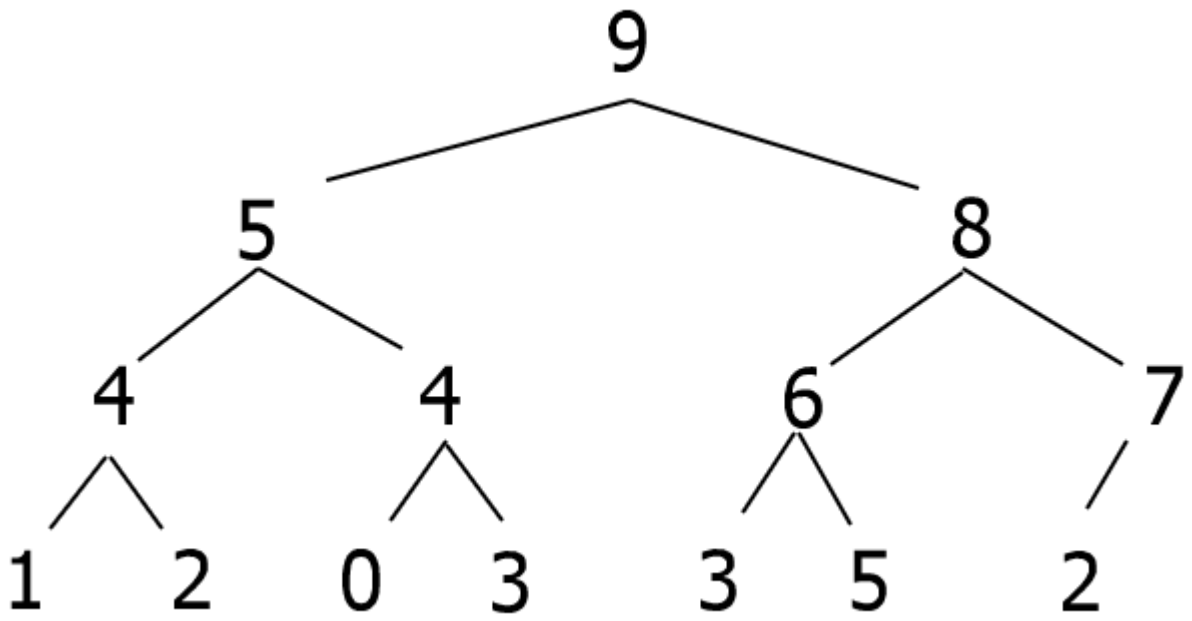

*Потомки вершины i хранятся в элементах 2i, 2i+1*

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

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

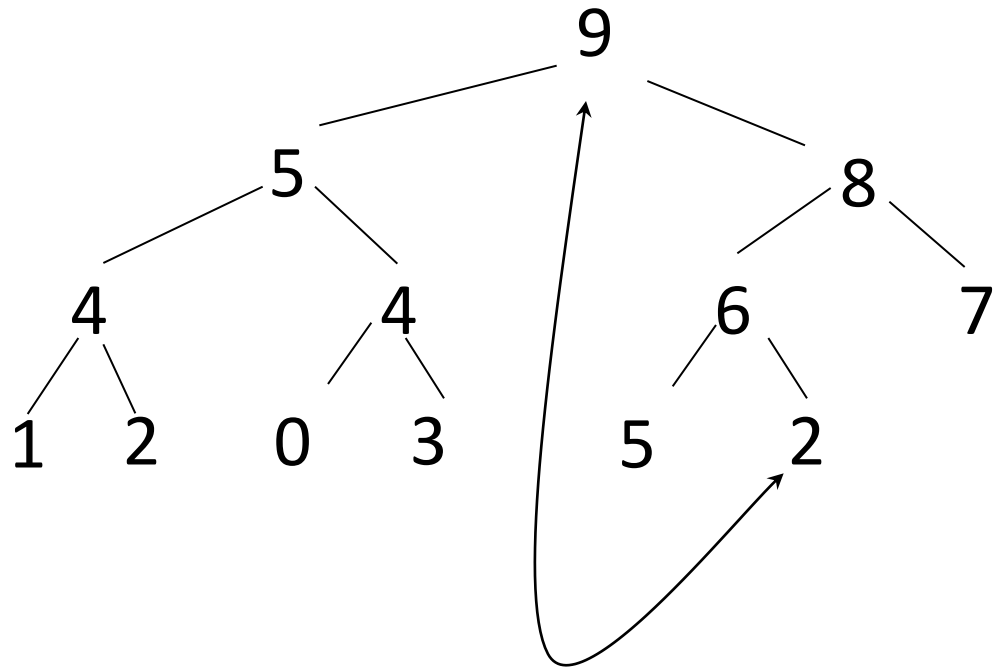
# Упорядоченная пирамида

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



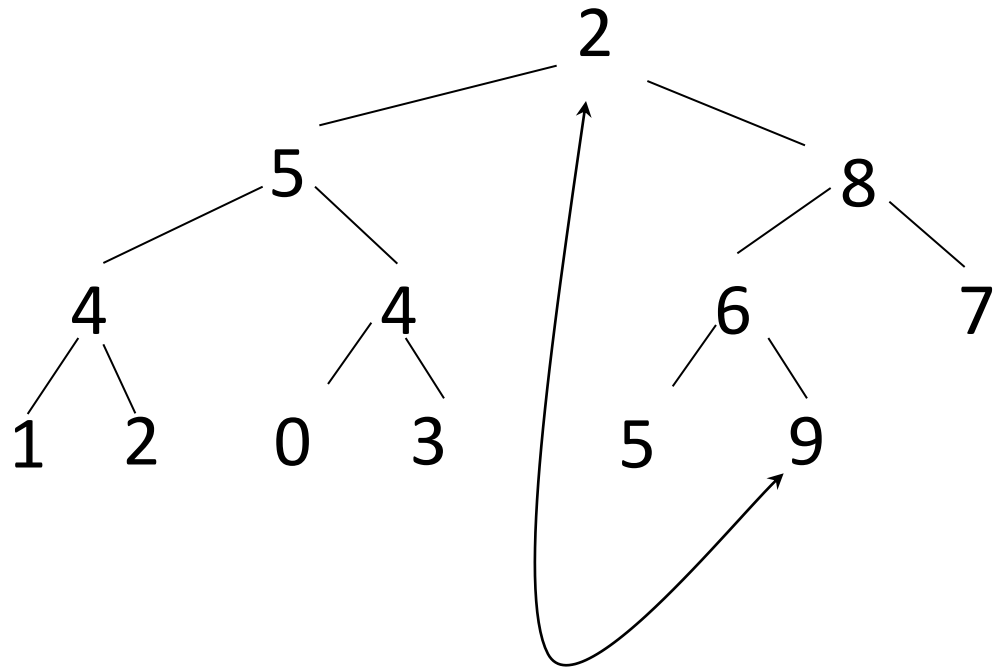
# Поменять корень местами с последним потомком

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



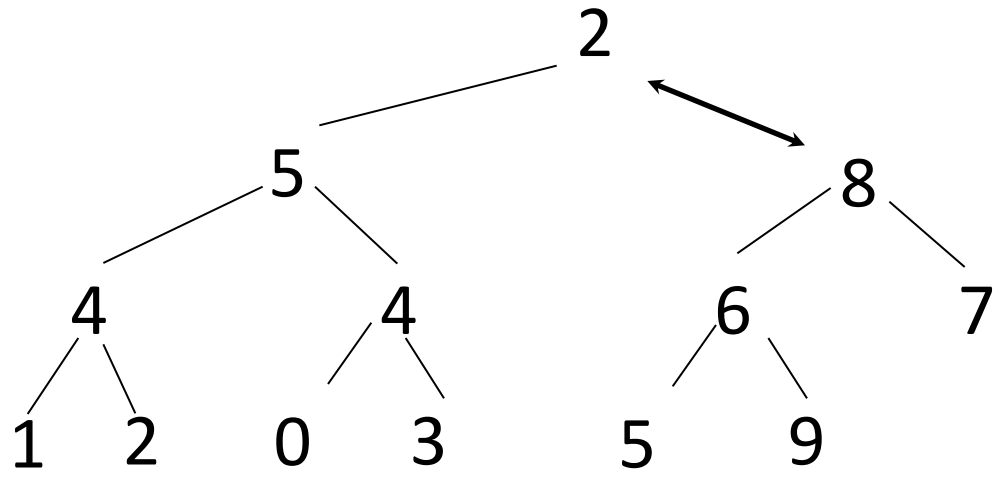
# Поменять корень местами с последним потомком

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



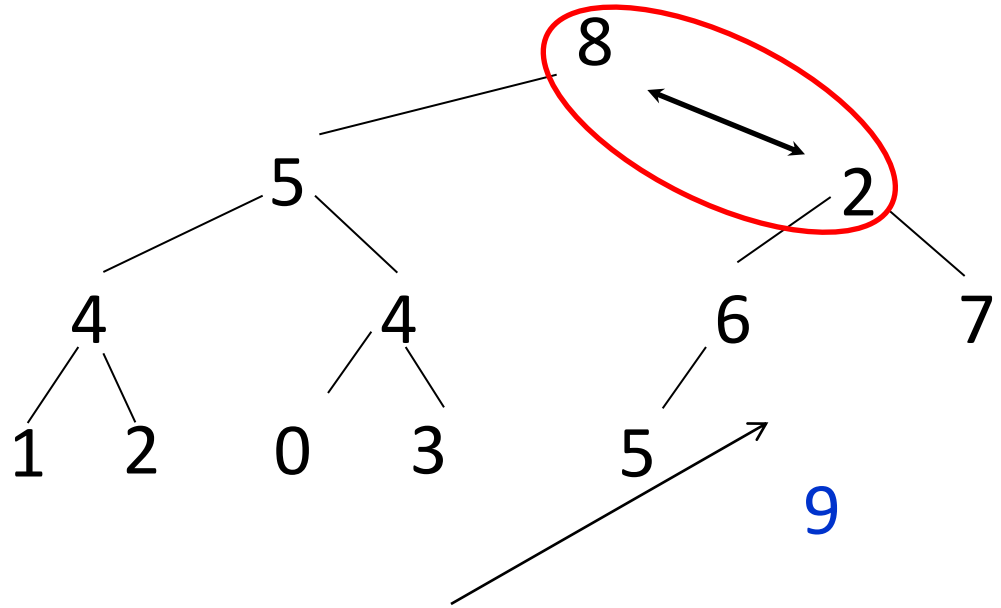
# Вернуть остатку пирамиды упорядоченность

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



# Вернуть остатку пирамиды упорядоченность

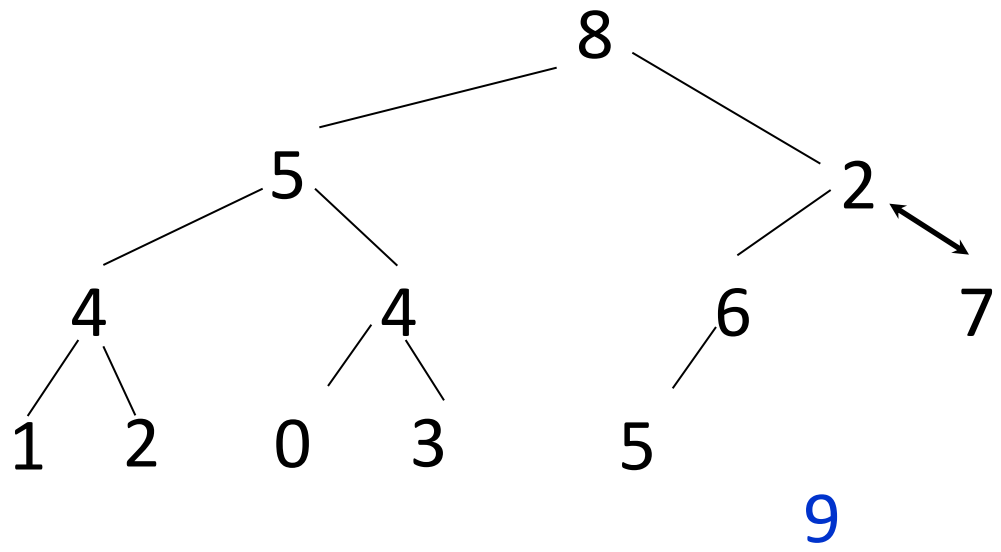
- 8 5 2 4 4 6 7 1 2 0 3 3 5 [ 9




***Пирамида стала меньше и перестала быть упорядоченной***

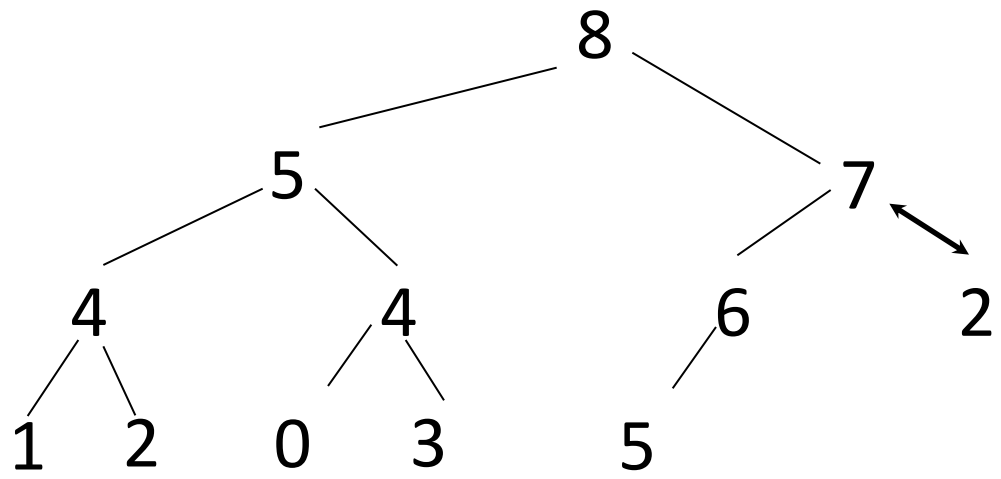
# Вернуть остатку пирамиды упорядоченность

