3. ANALÝZA ALGORITMŮ

Upd 2005

Cor 2005

Upd pre 2007/8

3. ANALÝZA ALGORITMŮ

- **Počet procesorů**
  - (p) potřebných k řešení úlohy
    v závislosti na velikosti instance n

$p(n) = \{1, c, \log n, n, n \log n, n^2, n^r, r^n\}$

sekvenční alg.   konstantní alg.   rozumný počet CPU   nerozumný počet CPU

- **Čas** řešení potřebný k řešení úlohy v jednotkách (krocích) t(n)
- **Cena** paralelního řešení: c(n) = p(n) . t(n)
  - Algoritmus s optimální cenou: $c(n)_{optim} = t_{seq}(n)$
- **Zrychlení** x **Efektivnost**
  - Zrychlení $t_{seq}(n) / t(n)$
  - Efektivnost $t_{seq}(n) / c(n)$
    - <1 neoptimální (přidá se režie)
    - » =1 optimální
    - » >1 ?

4. ŘAZENÍ

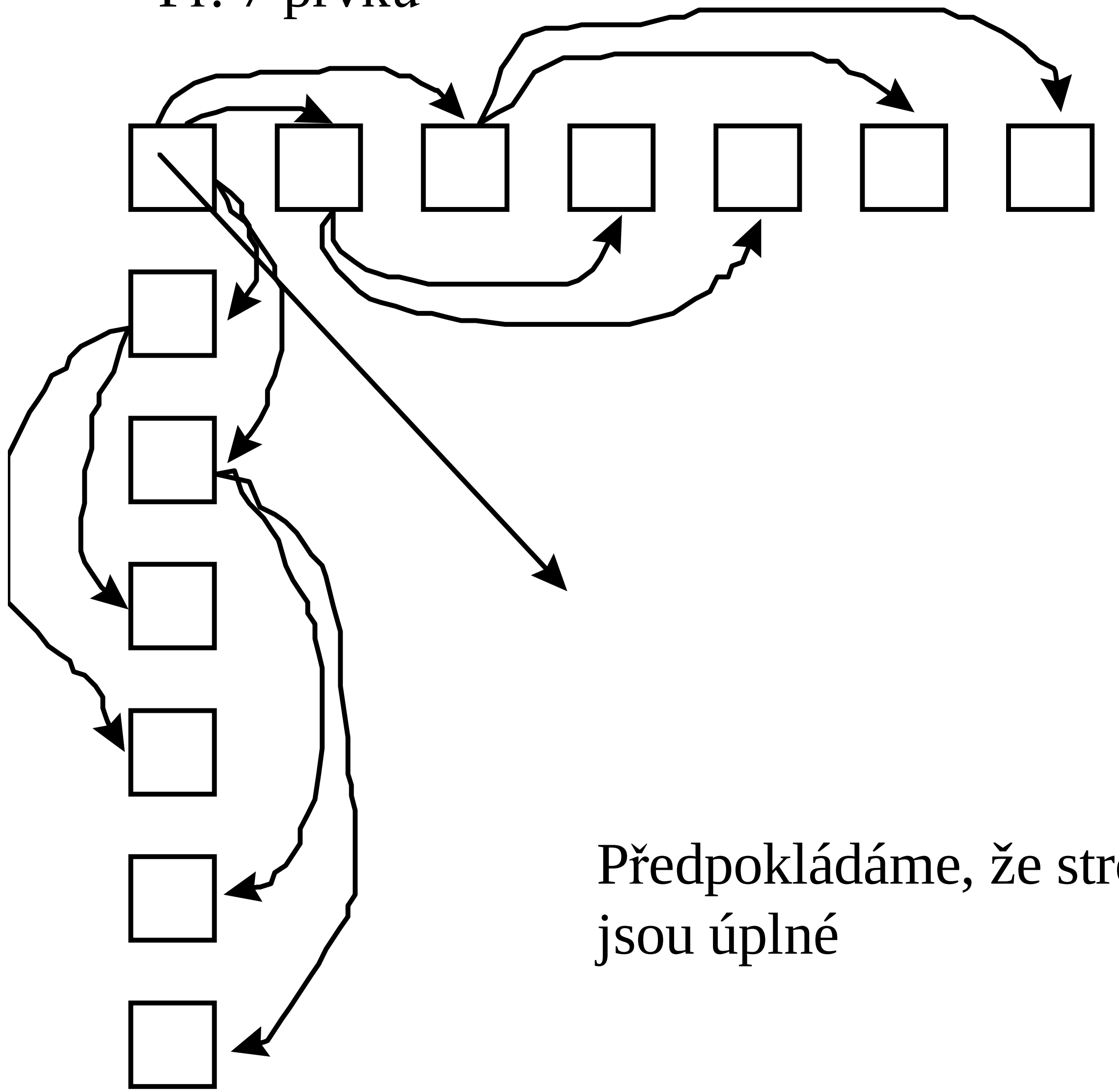
- Máme posloupnost $X = \{x_1, \dots, x_n\}$ s <u>n</u> prvky a lineární uspořádání >
- Cílem je vytvořit z prvků x_i novou posloupnost $Y = \{y_1, \dots, y_n\}$, kde platí $y_i < y_{i+1}$, $i = 1, \dots N-1$

