Quick Sort

- В основе алгоритма лежит идея ведущего элемента, разделяющего элементы на две части left и right. Элементы left меньше ведущего элемента, элементы right больше ведущего элемента.
- Как правило, алгоритм работает рекурсивно: разбивает последовательность на две части в зависимости от ведущего элемента (pivot)
- Выбор ведущего элемента: начальный элемент, среднее число от начального и конечного элемента, средний элемент и т.п.
- При достижении достаточно малого числа элементов, как правило, выполняется сортировка слиянием

Parallel QSort

- Параллельная реализация: обработка левой и правой части возможна одновременно
- Большое число рекурсивных вызовов может существенно снижать эффективность параллельного выполнения
- Типовое решение: использовать параллельную или последовательную обработку в зависимости от уровня рекурсии
- Контроль параллельных запусков:
  - Учет числа элементов
  - Учет глубины рекурсии
  - Учет числа фактически выполняющихся потоков

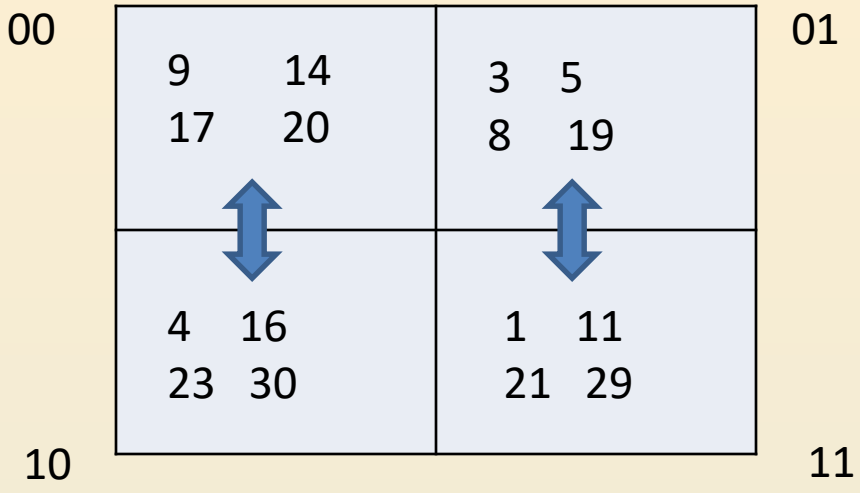
Сортировка Шелла

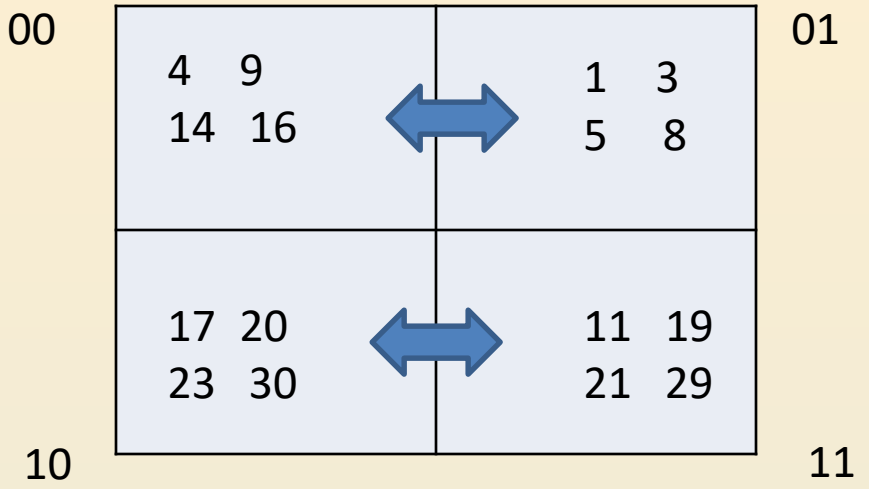
- Идея сортировки Шелла: упорядочивание максимально-удаленных элементов
- Параллельная сортировка Шелла
- Число блоков $q = 2^N$
- Число потоков/процессоров: $p = q / 2$
- Алгоритм:
  - 1 этап: локальная сортировка блоков
  - 2 этап: N итераций merge-split для блоков