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



# Вернуть остатку пирамиды упорядоченность

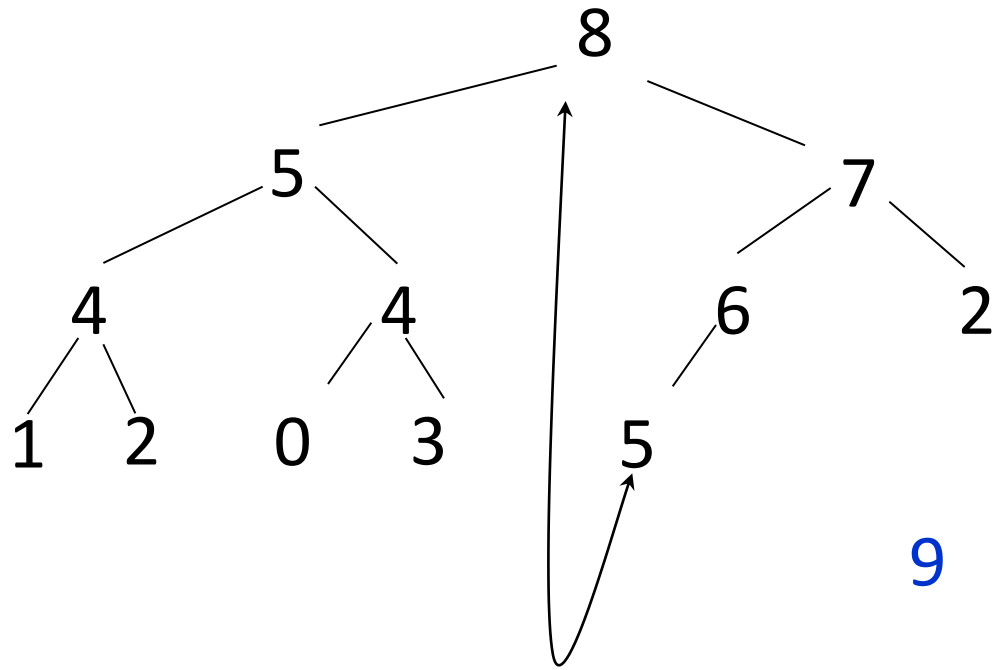
- 8 5 7 4 4 6 2 1 2 0 3 3 5 [ 9




9

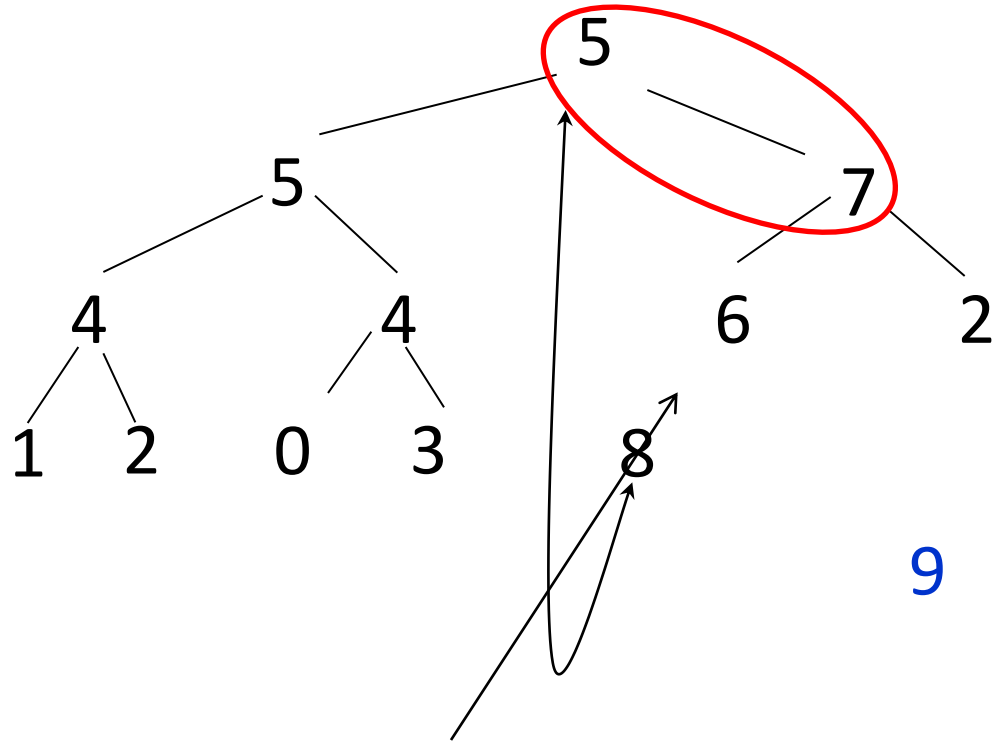
# Поменять корень местами с последним потомком