- V X nejsou žádné dva prvky rovny
- Optimální sekvenční algoritmus - *platí pro řadící algoritmy založené na porovnávání prvku*
  - $p(n) = 1$
  - $t(n) = O(n.\log n)$
  - $c(n) = O(n.\log n)$

4.1 Enumeration sort

- Princip: správná pozice každého prvku ve výstupní seřazené posloupnosti je dána počtem prvků, které jsou menší než tento prvek
- Topologie:
  - n^2 procesorů je uspořádáno do mřížky n x n
  - Procesory v každém řádku i jsou propojeny do binárního stromu, kde P(i, j) je propojen s P(i, 2j) a P(i, 2j + 1)
  - Procesory v každém sloupci j jsou propojeny do binárního stromu, kde P(i, j) je propojen s P(2i, j) a P(2i + 1, j)
  - 10 prvků $\Rightarrow$ 100 procesorů
- Ideální algoritmus pro paralelní zpracování
- Vlastnosti: každý procesor
  - Může uložit dva prvky do svých registrů A a B
  - Může porovnat A a B a uložit výsledek do registru RANK
  - Pomocí stromového propojení může předat obsah kteréhokoli registru jinému procesoru
  - Může přičítat k registru RANK

Př. 7 prvků



Předpokládáme, že stromy jsou úplné

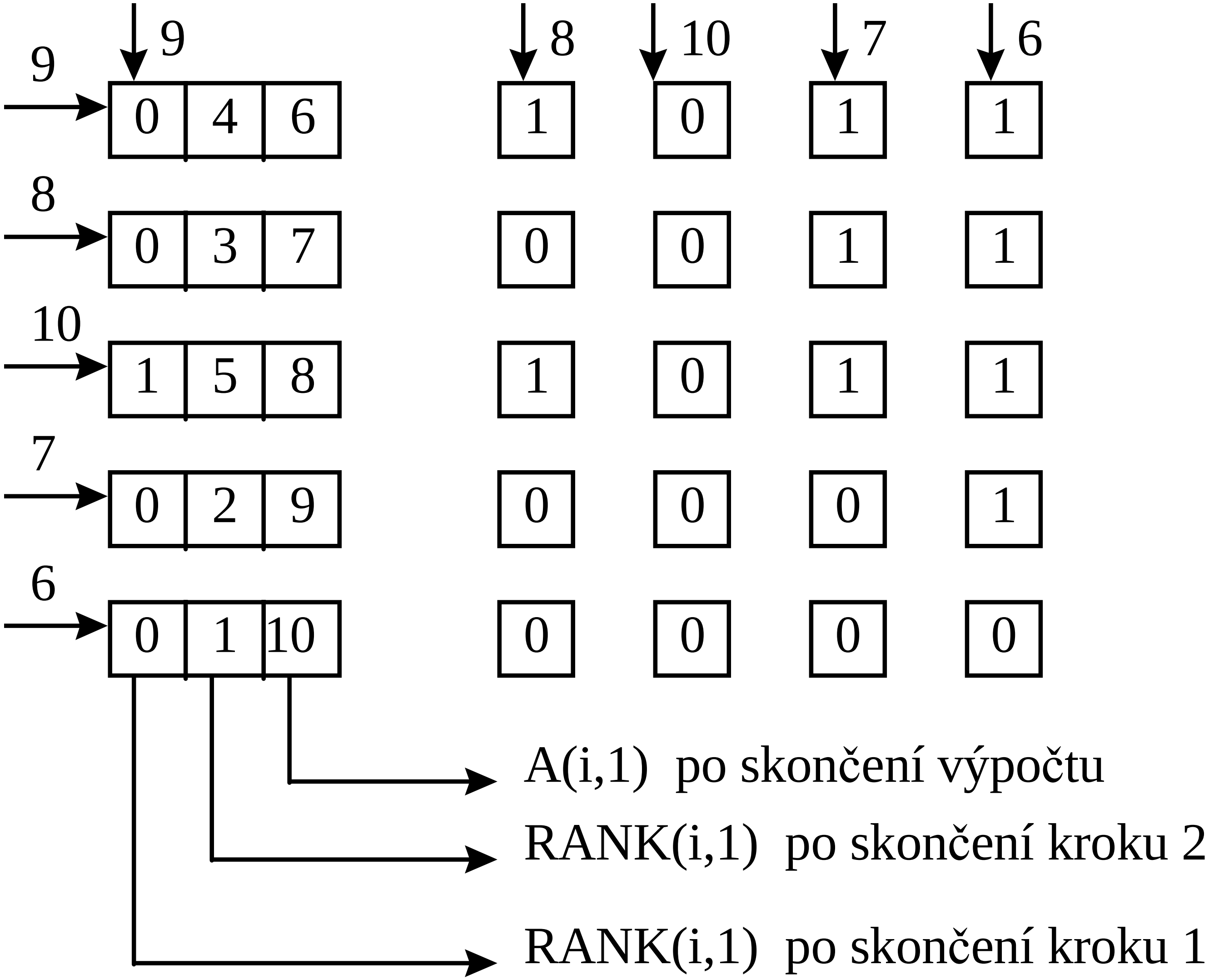
Algoritmus

- 1) Každý prvek je porovnán se všemi ostatními pomocí jedné řady procesorů