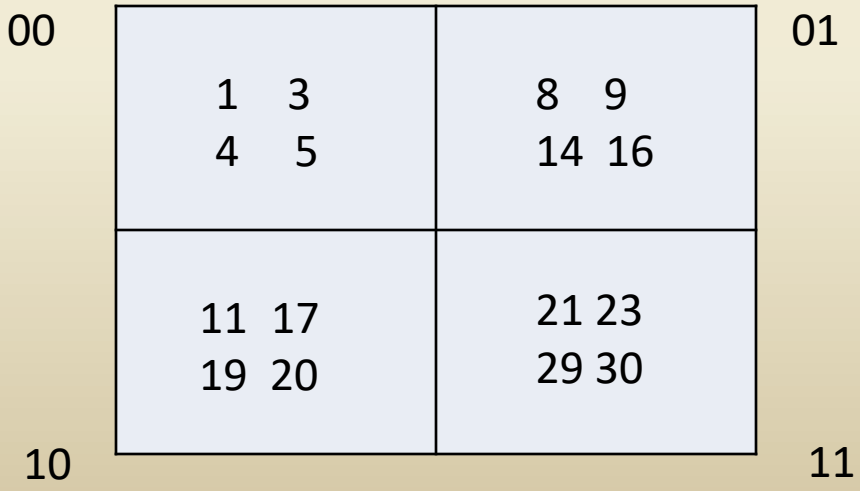
    на каждой i-итерации взаимодействуют блоки, номера которых различаются только в (N-i)-разряде в битовом представлении
  - 3 этап: чет-нечетная сортировка до прекращения изменений, число итераций L = 1.. q

$$T_p = \frac{n}{2p}\log\frac{n}{2p} + \frac{n}{p}\cdot\log 2p + L\cdot n/p$$

14 9 17 20 5 3 8 19 4 16 23 30 29 11 1 21







Быстрая блочная сортировка

- Элементы разбиваются на блоки
- Число блоков: $q = 2^N$
- Число потоков/процессоров: $p = q / 2$

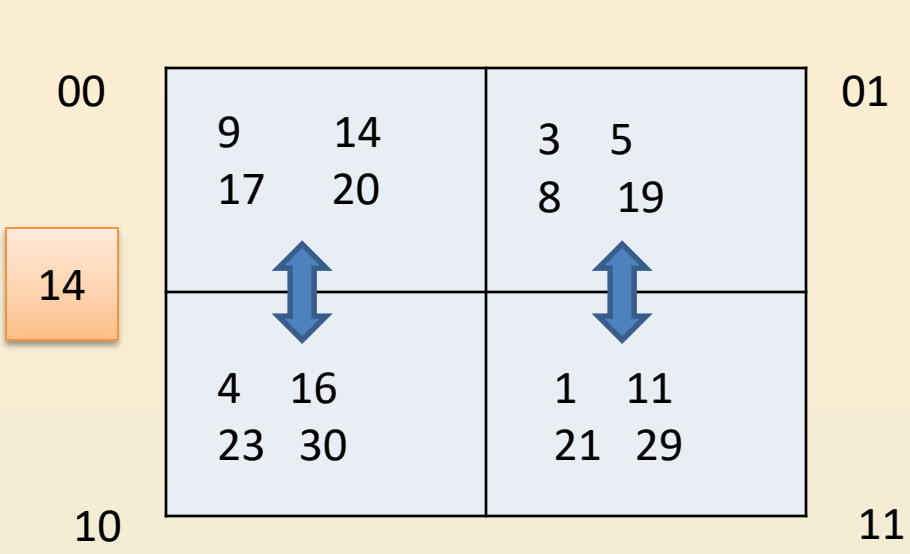
Выполняется N итераций взаимодействия пар блоков

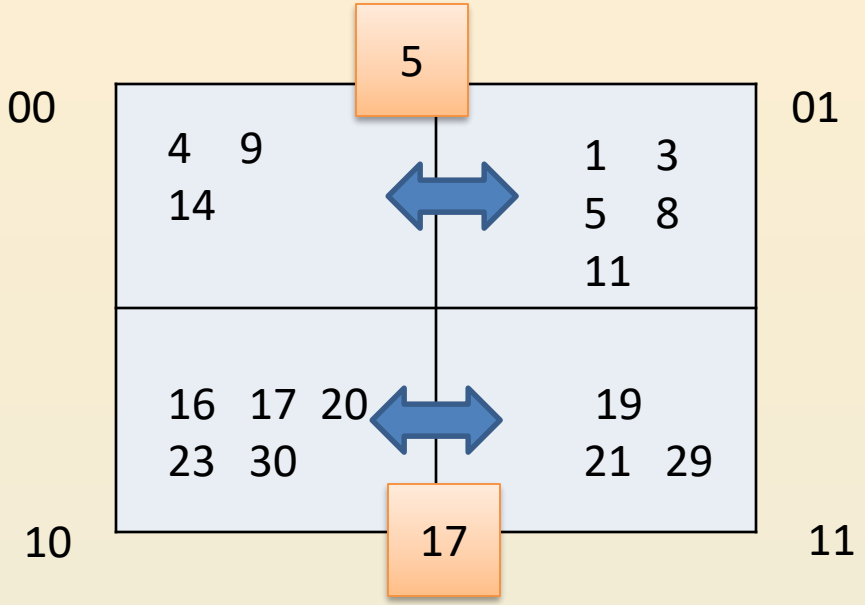
На каждой i-итерации взаимодействуют блоки, номера которых различаются только в (N-i)-разряде в битовом представлении