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



# Поменять корень местами с последним потомком

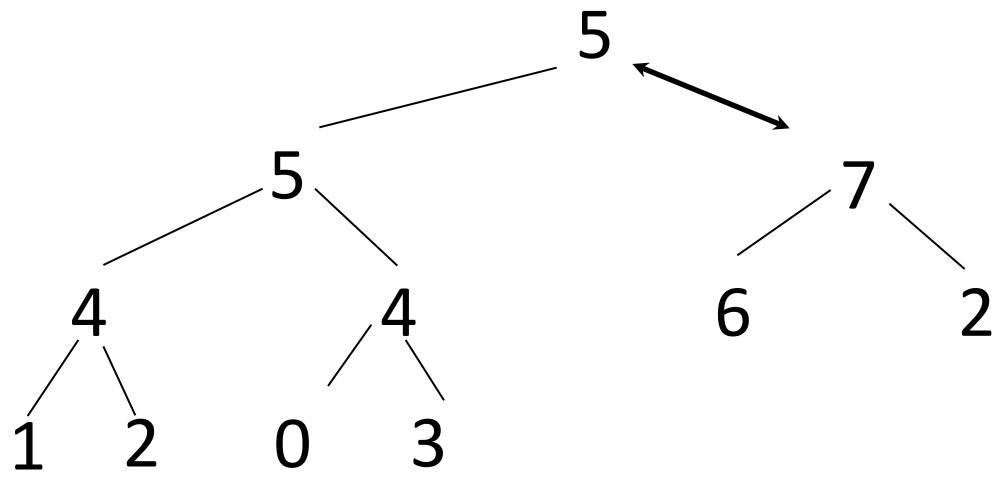
- 8 5 7 4 4 6 2 1 2 0 3 3 5 [ 9




***Пирамида стала меньше и перестала быть упорядоченной***

# Вернуть остатку пирамиды упорядоченность

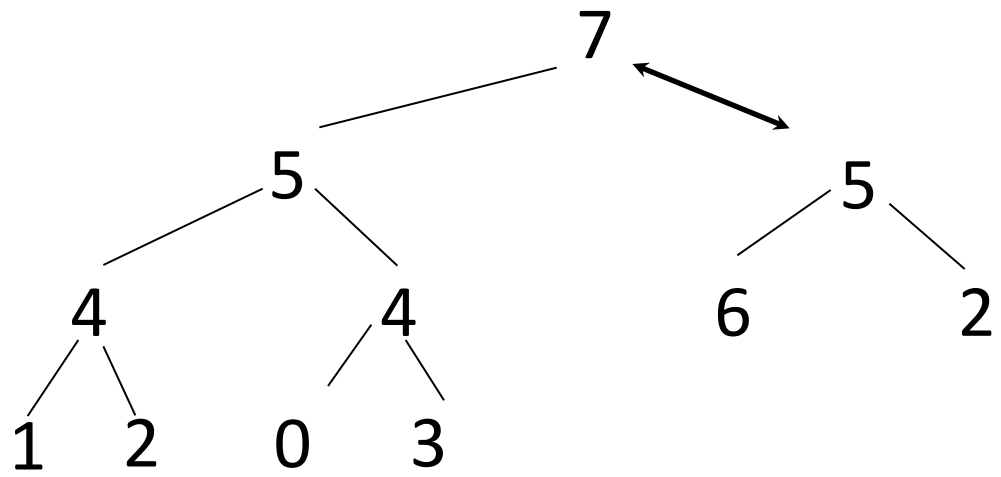
- 5 5 7 4 4 6 2 1 2 0 3 3 [ 8 9




8 9

# Вернуть остатку пирамиды упорядоченность

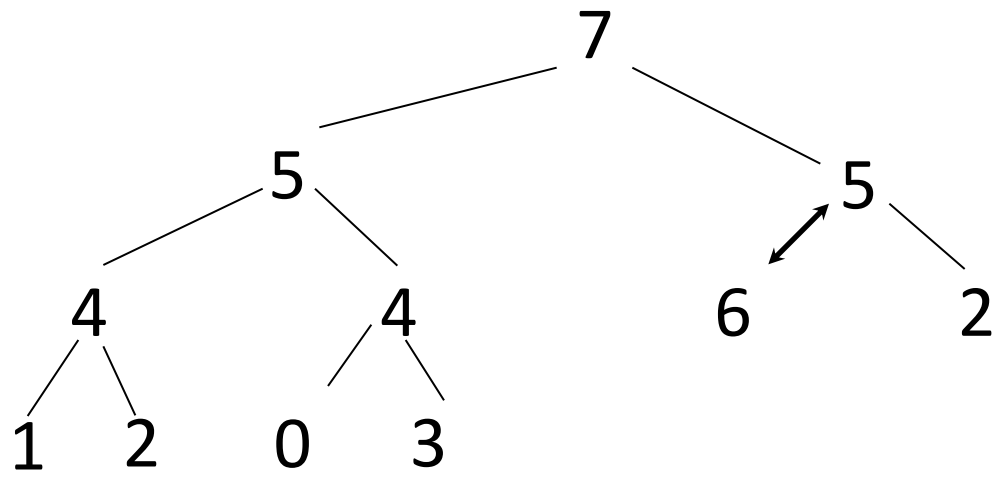
- 5 5 7 4 4 6 2 1 2 0 3 3 [ 8 9




8 9

# Вернуть остатку пирамиды упорядоченность

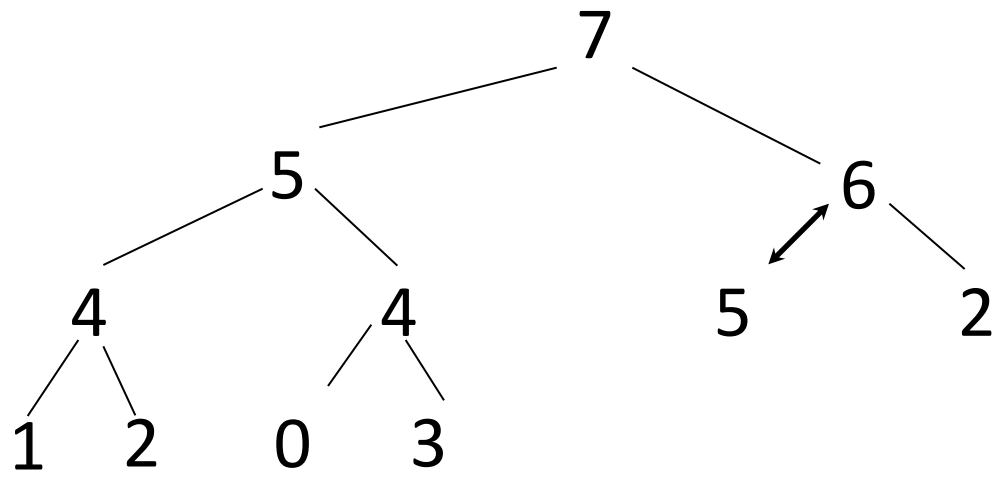
- 7 5 5 4 4 6 2 1 2 0 3 3 [ 8 9




8 9

# Вернуть остатку пирамиды упорядоченность

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




8 9

# Пирамидальная сортировка – хаотичные обращения к памяти

*меняем местами вершину пирамиды и последний элемент пирамиды*

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

*восстанавливаем упорядоченность пирамиды*

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

*меняем местами вершину пирамиды и последний элемент пирамиды*

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

*восстанавливаем упорядоченность пирамиды*

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

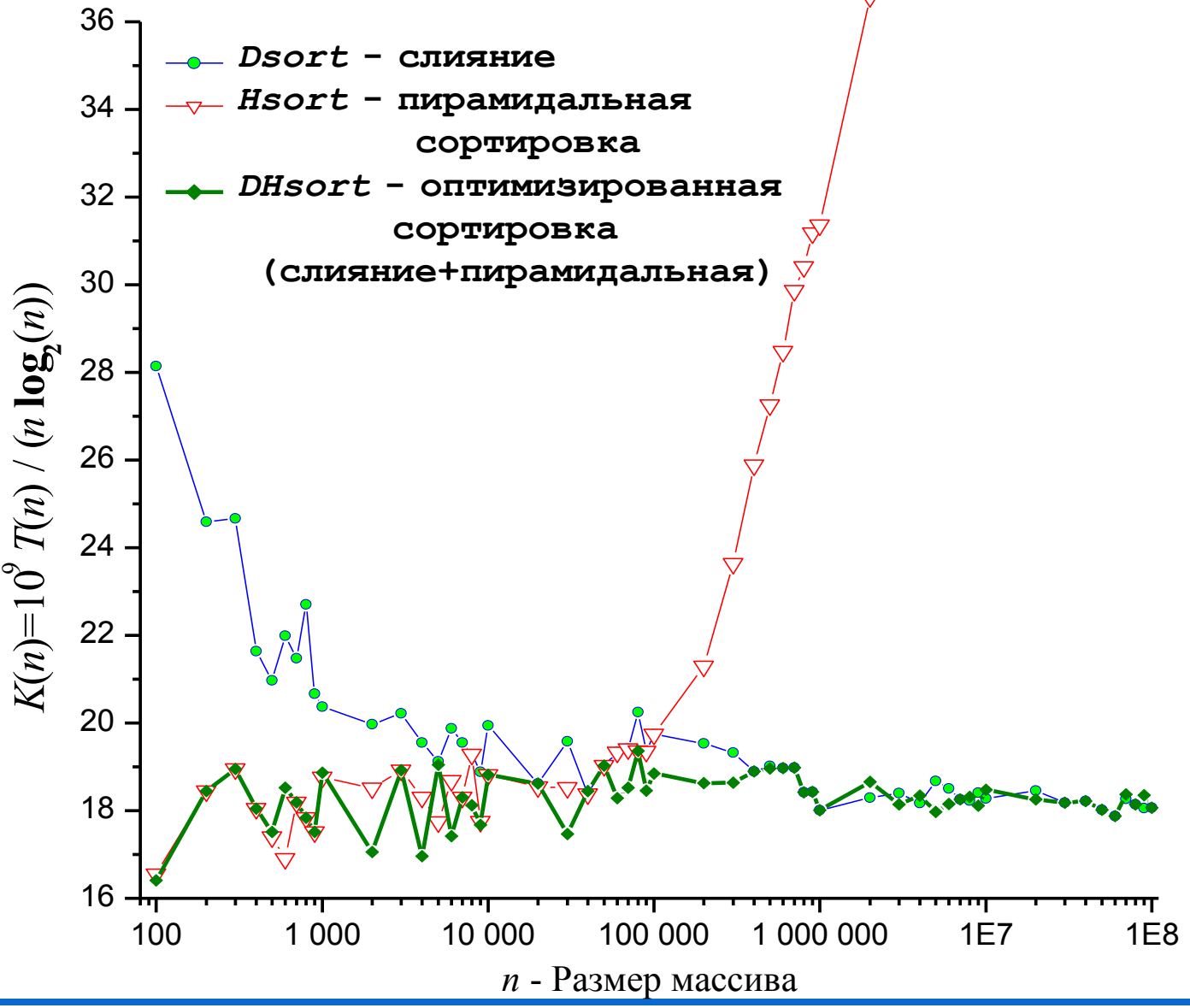
7 5 5 4 4 6 2 1 2 0 3 3 [ 8 9

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

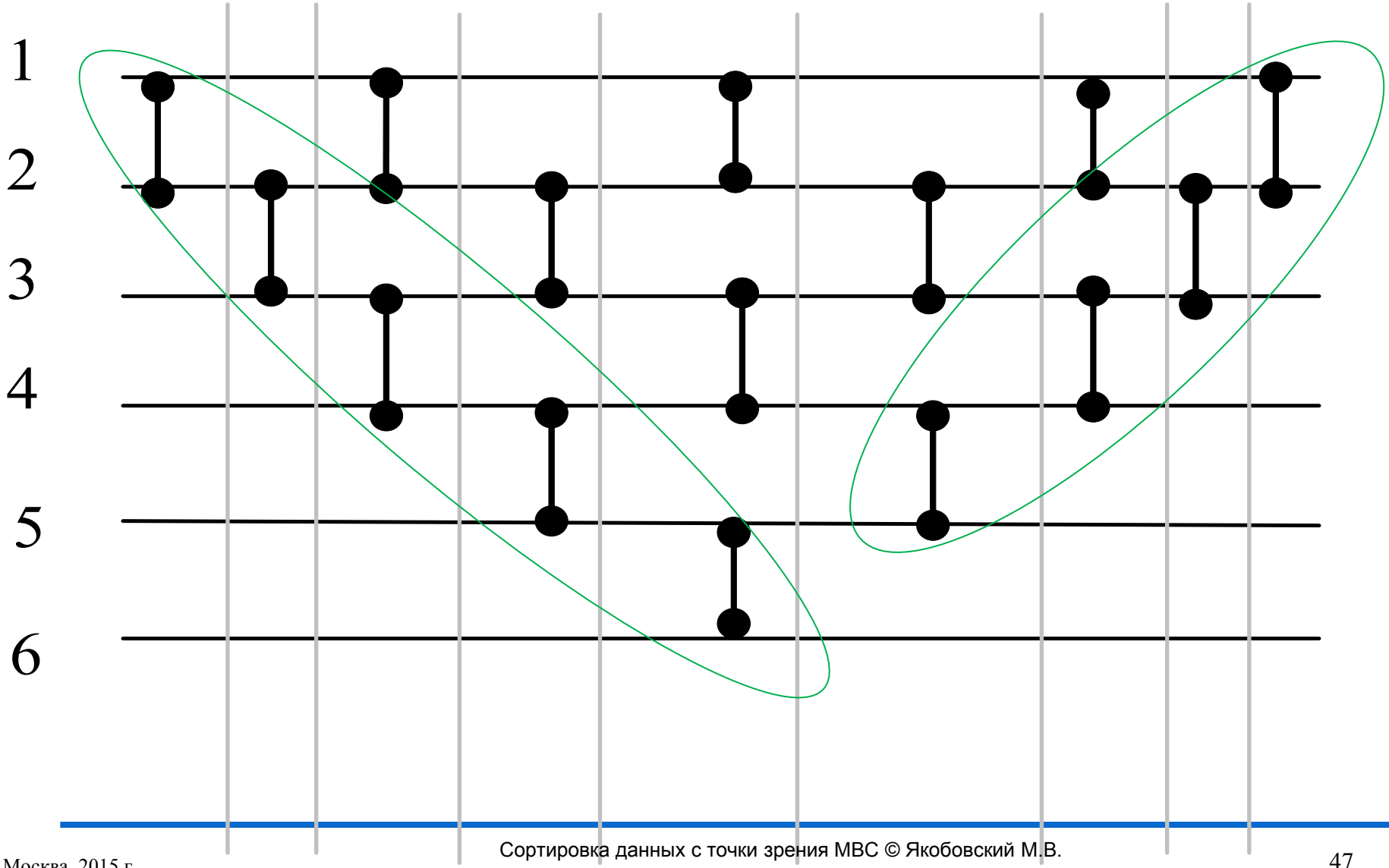
# Оптимальный алгоритм

- Оптимальна комбинация:
- H алгоритм (пирамидальная сортировка) при n от 10 до 50 000
- DH алгоритм (пирамидальная сортировка блоков размером до 50 000 и их последующее слияние) при n больше 50 000

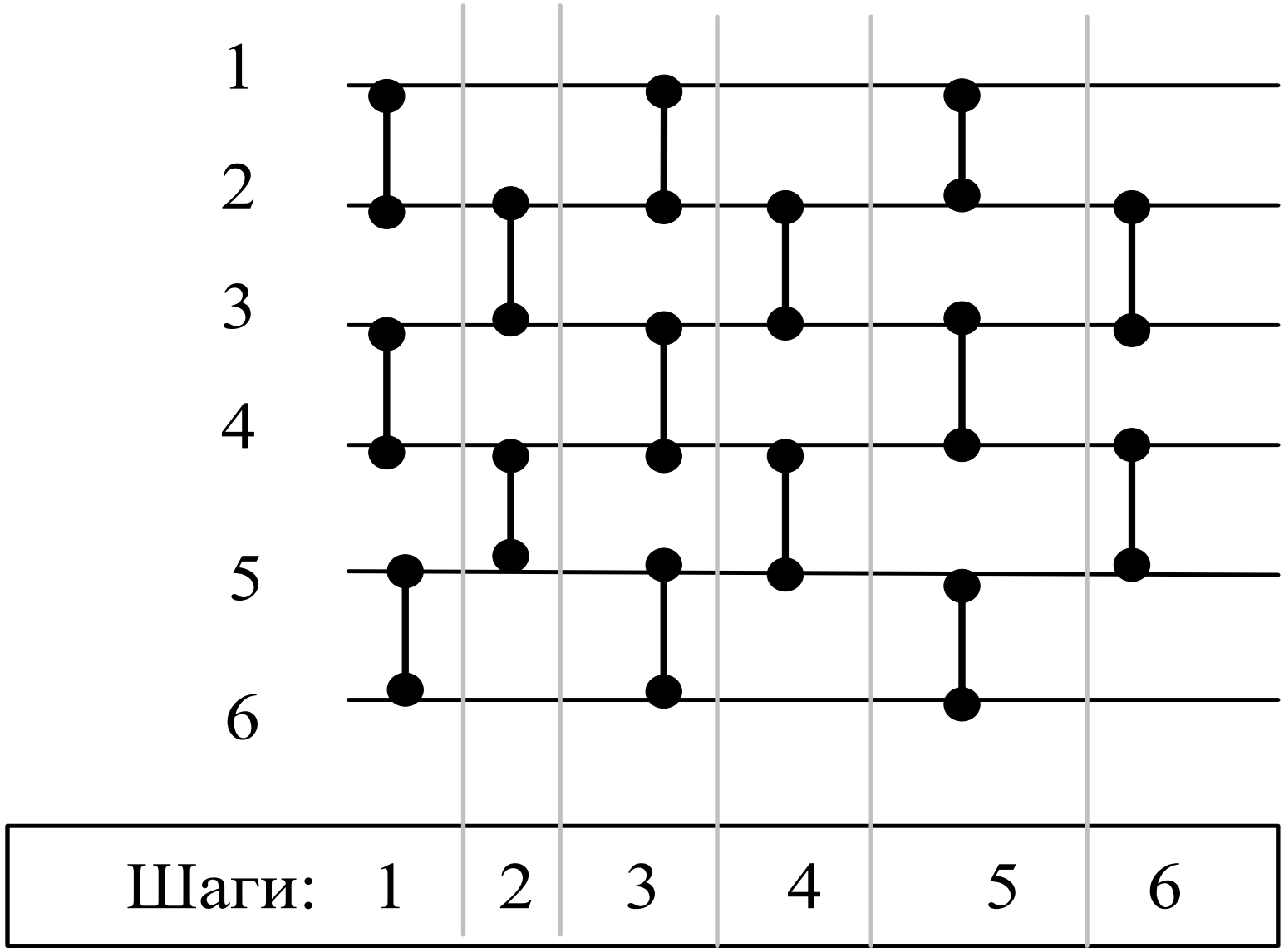
# Константа времени сортировки наилучшего алгоритма



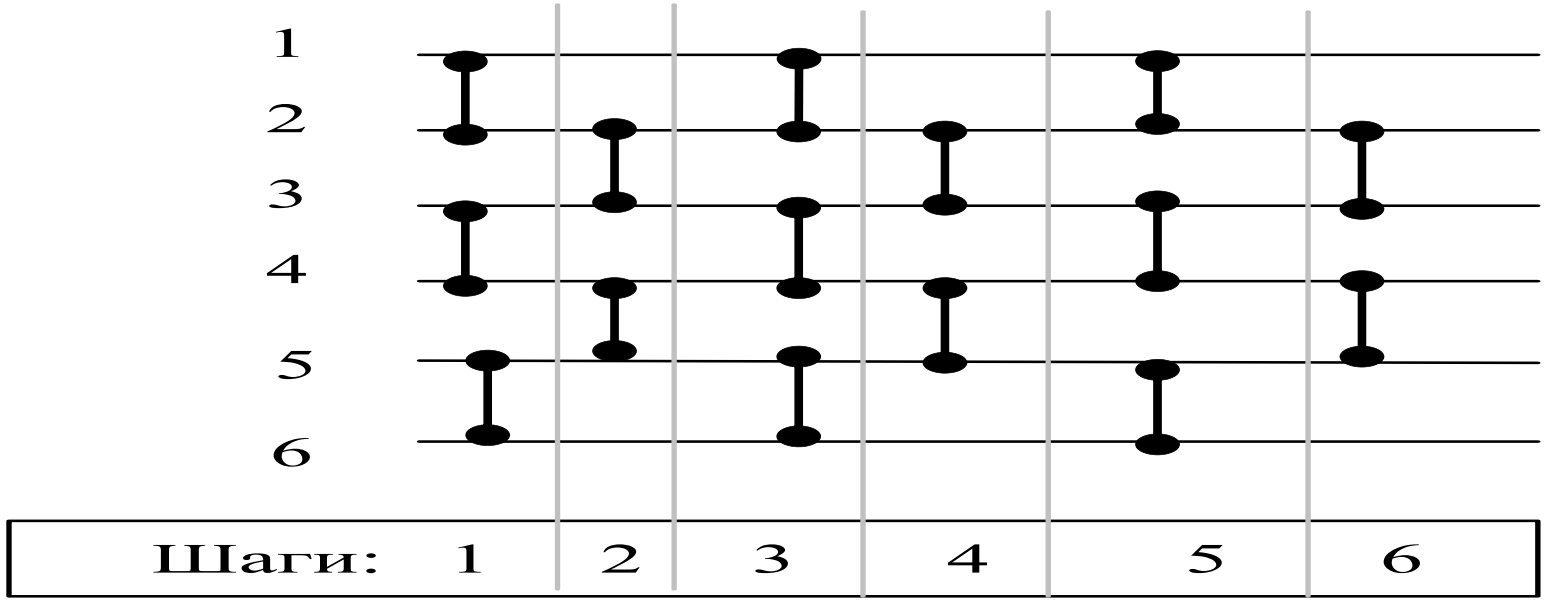
# Сеть сортировки (пузырёк) *n=6 s=2n-3=9*



# Сеть сортировки четно-нечетные перестановки *n=6 s=n=6*



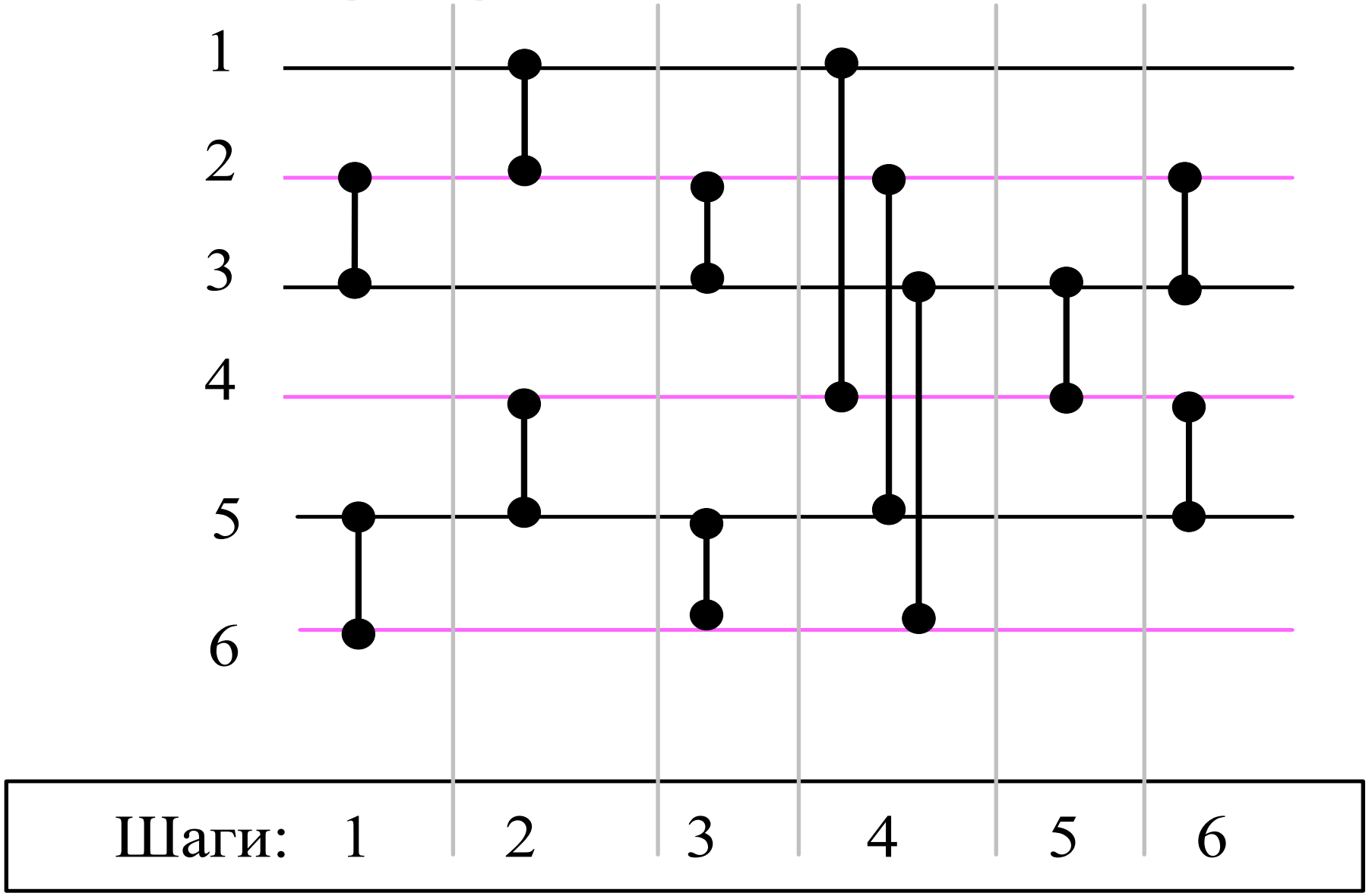
# Сеть сортировки четно-нечетные перестановки *n=6 s=n=6*