- 2) Správná pozice prvku je RANK(x_i) = 1 + počet menších prvků
- 3) Každý prvek je zadán na správné místo

```
1) for i=1 to n do in parallel
        1.1) každý procesor P(i, j) v řadě i získá xi a xj
                 (i, j = 1...n) a uloží je do A(i, j) a B(i, j)

        1.2) if B(i, j) < A(i, j) then RANK(i, j) = 1
                                   else RANK(i, j) = 0
             endif
   endfor
2) for i = 1 to n do in parallel
        2.1) obsah registrů RANK všech procesorů v řadě i je sečten a
         uložen do RANK(i, 1)
        2.2) P(i, 1) spočte RANK(xi) jako RANK(i, 1) += 1
   endfor
3) for i=1 to n do in parallel
        if  RANK(i, 1)=j then xi je přesunuto z A(i, j) do A(j, 1)
        endif
   endfor
```

Př. x = {9, 8, 10, 7, 6}, u prvního sloupce pro všechny registry, u ostatních jen RANK



- Analýza kroku 1
  - Prvek x_i musí být rozeslán všem procesorům v řadě i a ve sloupci j. Příklad pro řadu i.

```
Procedure PROPAGATE(xi)
(1) A(i, 1) = xi
(2) for k = 1 to ((log n) - 1) do
     for j = 2^(k-1) to 2^k-1 do in parallel        -počet procesorů
            A(i, 2j) = A(i, j)
            A(i, 2j + 1) = A(i, j)
     endfor
    endfor
```