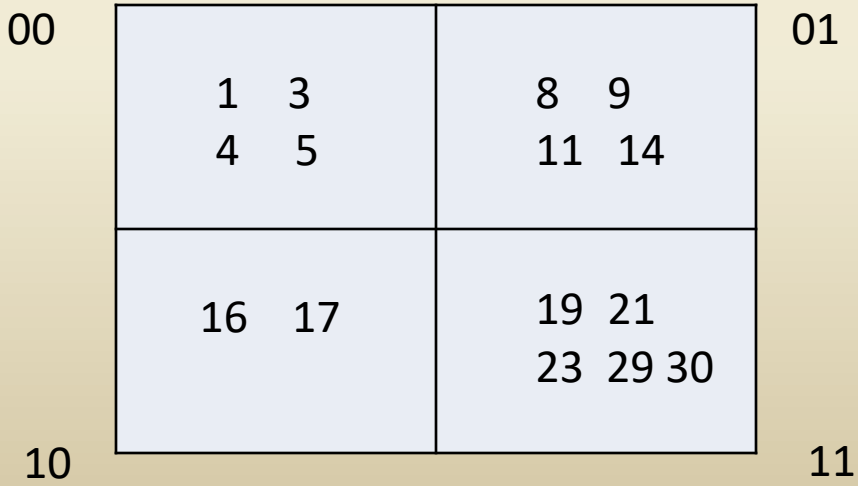
На каждой i-итерации выбирается 2^i ведущих элемента, по одному элементу для всех пар, номера блоков которых имеют одинаковые биты в разрядах [N+1; N-i+1);

При взаимодействии блок с меньшим номером получает все элементы, меньшие ведущего элемента; блок с большим номером получает все элементы, большие ведущего элемента.

14 9 17 20 5 3 8 19 4 16 23 30 29 11 1 21





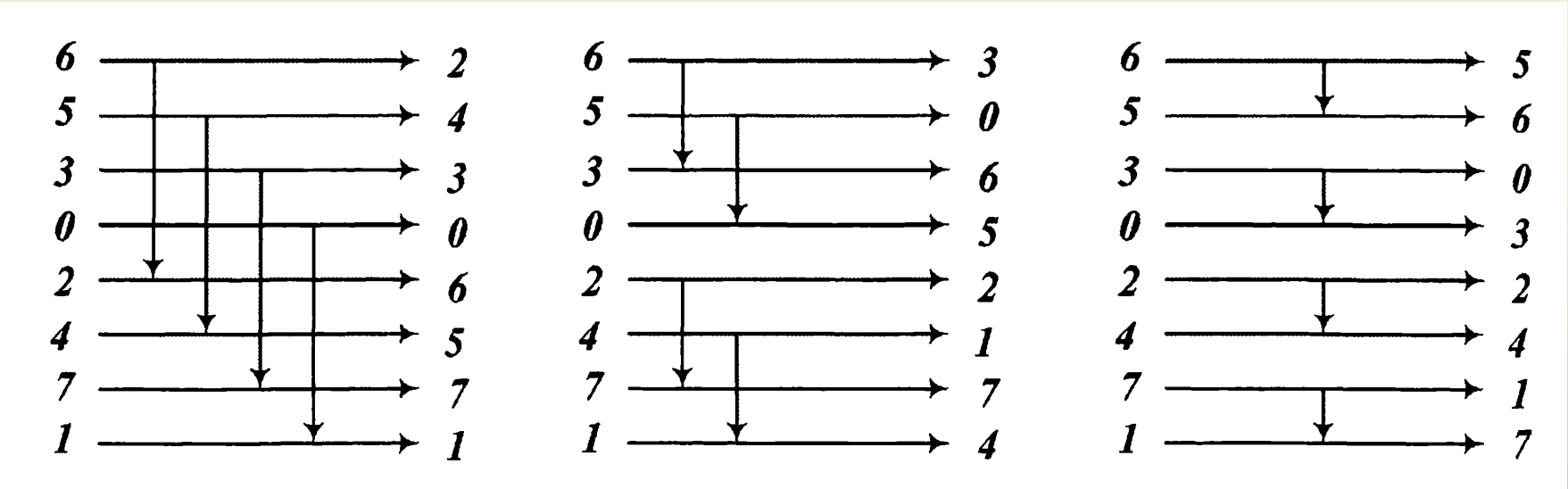


Bitonic Sort

- Битоническая сортировка основана на свойствах битонической последовательности

Битонический оператор

- Битонический оператор = полуочиститель (half-cleaner)
- Битонический оператор Bk сравнивает и упорядочивает элементы пары (aj, aj+k/2)
- Примеры битонических операторов B8, B4, B2