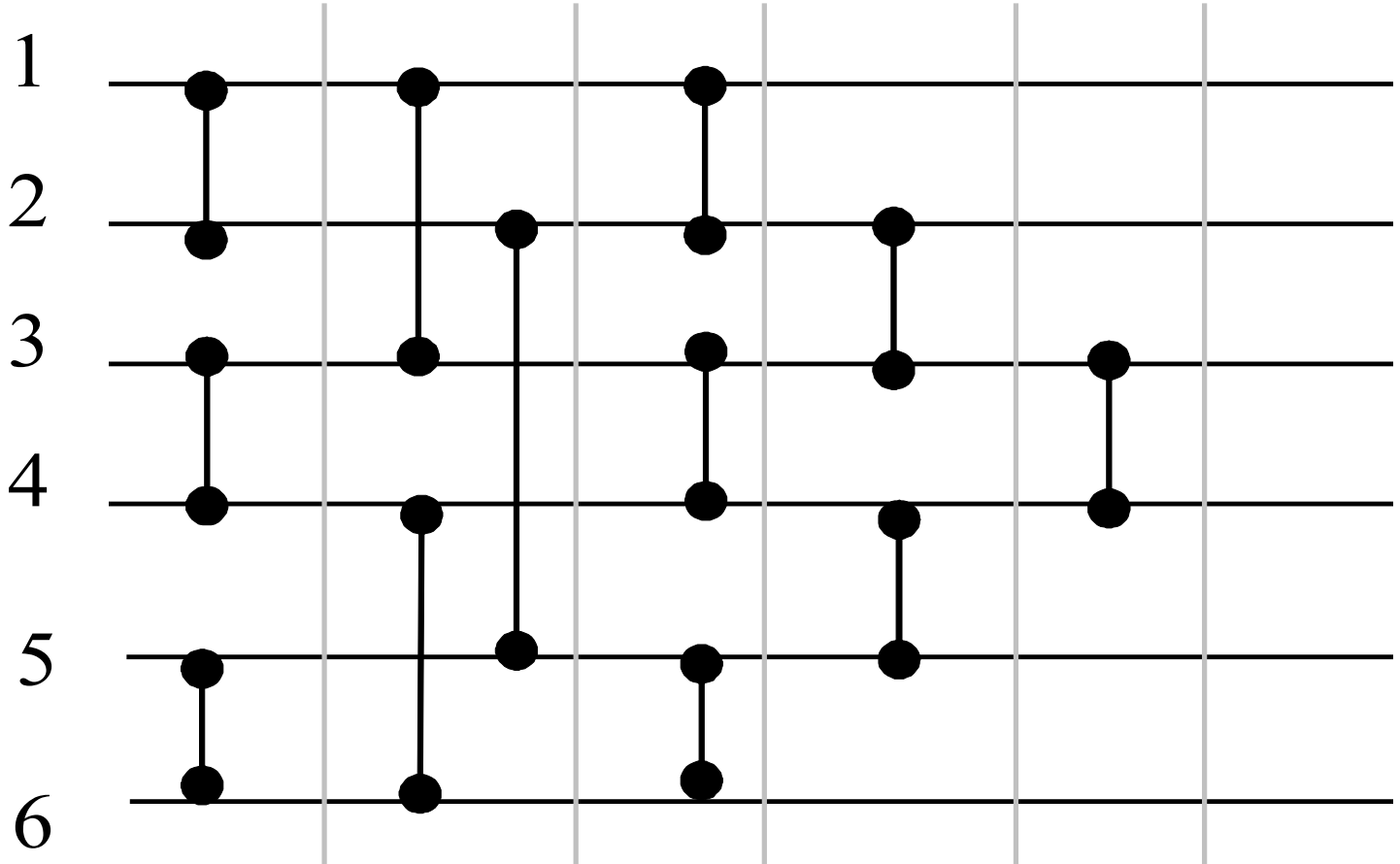

$$O\left(\frac{n}{p}\left[\log_2\frac{n}{p}+p\right]\right)=O\left(\frac{n}{p}\log_2\frac{n}{p}+n\right)\approx O(n)$$

$$T_p=O(n)$$

# Сеть сортировки n=6 s=6



# Минимальная сеть сортировки n=6 s=5

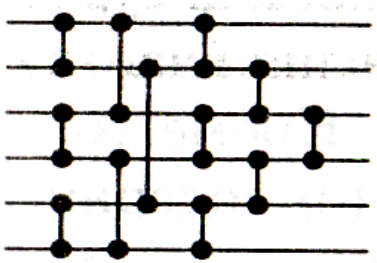



Шаги: 1 | 2 | 3 | 4 | 5

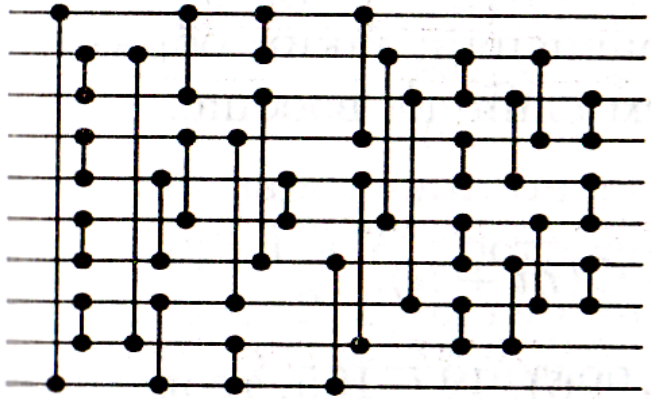
# Минимальные сети сортировки

[Дн.Кнут]