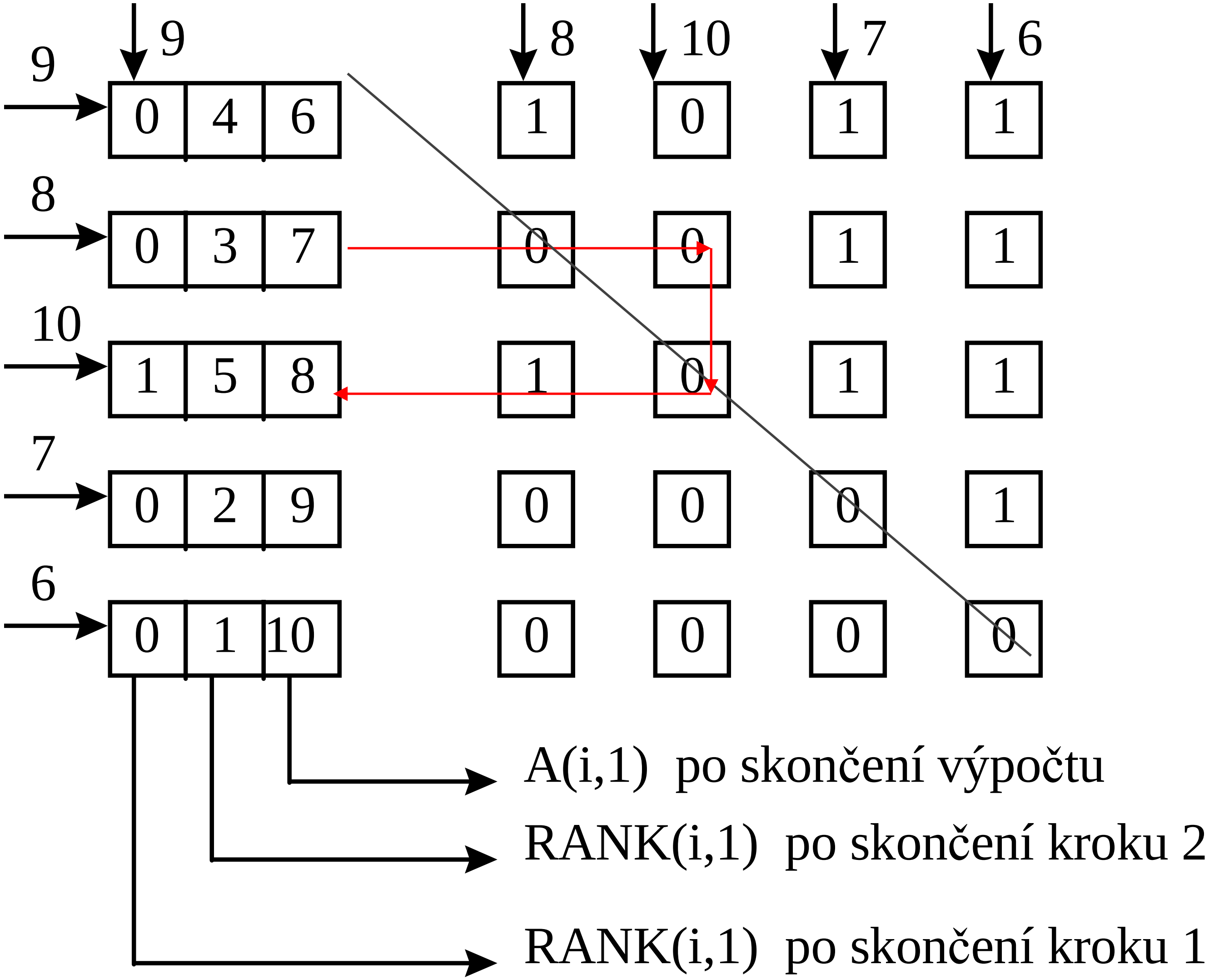
- Tato procedura se složitostí O(log n) je prováděna paralelně pro všechny řady. Podobná procedura se stejnou složitostí slouží pro šíření ve sloupci. Porovnání A a B je v konstantním čase. Složitost kroku 1 je O(log n).