Битоническая последовательность

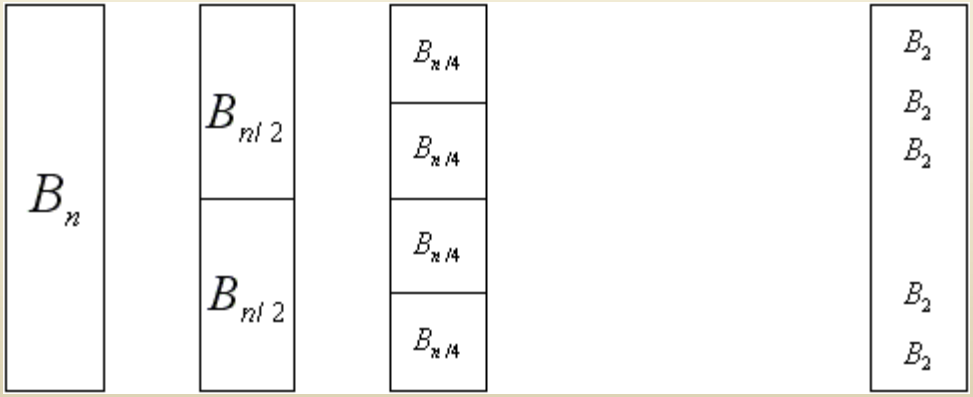
- Последовательность a_0, a_1, a_2, .. a_{n-1} – битоническая, если она состоит из двух монотонных частей (возрастающей и убывающей) или может быть получена из такой последовательности с помощью сдвига.
- Примеры : 1, 3, 5, 8, 7, 6, 4, 2; 5, 8, 7, 6, 4, 2, 1, 3
- Применение оператора Bn к битонической последовательности приводит к следующему:
  - Обе ее половины также будут битоническими
  - Любой элемент первой половины будет не больше любого элемента второй половины
  - ?Хотя бы одна из половин является монотонной

(1, 3, 5, 8), (7, 6, 4, 2) -> (1, 3, 4, 2), (7, 6, 5, 8)

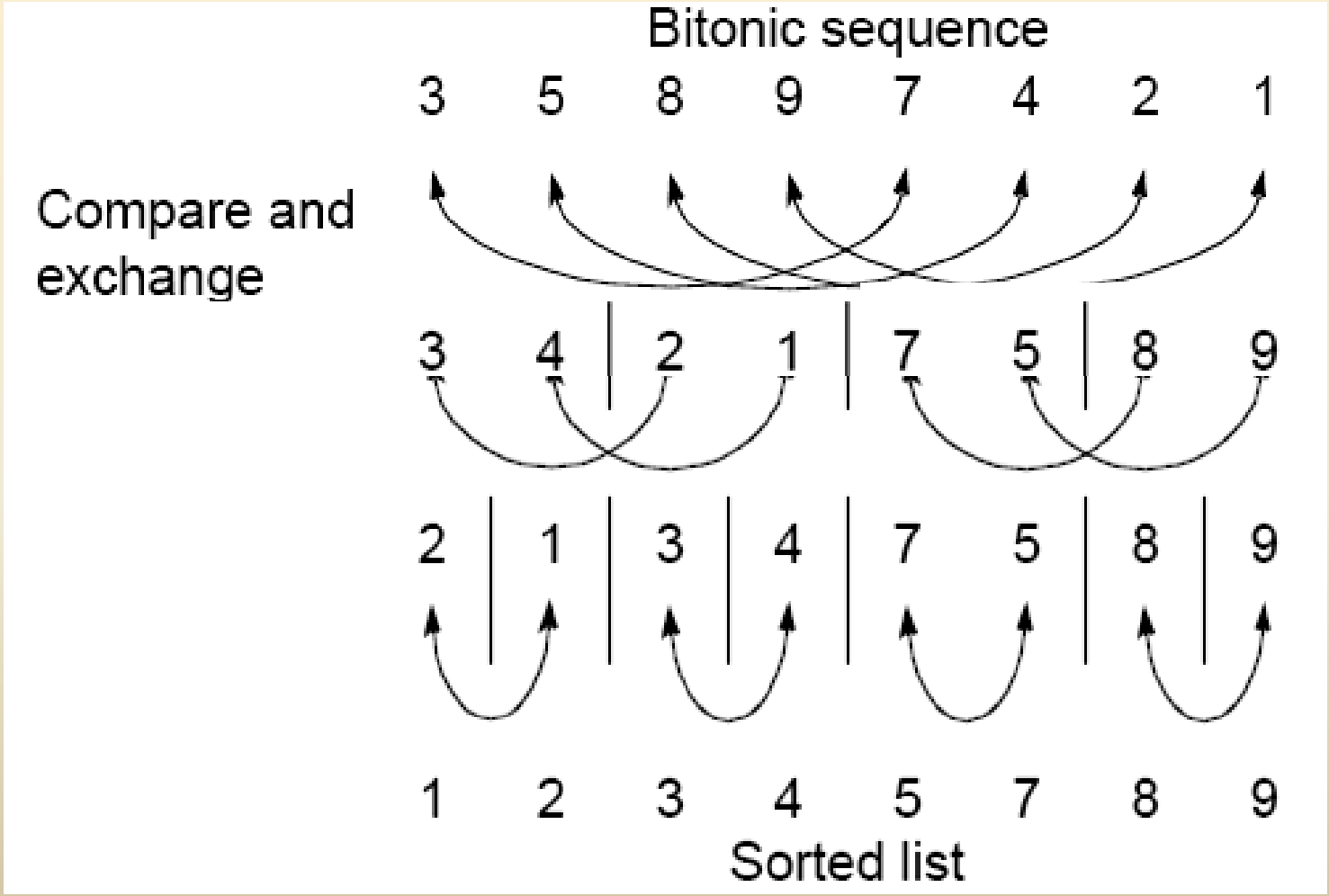
(5, 8, 7, 6), (4, 2, 1, 3) -> (4, 2, 1, 3), (5, 8, 7, 6)