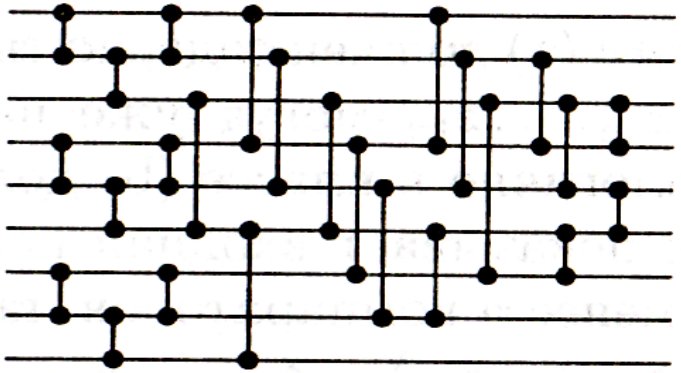
n=6 s=5



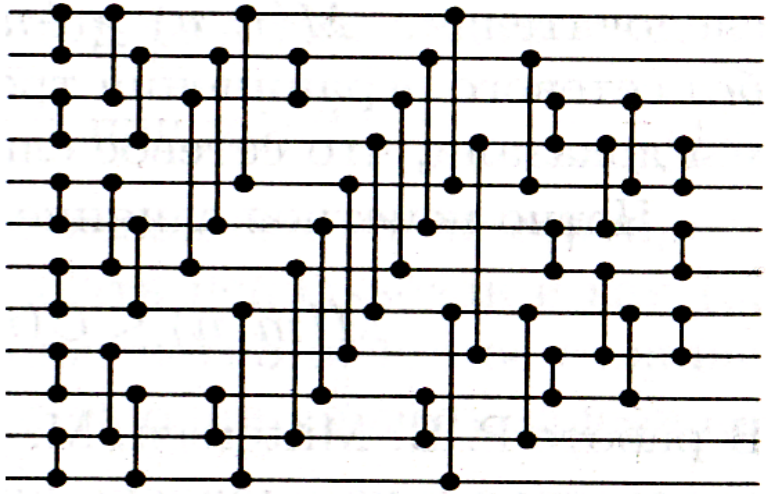
n=10 s=7



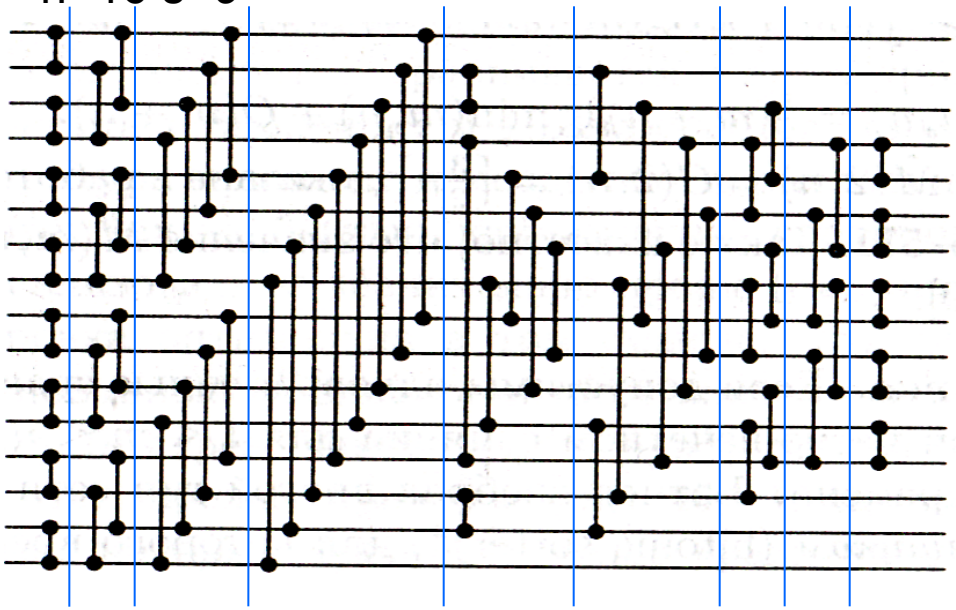
n=9 s=8



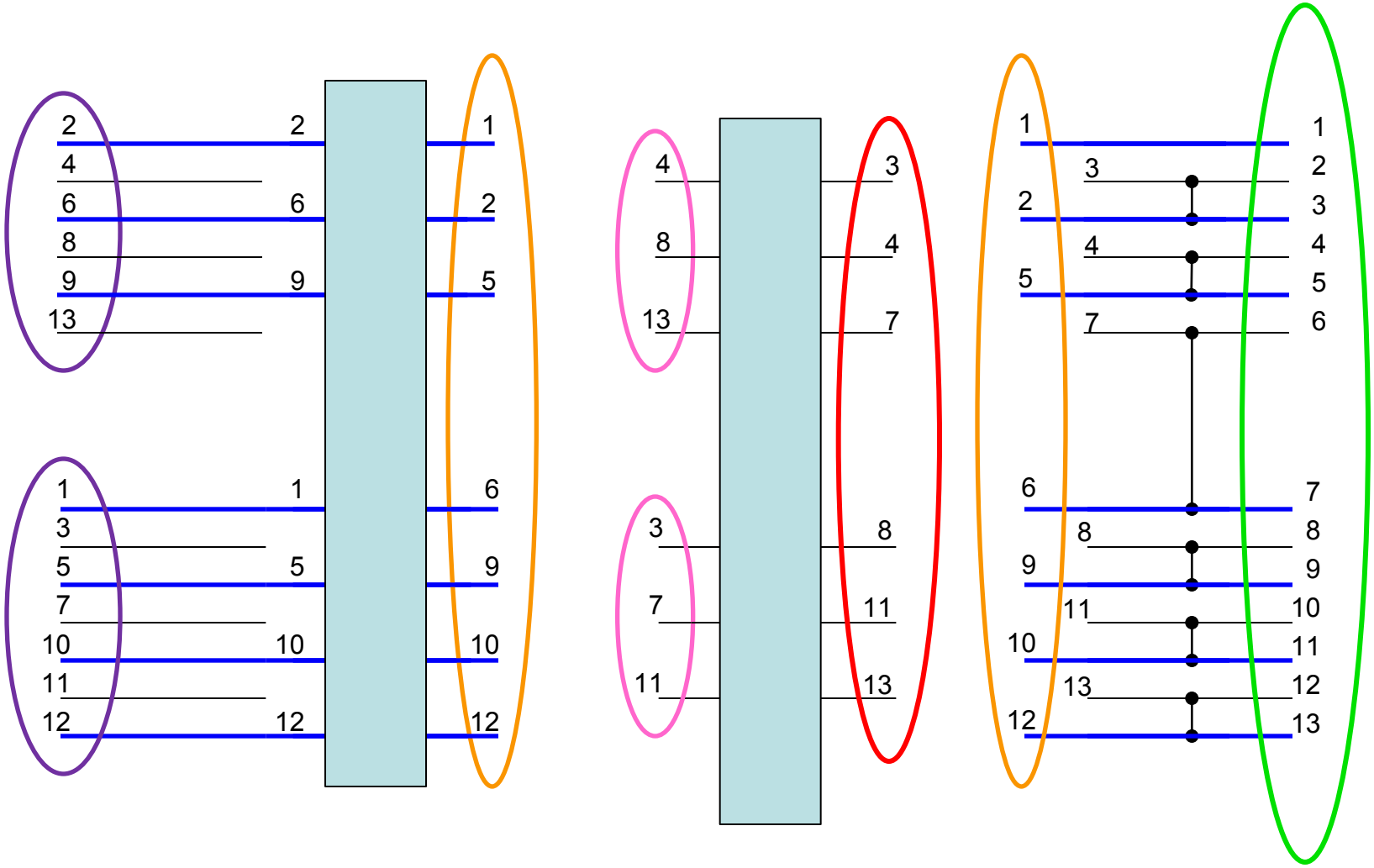
n=12 s=8



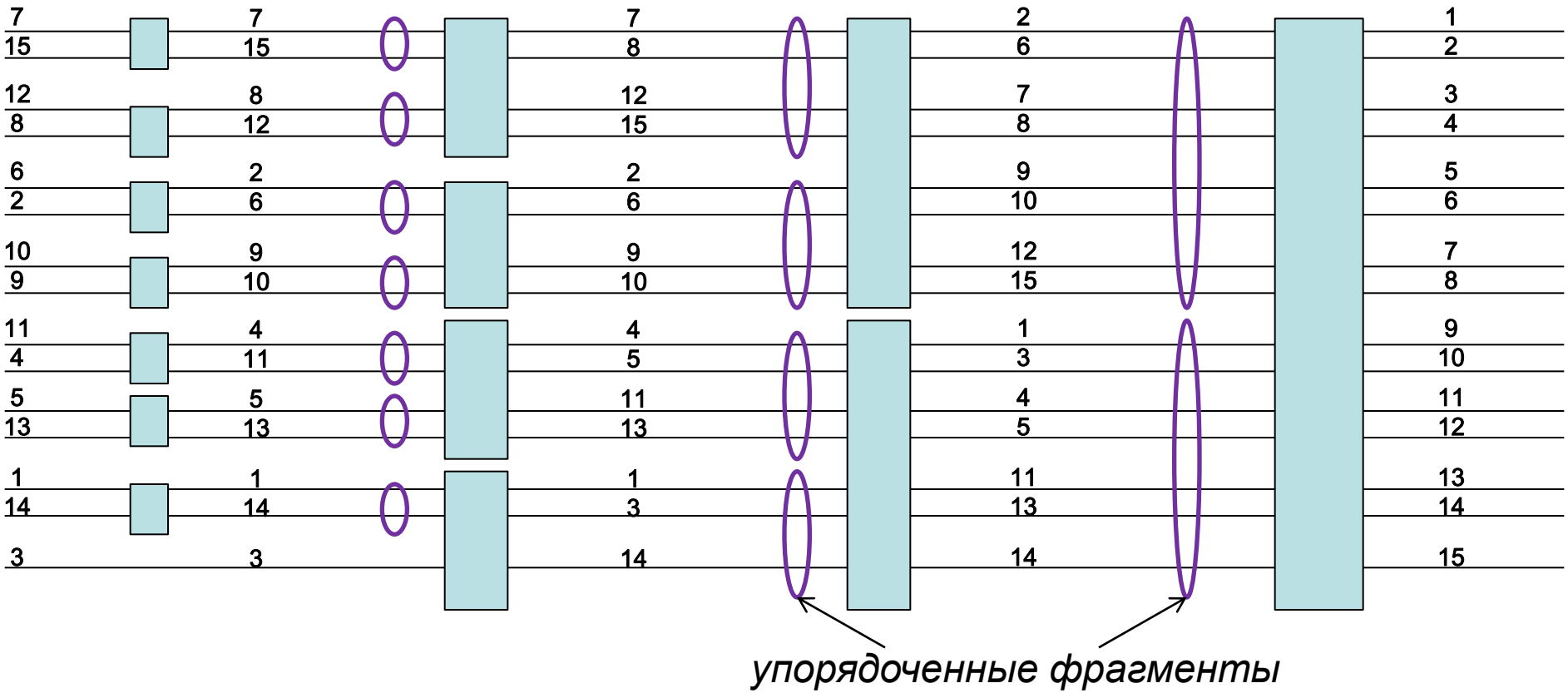
n=16 s=9



# Четно-нечетное слияние Бэтчера – масштабируемая сеть

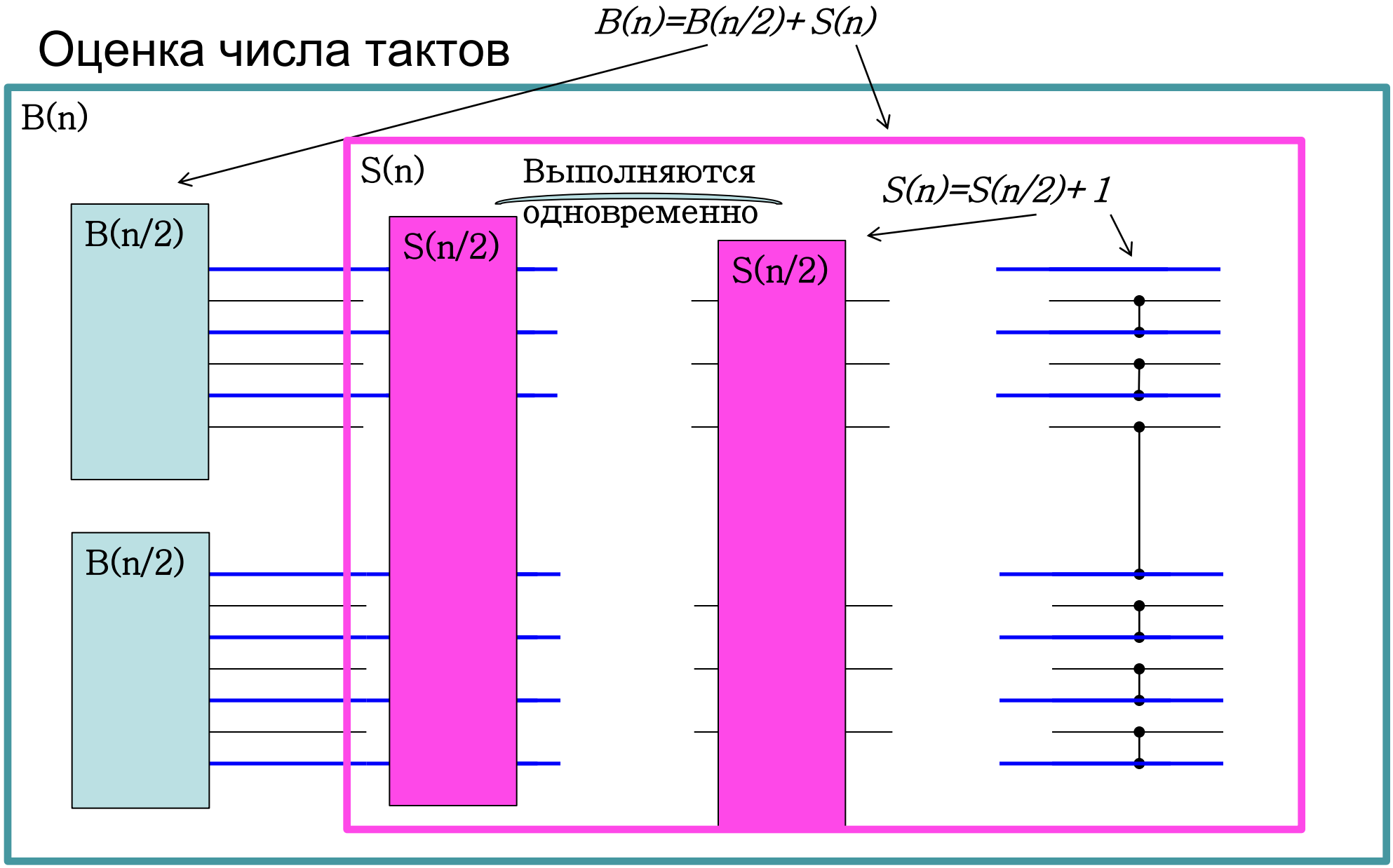


# Сортировка массива из 15 элементов на основе четно-нечетного слияния Бэтчера




- *сеть четно-нечетного слияния Бетчера*

# Оценка числа тактов



# Оценка числа последовательно расположенных групп компараторов

$$B(n)=B(n/2)+S(n)$$

$$S(n)=S(n/2)+1$$

# Оценка числа последовательно расположенных групп компараторов

$B(n)=B(n/2)+S(n)$

$S(n)=S(n/2)+1$ $\qquad\qquad$ $S(n)=\log_2(n)$

# Оценка числа последовательно расположенных групп компараторов

$B(n)=B(n/2)+S(n)$

$S(n)=S(n/2)+1$ $\qquad$ $S(n)=\log_2(n)$

$B(n)=B(n/2)+\log_2(n)$

# Оценка числа последовательно расположенных групп компараторов

$B(n)=B(n/2)+S(n)$

$S(n)=S(n/2)+1$ $\qquad$ $S(n)=\log_2(n)$

$B(n)=B(n/2)+\log_2(n)$

$B(n)=\log_2(n)\ (\log_2(n)+1)\ /\ 2$

# Оценка числа последовательно расположенных групп компараторов

$B(n)=B(n/2)+S(n)$

$S(n)=S(n/2)+1$ $\qquad$ $S(n)=\log_2(n)$

$B(n)=B(n/2)+\log_2(n)$

$B(n)=\log_2(n)\ (\log_2(n)+1)\ /\ 2$

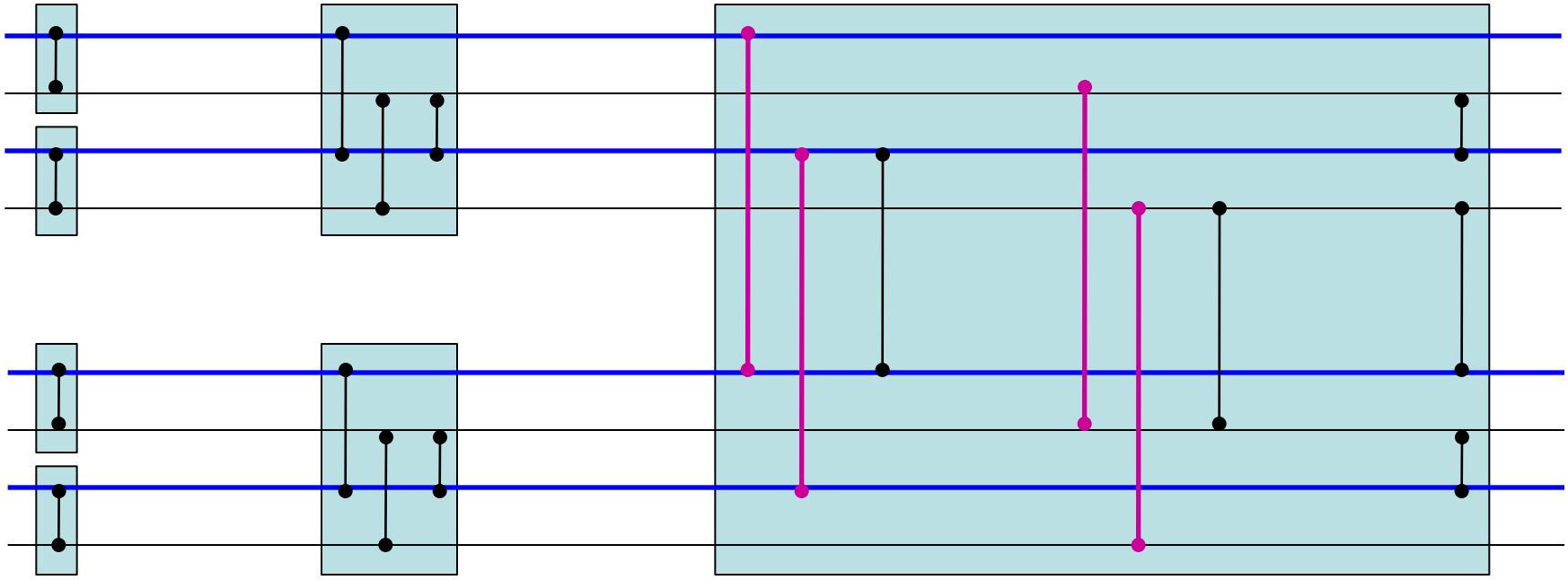
$$s_p \approx \frac{\lceil \log_2 p \rceil \left( \lceil \log_2 p \rceil + 1 \right)}{2}$$

# Сортировка восьми элементов

$$O\left(\frac{n}{p}\left[\log_2\frac{n}{p}+\frac{\lceil\log_2 p\rceil^2}{2}\right]\right)$$