- Analýza kroku 2

```
Procedure SUM(i)
   for k = ((log n) - 1) downto 1 do
     for j = 2^(k-1) to 2^k-1 do in parallel                 - počet procesorů
       RANK(i, j) += RANK(i, 2j) + RANK(i, 2j+1)
                                           sčítání na stromové struktuře
     endfor
   endfor
```

- Což lze provést se složitostí O(log n)
- Analýza kroku 3
  - Procesor P(i, j) zašle x_i do P(RANK(x_i), 1)
    - » 1) P(i, 1) pomocí stromu předá RANK(i, 1) = j do P(i, j)
    - » 2) P(i, i) pomocí stromu ve sloupci i předá A(i, i), t.j. x_i do P(j, i)
    - » 3) P(j, i) předá stromem v řadě j hodnotu x_i do P(j, 1)
  - Každý z těchto kroků má stejnou složitost jako PROPAGATE.
- Krok 3 má stejnou složitost O(log n)

Př. x = {9, 8, 10, 7, 6}, u prvního sloupce pro všechny registry, u ostatních jen RANK



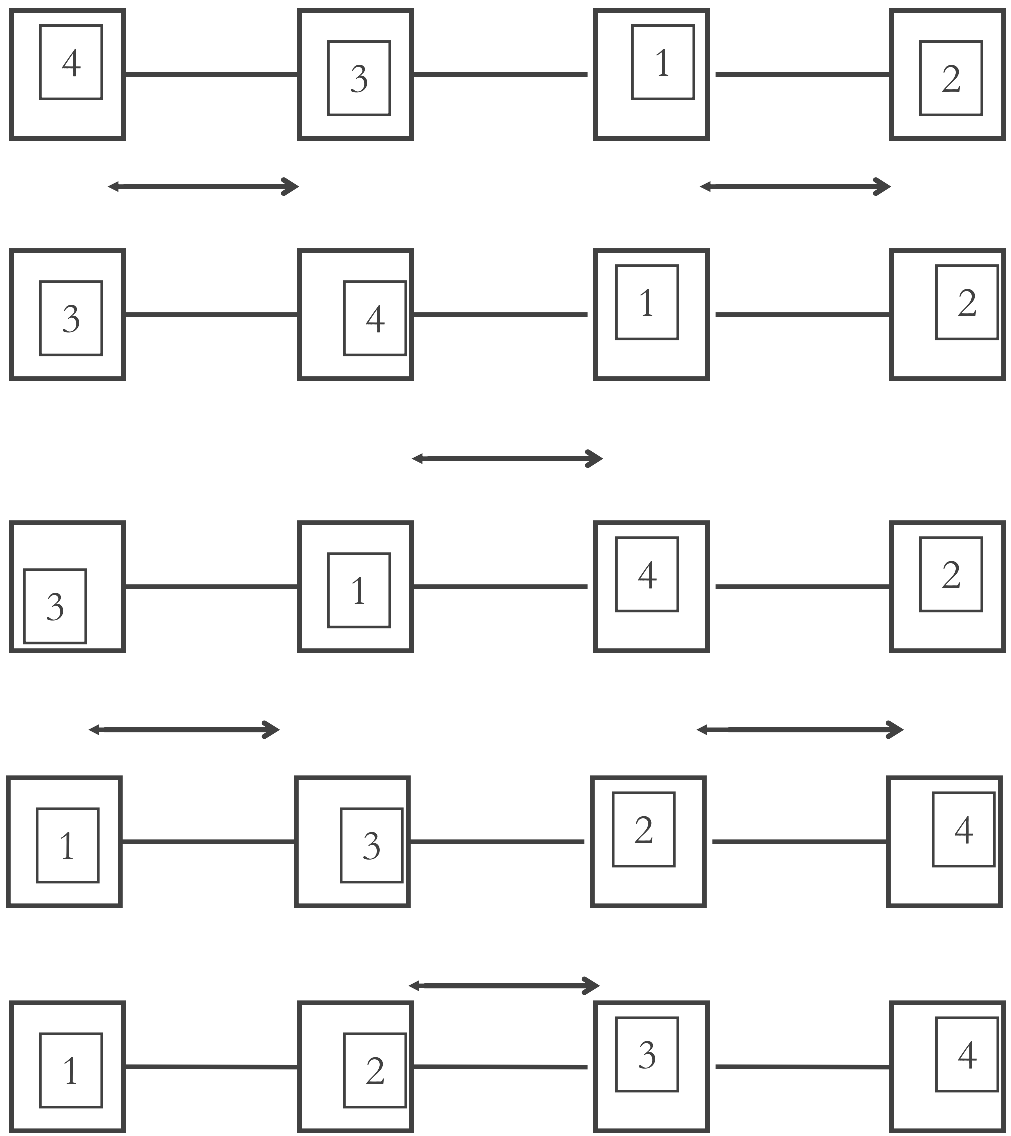
- <u>Analýza</u>
  - $t(n) = O(\log n)$ $p(n) = n^2$
  - $c(n) = O(n^2 . \log n)$ což není optimální