Битоническое слияние

- Последовательное применение к битонической последовательности операторов B_n, $B_{n/2}$, $B_{n/4}$, .. B_2 приводит к упорядоченной последовательности
- Поочередное выполнение операторов B_n, $B_{n/2}$, $B_{n/4}$, .. B_2 называется битоническим слиянием M_n

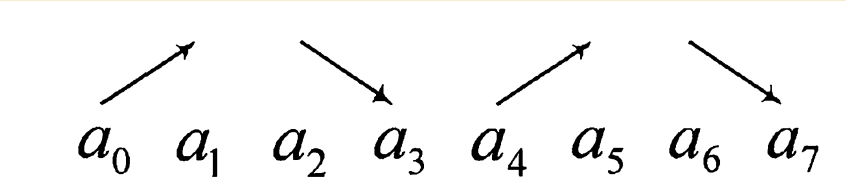


Битоническое слияние



Получение битонической последовательности

- Применяем B2 так, чтобы в соседних парах был разный порядок



- Каждая четверка образует битоническую последовательность
- Применяем M4 с разным порядком и получаем битонические последовательности из 8 элементов

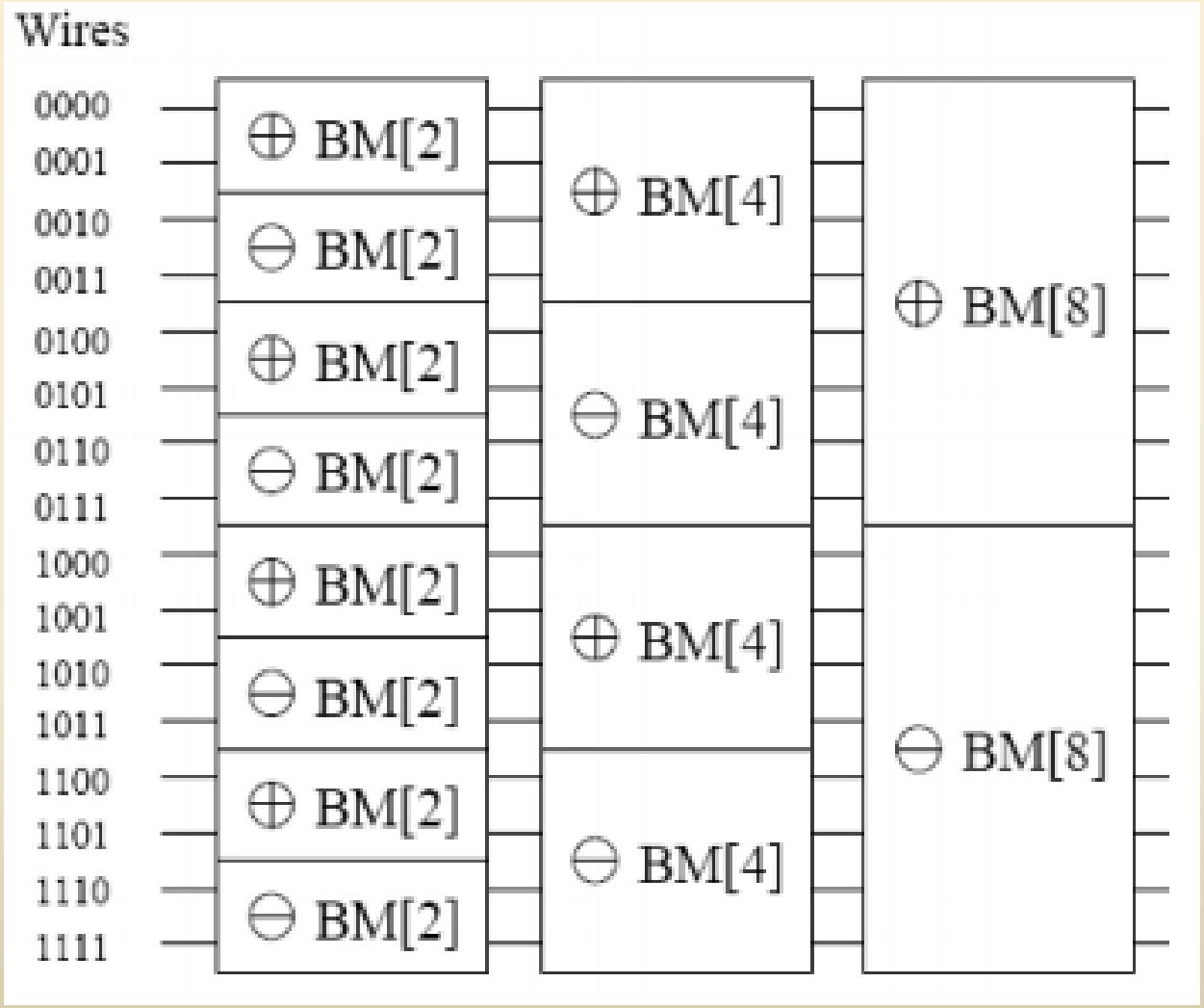
  Исходная четверка: 10, 20, 5, 9

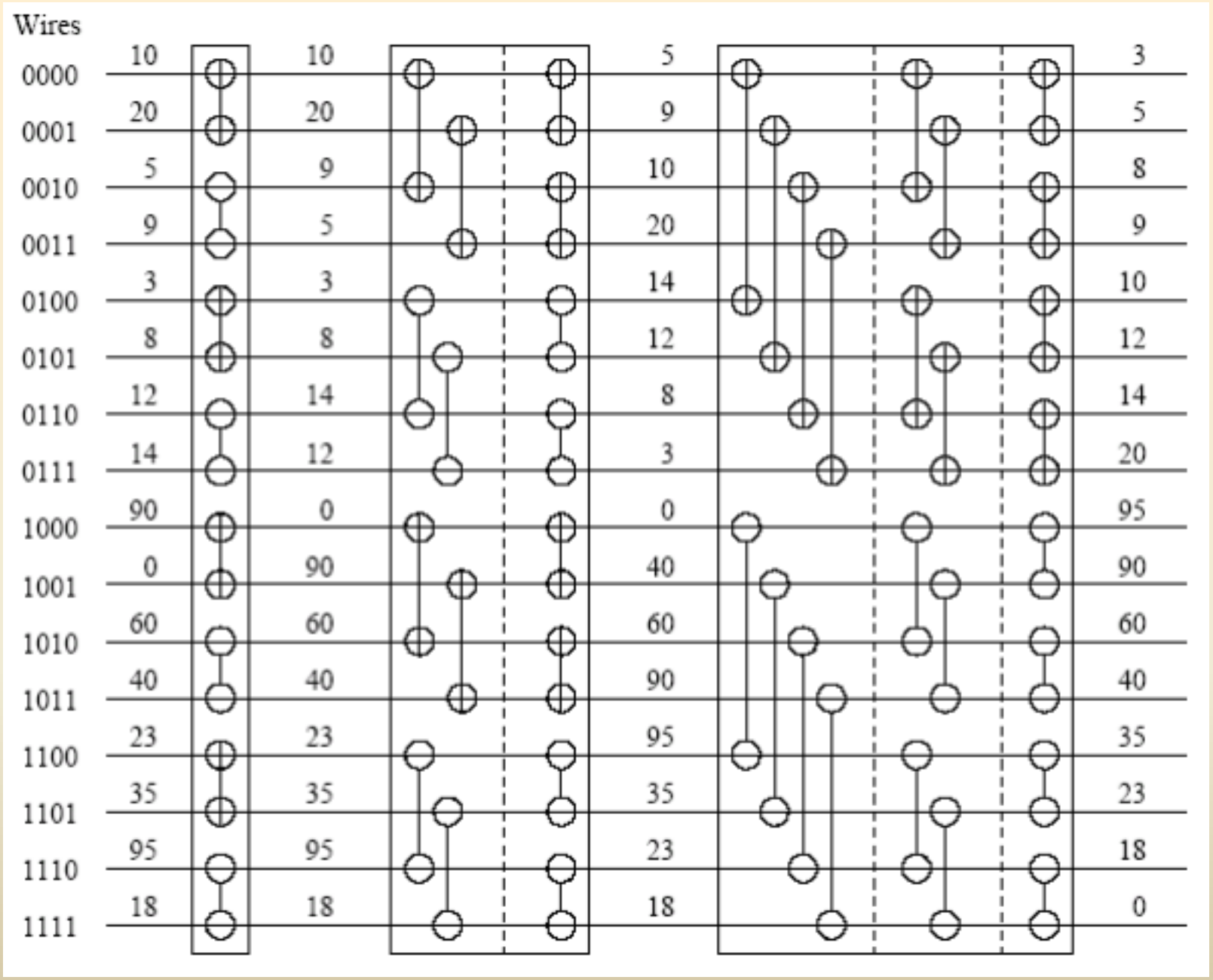
  B2(+-) : 10, 20, 9, 5

  B4(+) : 9, 5, 10, 20

  B2(+) : 5, 9, 10, 20

Получение битонической последовательности




Wires
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
10 20 5 9 3 8 12 14 90 0 60 40 23 35 95 18
10 20 9 5 3 8 14 12 0 90 60 40 23 35 95 18
5 9 10 20 14 12 8 3 0 40 60 90 95 35 23 18
3 5 8 9 10 12 14 20 95 90 60 40 35 23 18 0