*или*

*n* элементов восемью процессорами

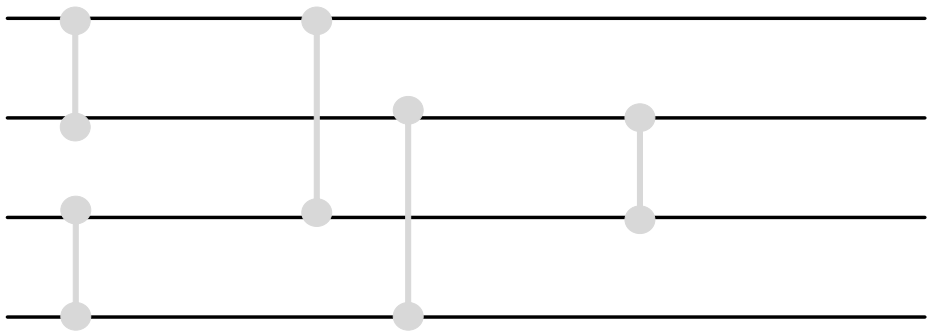


*- сеть четно-нечетного слияния Бетчера*

# Пример работы алгоритма
# Начало, массив распределен по процессорам

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

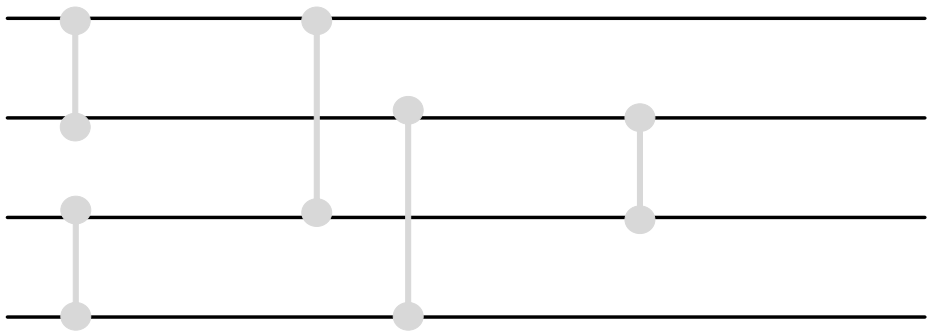
8 7 4
3 9 2
5 1 2
4 0 6



# Сортируем фрагменты

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

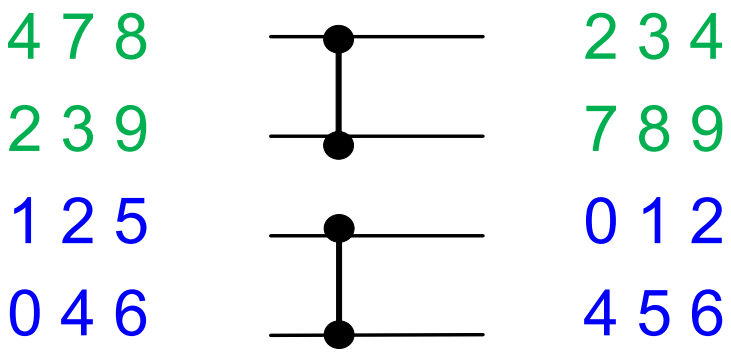
4 7 8
2 3 9
1 2 5
0 4 6



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

# Первые два компаратора слияния перестановки

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

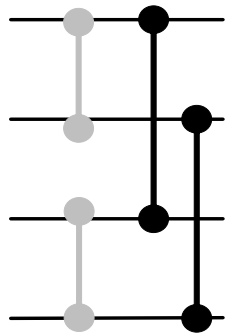



2 3 4  7 8 9  0 1 2  4 5 6

# Вторая пара компараторов слияния перестановки

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

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

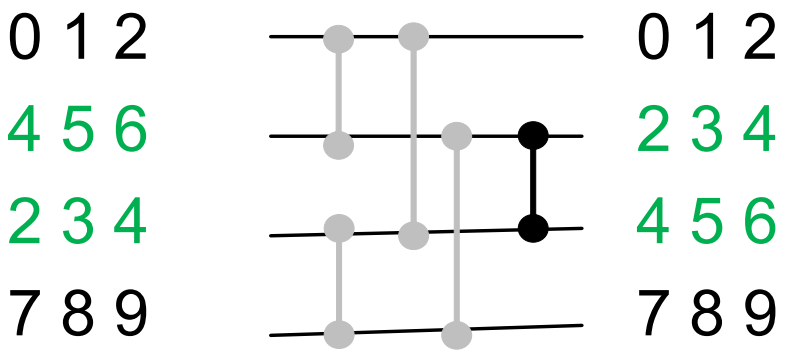


0 1 2
4 5 6
2 3 4
7 8 9

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

# Последний компаратор слияния перестановки

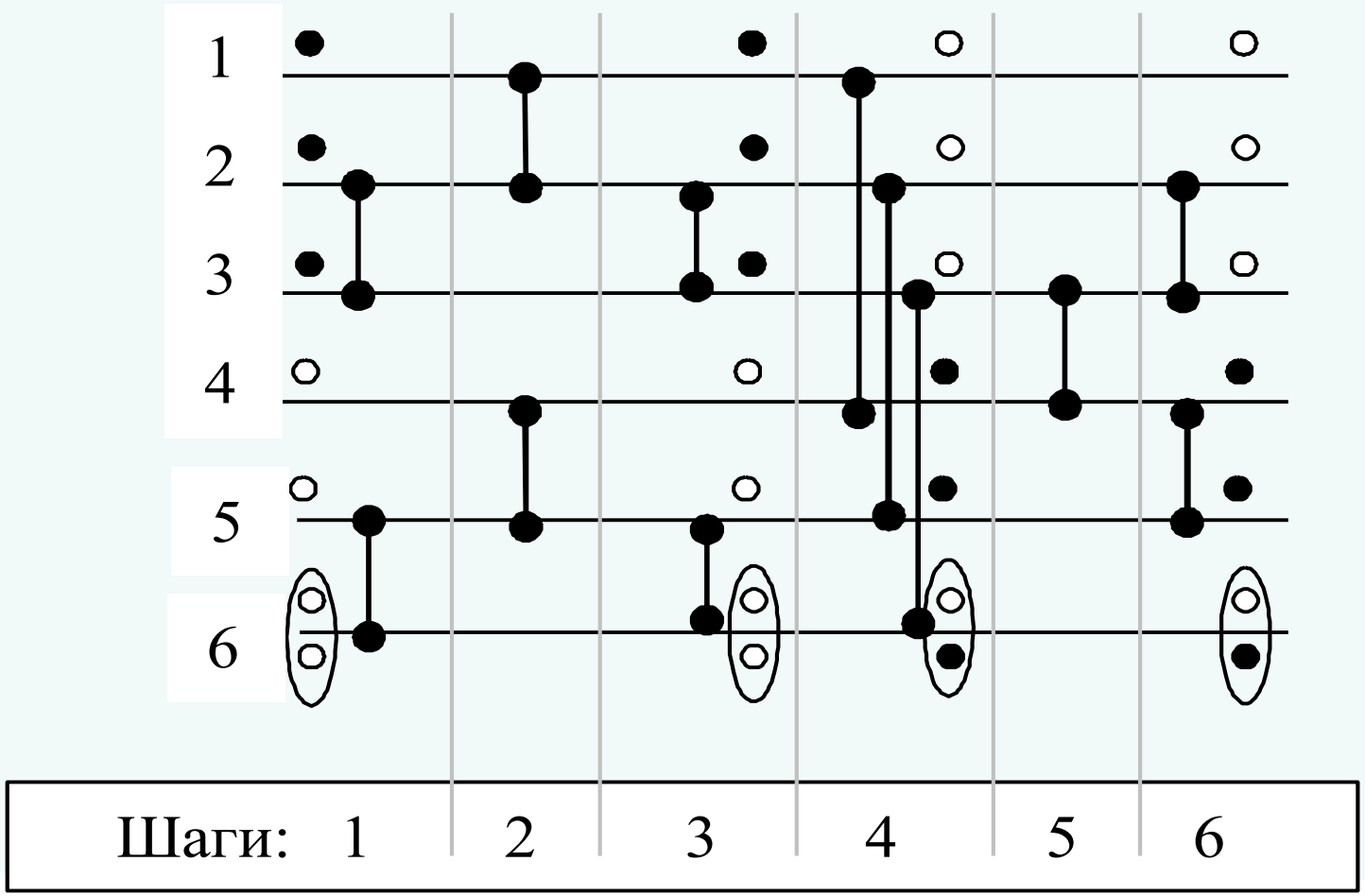
0 1 2  4 5 6  2 3 4  7 8 9




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

Массив упорядочен

# Ограничение метода:
# Сортировка блоков – ОДИНАКОВОГО РАЗМЕРА



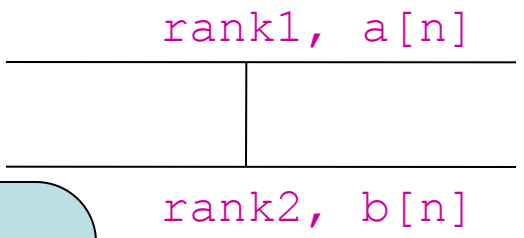
# Слияние упорядоченных фрагментов

```
// объединить два упорядоченных массива a,b

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

# Слияние упорядоченных фрагментов

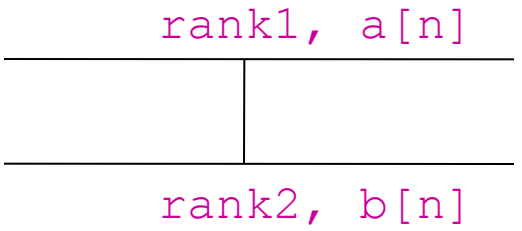
```
for(ia=0,ib=0,k=0;k<n;k++)
    {
    if(ia>=n1) c[k]=b[ib++];
    else
    if(ib>=n2) c[k]=a[ia++];
    else
    if(a[ia]<b[ib]) c[k]=a[ia++];
    else            c[k]=b[ib++];
    }
```




```
// n – число элементов в каждом из массивов a,b
```

# Слияние упорядоченных фрагментов

```
for(ia=0,ib=0,k=0;k<n;k++)
     {
```

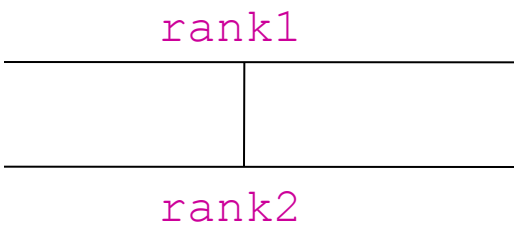



```
     if(a[ia]<b[ib]) c[k]=a[ia++];
     else            c[k]=b[ib++];
     }
```

```
// n – число элементов в каждом из массивов a,b
```

# Слияние упорядоченных фрагментов

```
Join(int *a, int *b, int *c, int n,rank1,rank2)
{
if(rank==rank1)
       for(ia=0,ib=0,k=0;k<n;)
              {
              if(a[ia]<b[ib]) c[k++]=a[ia++];
              else            c[k++]=b[ib++];
              }
else
       for(ia=n-1,ib=n-1,k=n-1;k>=0;)
              {
              if(a[ia]>b[ib]) c[k--]=a[ia--];
              else            c[k--]=b[ib--];
              }
}
```



# Реализация компаратора слияния

```
// взаимодействие процессоров rank и rankC
int *a,*b,*c,*tmp;
ASend(a,n,rankC);
ARecv(b,n,rankC);
ASync();

Join(a,b,c,n, rank, rankC);

tmp=a;
a=c;
c=tmp;
```

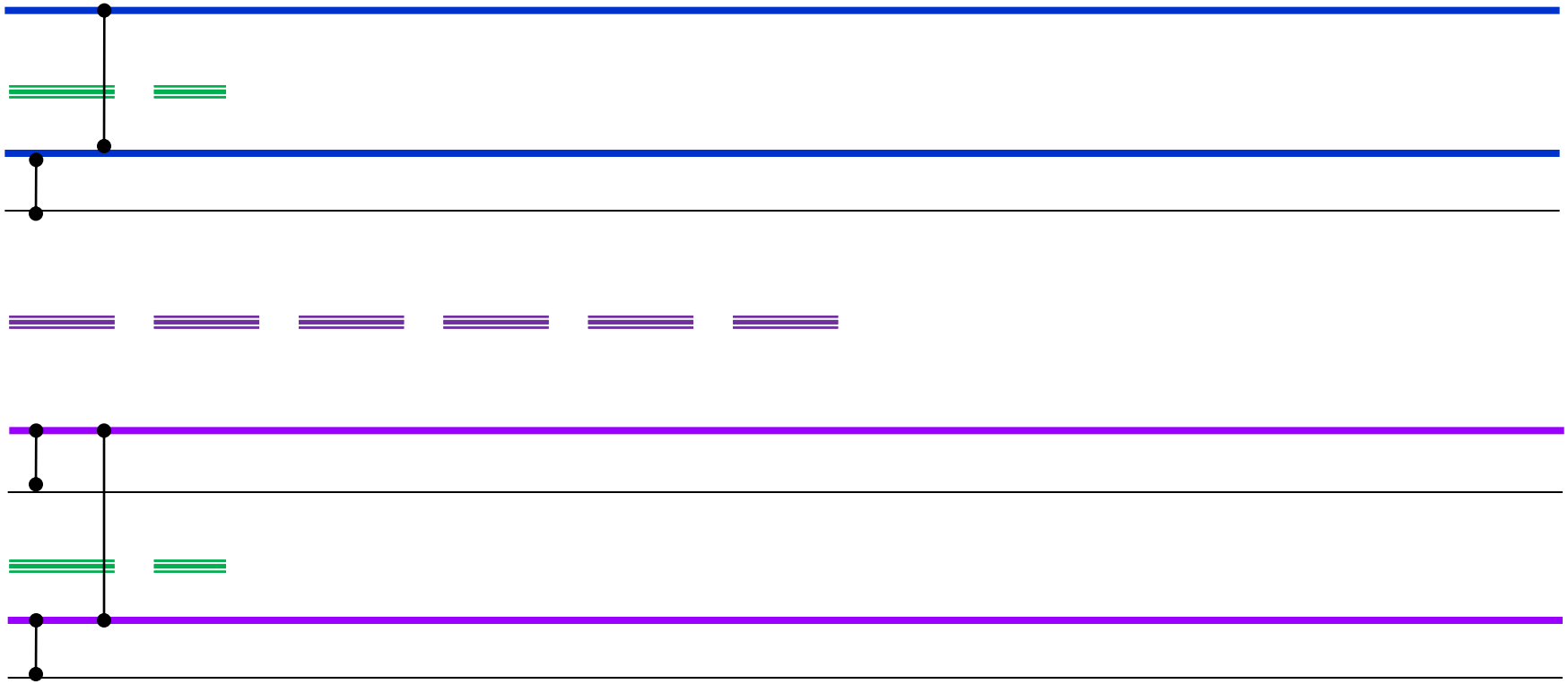
# Сортировка семи элементов

# Разделить на две группы

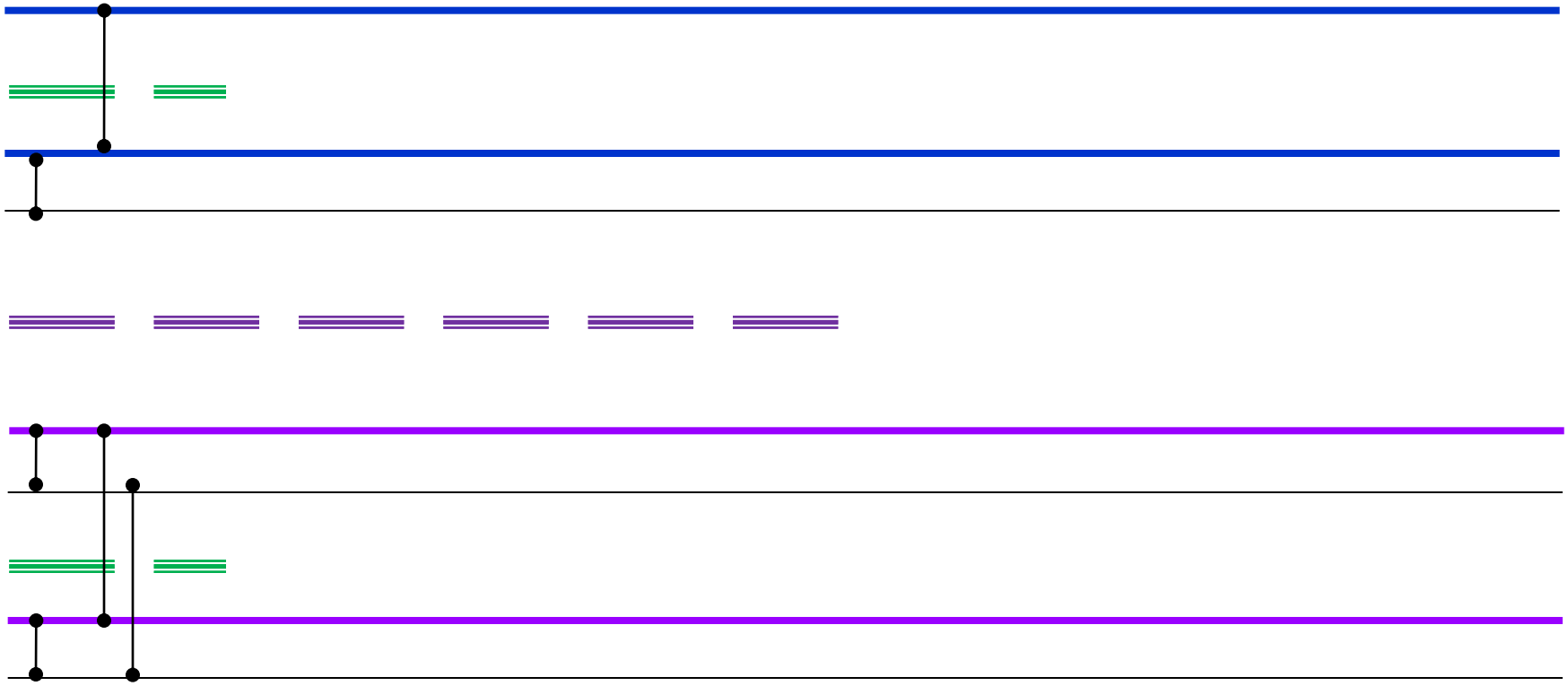
# Снова разделить каждую группу

# Сортировать каждый короткий фрагмент

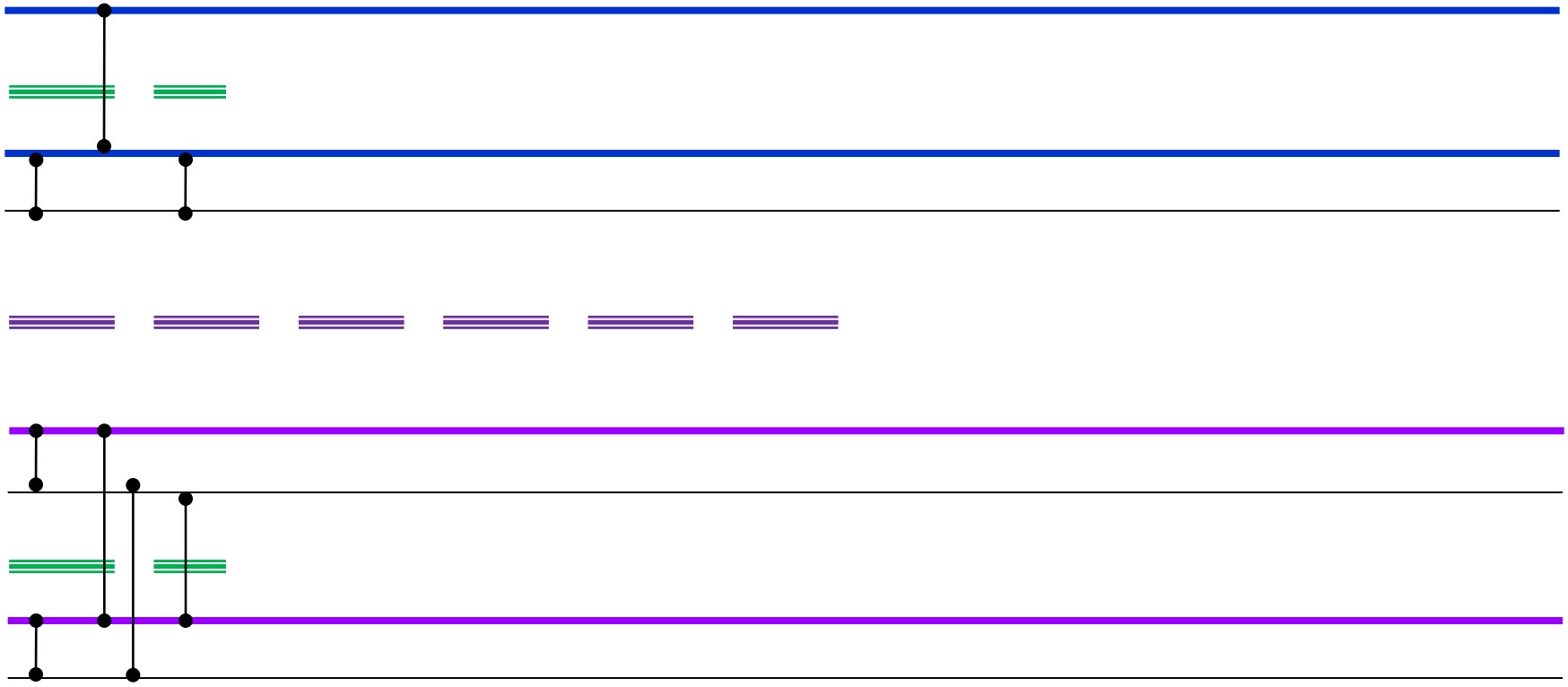
# Сортировать нечетные строки в каждом из фрагментов



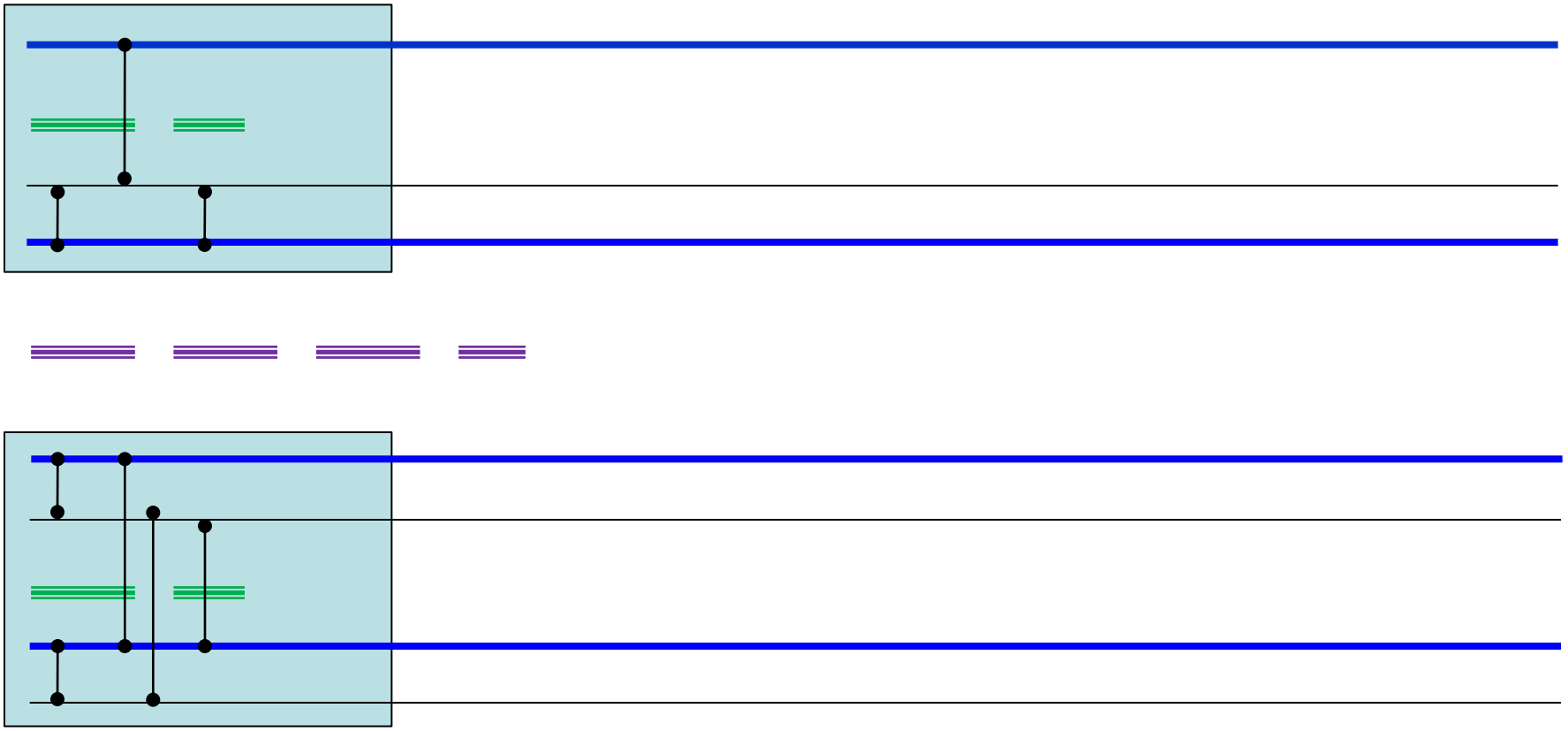
# Сортировать четные строки в каждом из фрагментов



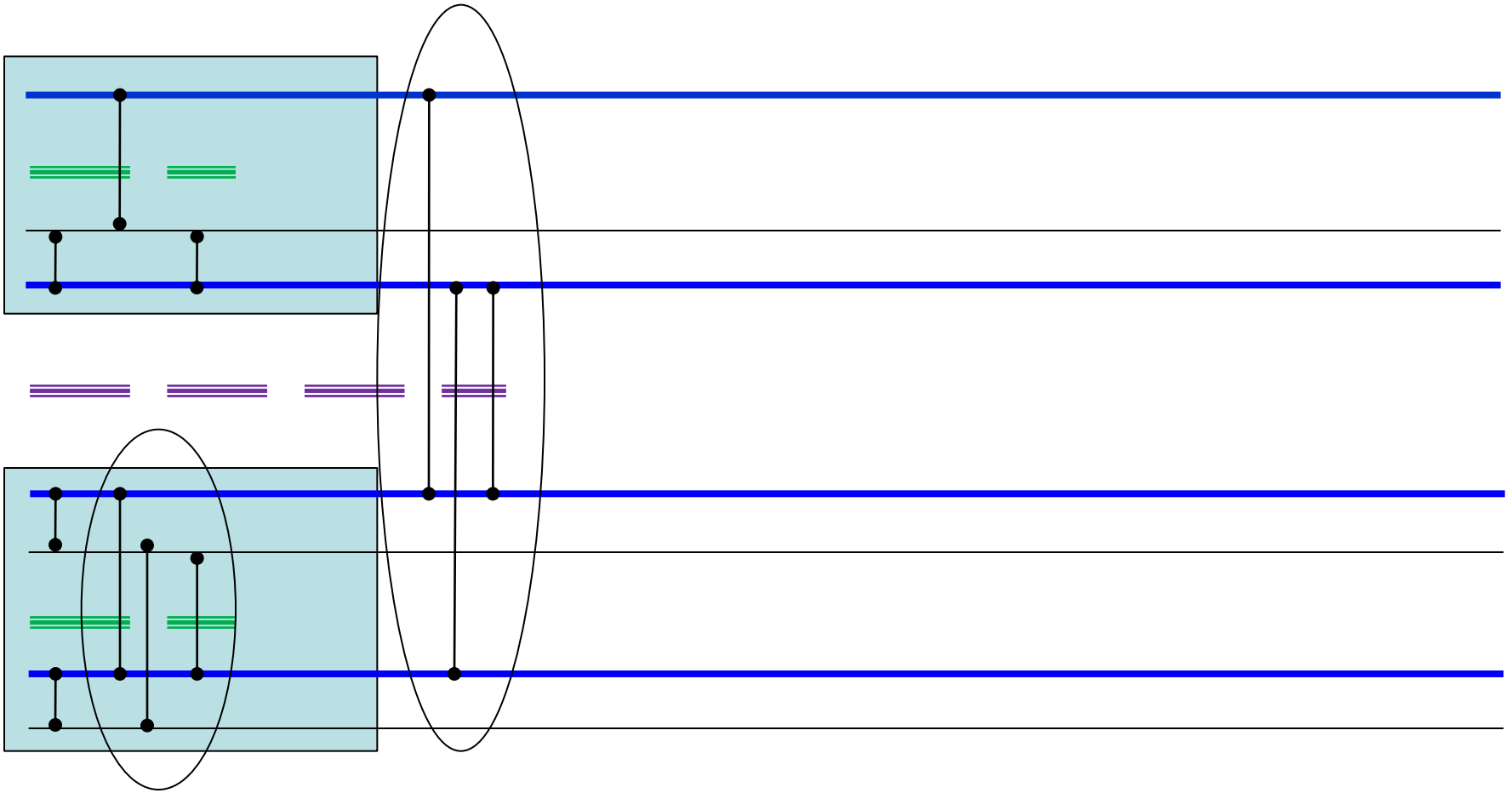
# Объединить отсортированные четные и нечетные строки в каждом фрагменте



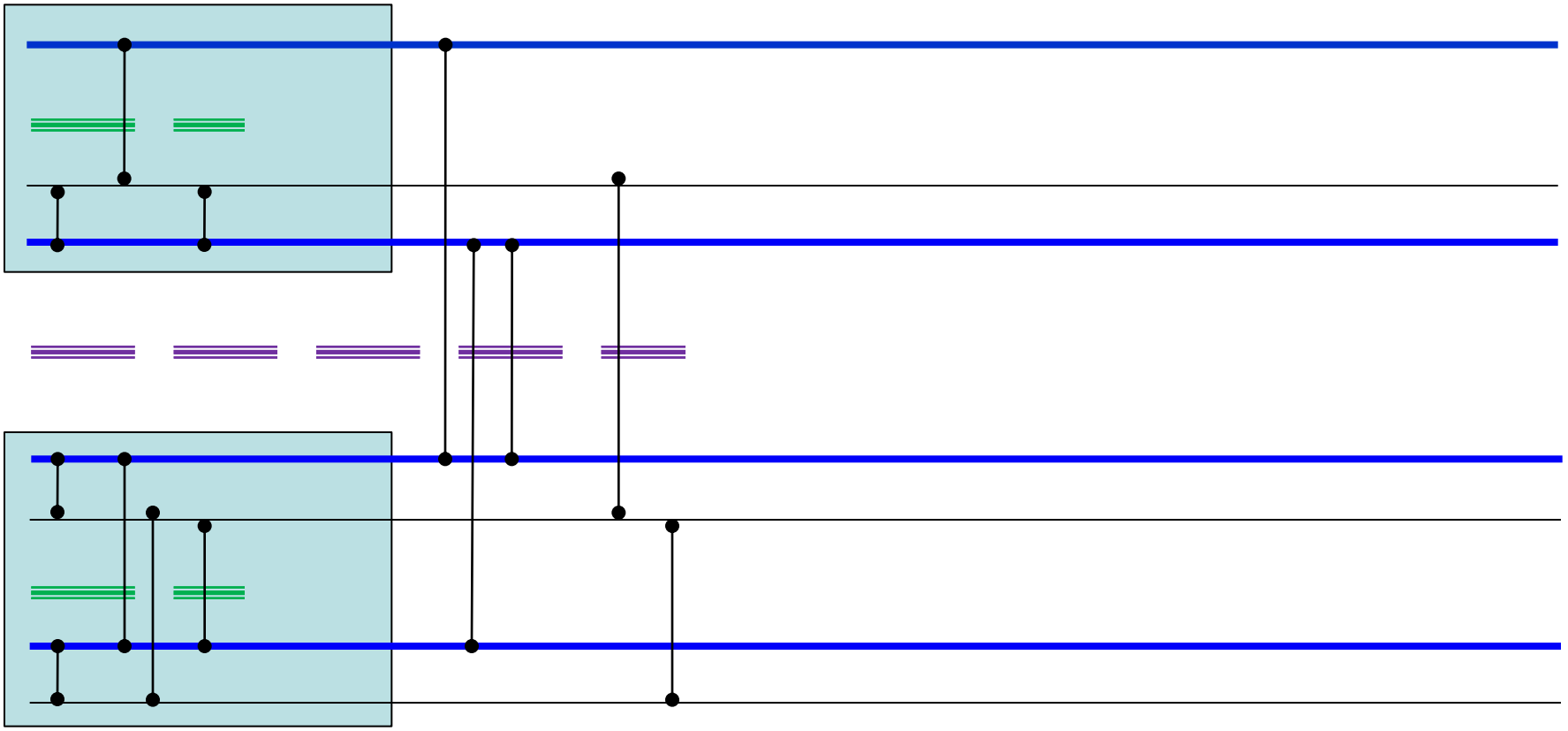
# Объединить два отсортированных фрагмента



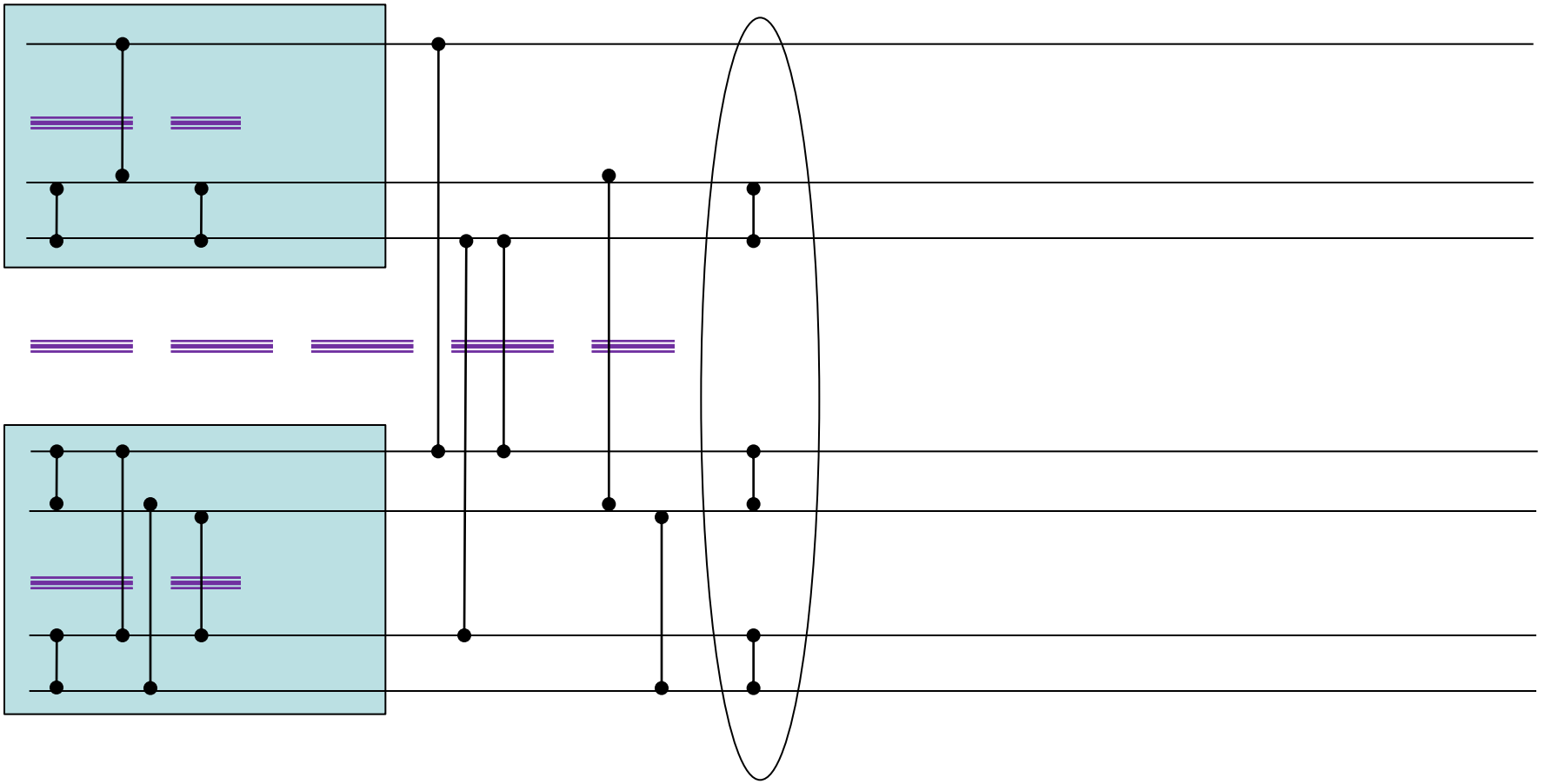
# Объединить четные строки отсортированных фрагментов



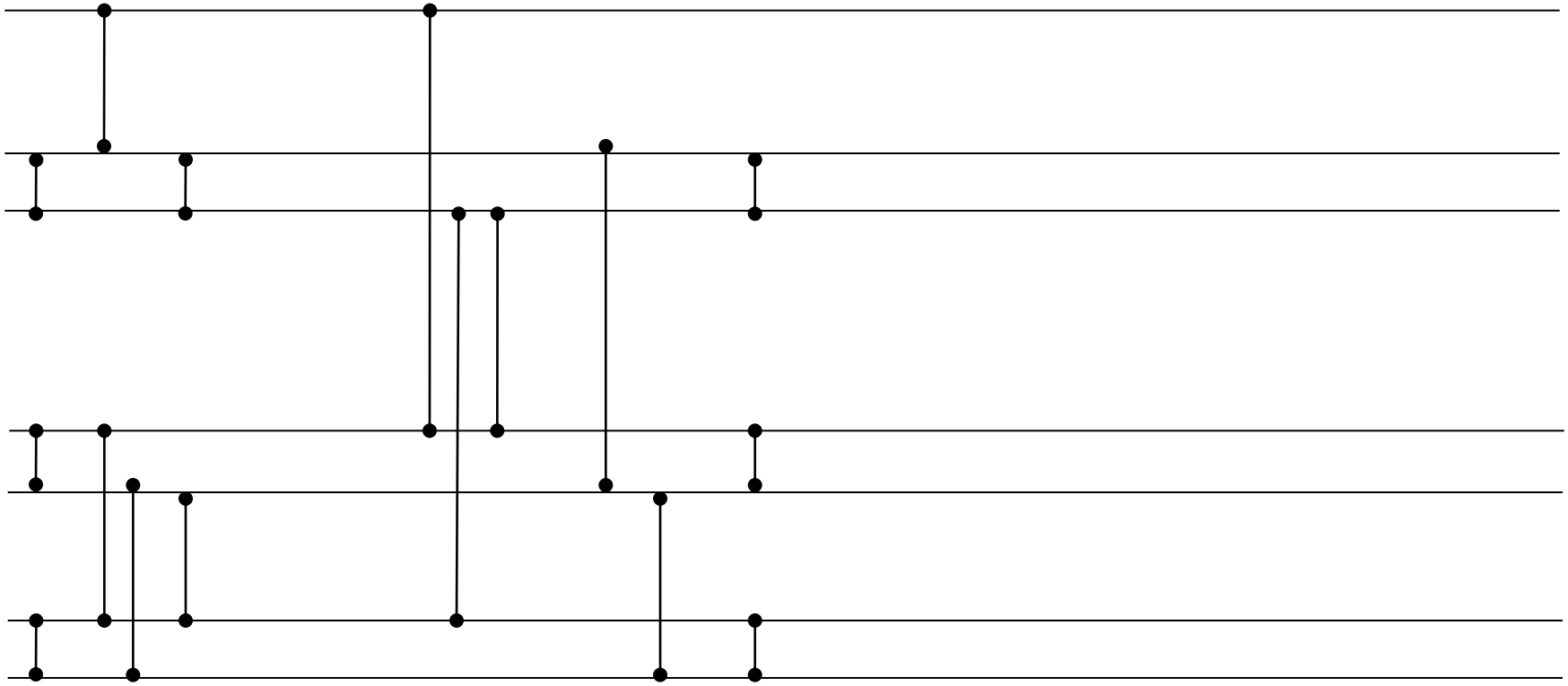
# Объединить нечетные строки отсортированных фрагментов



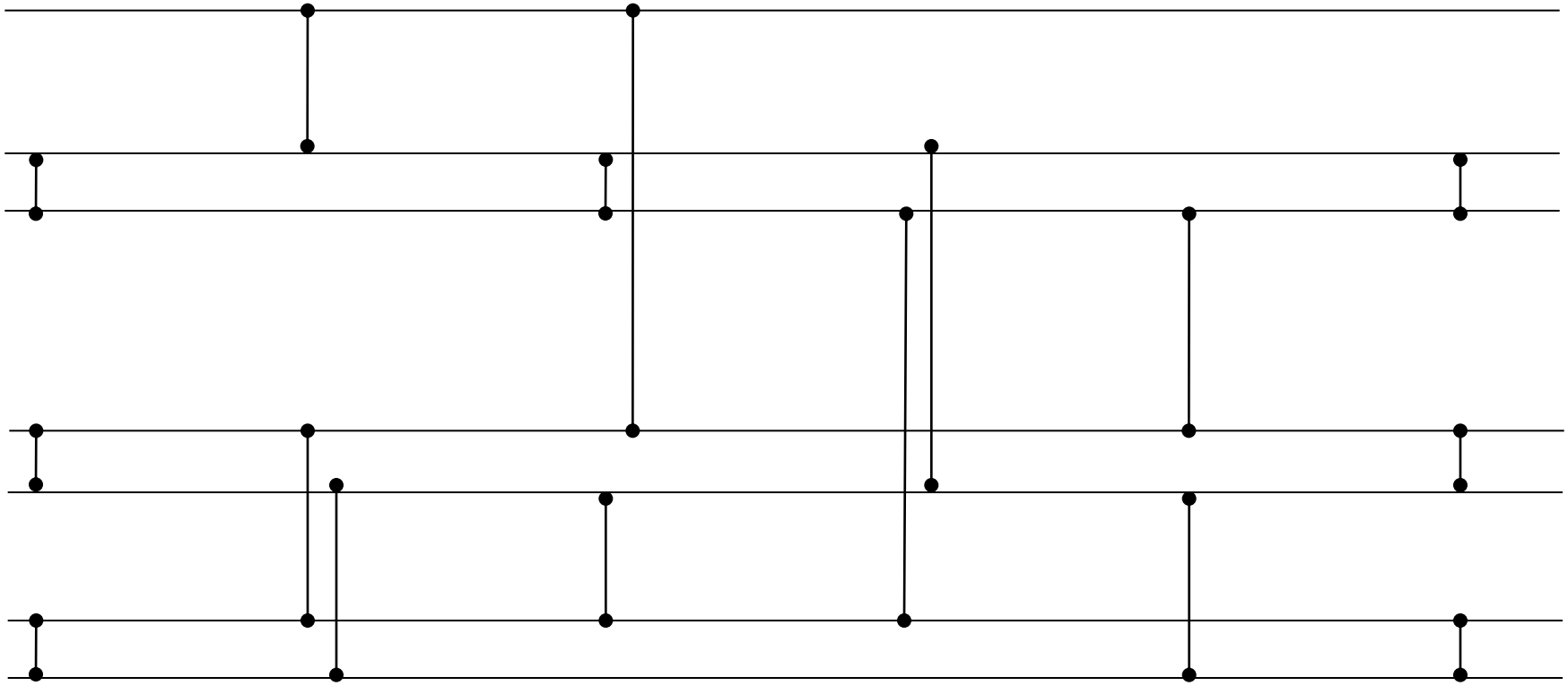
# Замыкающая цепочка компараторов



# Полная сеть сортировки



# Полная сеть сортировки, такты выполнения

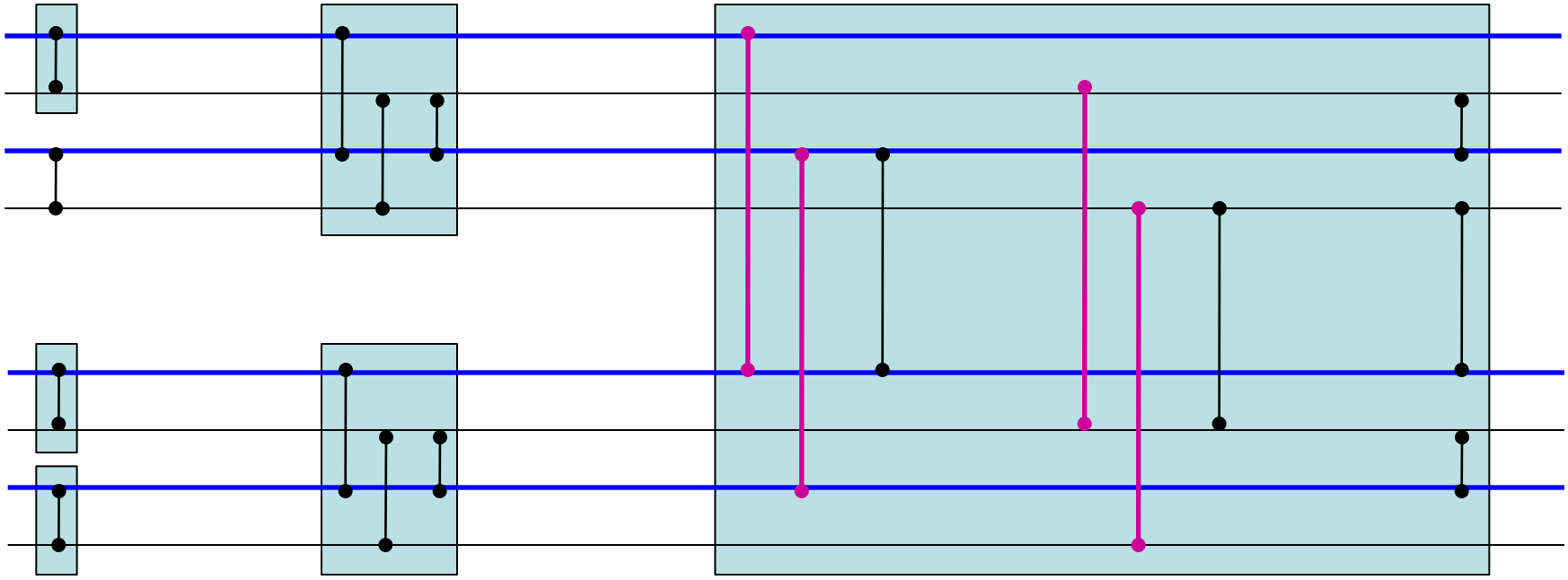


# Сортировка восьми элементов

$$O\left(\frac{n}{p}\left[\log_2\frac{n}{p}+\frac{\lceil\log_2 p\rceil^2}{2}\right]\right)$$

*или*

*n* элементов восемью процессорами



*- сеть четно-нечетного слияния Бетчера*

n=$10^8$

$$E^{\max}(n,p)=\frac{\log_2 n}{\log_2 n + s_p - \log_2 p} \approx \frac{1}{1+\log_n p(\log_2 p - 1)/2}$$

| P | T,сек | E | S | $E^{max}$ | $S^{max}$ | $s_p$ |