Общая схема битонической сортировки

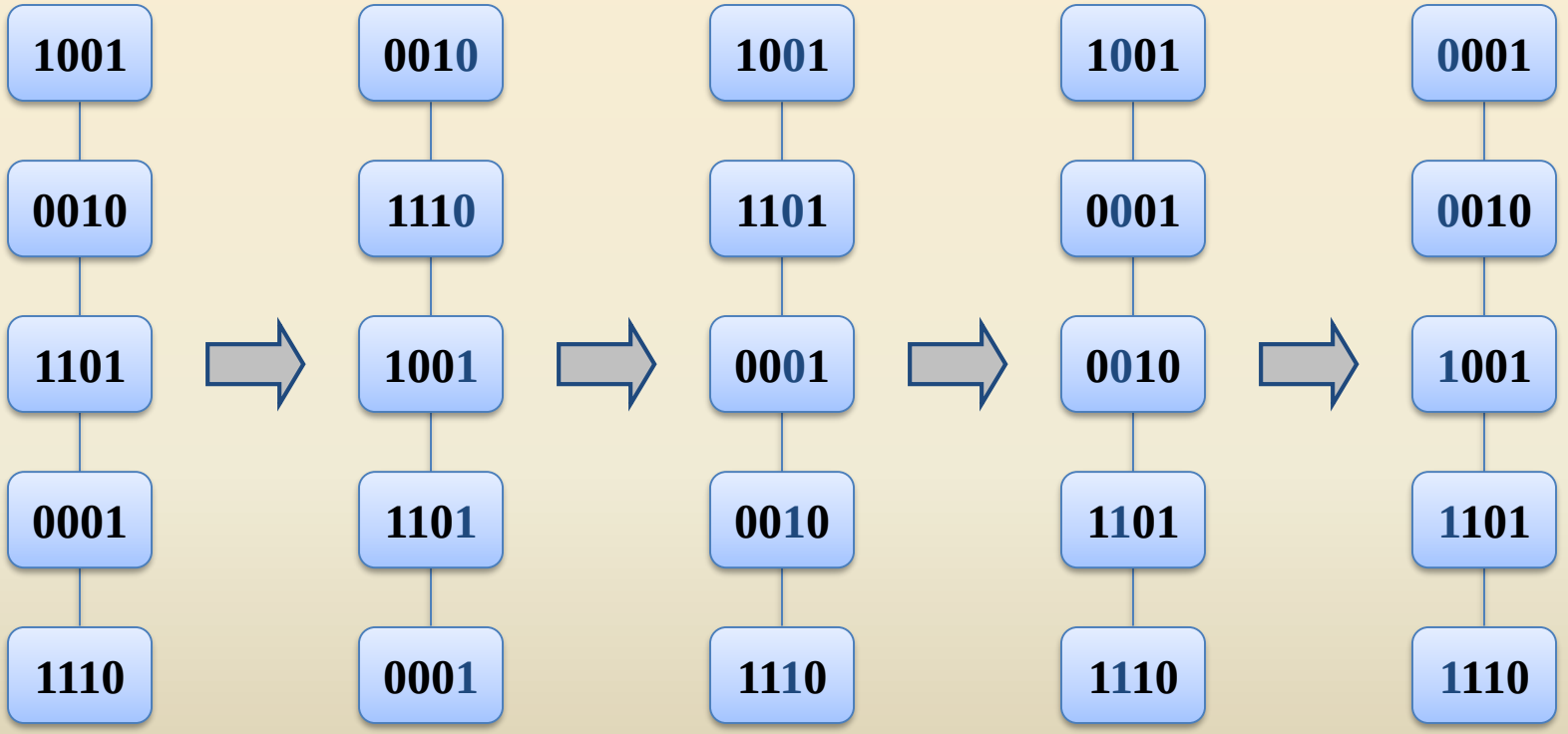
- Получить битоническую последовательность, выполняя битонические слияния: $M_2 + M_4 + M_8 + .. + M_{n/2}$
- Отсортировать битоническую последовательность, выполняя операторы B_n, $B_{n/2}$, $B_{n/4}$, .. B_2
- В параллельной реализации возможно одновременное выполнение операторов B_i, M_i над разными элементами

Radix Sort

- Поразрядная сортировка
- Вместо сортировки элементов выполняется несколько сортировок отдельно по каждому из битов
- Если сперва отсортировать массив по 0-биту, затем по 1-биту и т. д., заканчивая сортировкой по 31-биту, то в результате мы получим полностью отсортированный по всей совокупности бит массив.

Поразрядная сортировка

- Сортировка 4-битовых целых чисел



Вычисление позиции элемента в Radix

- Сортировка массива по каждому из битов фактически означает некоторую перестановку элементов местами.
- Пусть необходимо отсортировать последовательность по k-му биту. Тогда определим массив b_0, b_1, ..., b_{n-1} следующим образом: $b_i = (a_i >> k) \& 1$.
- Далее применим к этому массиву операцию scan, в результате которой мы получим массив частичных сумм s_0, s_1, s_2, .. s_{n-1} и полную сумму s_n всех элементов массива.
- Тогда новое положение элемента a_i будет равно $i - s_i$, если $b_i = 0$ и $s_i + N_z$ в противном случае, где N_z - это число нулей в массиве b_0, b_1, ..., b_{n-1} ($N_z = n - s_n$).

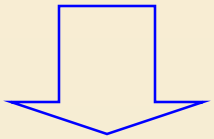
LSD-сортировка (radix)

- Поразрядная сортировка по младшим разрядам (least significant digit)
- Элементы перебираются по порядку и группируются по самому младшему разряду (сначала все, заканчивающиеся на 0, затем заканчивающиеся на 1, …, заканчивающиеся на 9). Возникает новая последовательность. Затем группируются по следующему разряду с конца, затем по следующему и т.д. пока не будут перебраны все разряды, от младших к старшим.

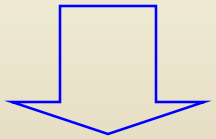
872	488	912	7	60	575	943	286	353	557

LSD-сортировка. 1 этап

12	58	37	64	52	36	99	63	18	9	20	88	47



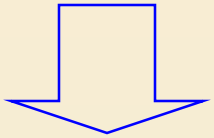
20		12 52	63	64		36	37 47	58 18 88	9 99



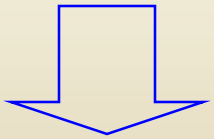
20	12	52	63	64	36	37	47	58	18	88	9	99

LSD-сортировка. 2 этап

20	12	52	63	64	36	37	47	58	18	88	9	99



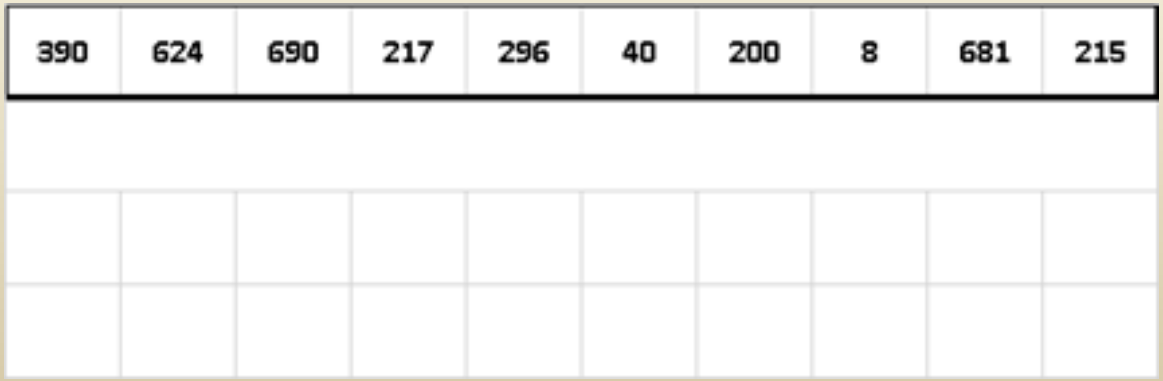
9	12 18	20	36 37	47	52 58	63 64		88	99



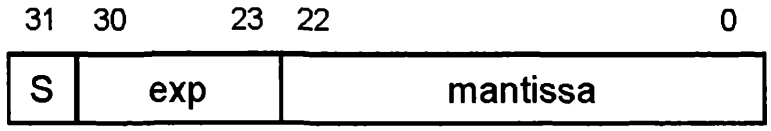
9	12	18	20	36	37	47	52	58	63	64	88	99

MSD-сортировка (radix)

- Поразрядная сортировка по старшим разрядам (most significant digit)
- Элементы перегруппировываются по определённому разряду (сначала по самому старшему). Затем разбиваются на подгруппы в зависимости от значения этого разряда: равного 0, равного 1, равного 2, …, равного 9. Каждая подгруппа обрабатывается отдельно, в ней к следующему разряду рекурсивно применяется radix sort.



Поразрядная сортировка вещественных чисел



$$f = (-1)^S \times 2^{\exp-127} \times 1.mantissa$$

В случае положительных 32-битовых float-величин, старший (знаковый) бит у них совпадает и равен нулю; следующими по значимости являются 8-бит экспоненты: если у одного числа соответствующее 8-битовое значение больше, то и данное число больше (так как мантисса умножается на $2^{exp-127}$). При совпадении экспонент идет сравнение мантисс. Таким образом, положительные 32-битовые floatвеличины можно просто сортировать как 32-битовые беззнаковые целые.

Обработка отрицательных float-величин

- Предварительно выполняется преобразование:
  - Если число положительное, то у него выставляется старший бит
  - Если число отрицательное, то все его биты инвертируются