|---|---|---|---|---|---|---|
| 1 | 83.51 | 100.00% | 1.00 | 100% | 1.0 | 0 |
| 2 | 46.40 | 90.00% | 1.80 | 100% | 2.0 | 1 |
| 3 | 35.93 | 77.48% | 2.32 | 95% | 2.8 | 3 |
| 4 | 29.68 | 70.35% | 2.81 | 96% | 3.9 | 3 |
| 5 | 24.45 | 68.33% | 3.42 | 91% | 4.5 | 5 |
| 6 | 22.16 | 62.80% | 3.77 | 92% | 5.5 | 5 |
| 7 | 21.82 | 54.67% | 3.83 | 89% | 6.2 | 6 |
| 8 | 19.95 | 52.32% | 4.19 | 90% | 7.2 | 6 |
| 16 | 12.36 | 42.22% | 6.75 | 82% | 13.1 | 10 |
| 27 | 9.32 | 33.20% | 8.97 | 74% | 20.0 | 14 |
| 32 | 7.85 | 33.24% | 10.64 | 73% | 23.3 | 15 |
| 48 | 6.45 | 26.97% | 12.95 | 66% | 31.9 | 19 |
| 64 | 4.92 | 26.53% | 16.98 | 64% | 40.9 | 21 |
| 128 | 3.19 | 20.47% | 26.20 | 56% | 71.5 | 28 |
| 192 | 2.52 | 17.29% | 33.19 | 51% | 98.2 | 33 |
| 256 | 1.99 | 16.41% | 42.02 | 49% | 124.6 | 36 |
| 384 | 1.63 | 13.33% | 51.20 | 49% | 187.0 | 41 |
| 512 | 1.29 | 12.64% | 64.74 | 42% | 217.4 | 45 |
| 640 | 1.21 | 10.78% | 69.02 | 41% | 264.7 | 47 |

$$s_p \approx \frac{\lceil \log_2 p \rceil \left( \lceil \log_2 p \rceil + 1 \right)}{2}$$

# Заключение

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

# Контакты

**Якобовский М.В.**

проф., д.ф.-м.н.,

зав. сектором

«Программного обеспечения многопроцессорных систем и вычислительных сетей»

Института прикладной математики им. М.В.Келдыша Российской академии наук

mail: lira@imamod.ru

web: http://lira.imamod.ru