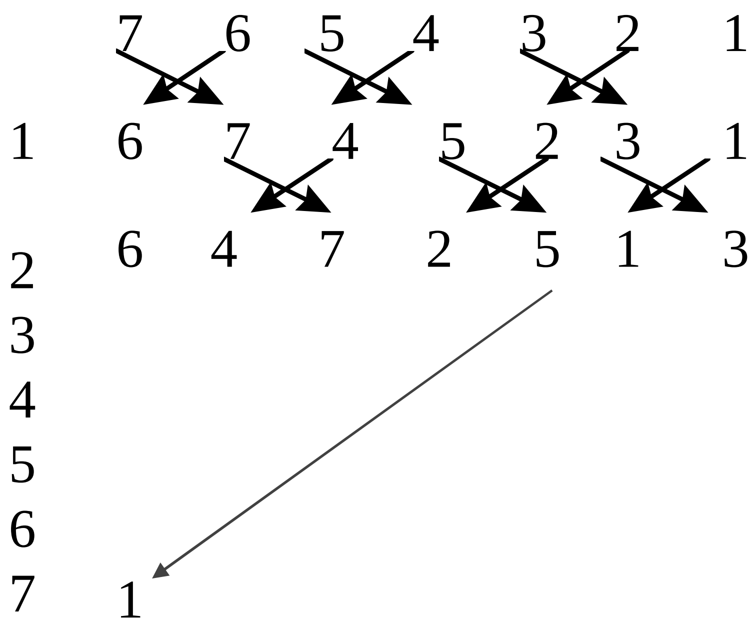
- <u>Diskuse</u>
  - Algoritmus je extrémně rychlý $O(\log n)$, což znamená zrychlení $O(n)$ krát oproti optimálnímu sekvenčnímu algoritmu.
  - Žádný paralelní algoritmus pro rozumný výpočetní model není rychlejší, bez ohledu na počet procesorů
  - Spotřebovává mnoho procesorů - n^2 je na hranici přijatelnosti
  - Vstupní posloupnost nesmí obsahovat stejné prvky (navrhněte úpravu)

Odd-even transposition sort

- Lineární pole n procesorů $p(n) = n$
- Na počátku každý procesor p_i obsahuje jednu z řazených hodnot y_i
- V prvním kroku se každý lichý procesor p_i spojí se svým sousedem p_{i+1} a porovnají své hodnoty je-li $y_i > y_{i+1}$, procesory vymění své hodnoty
- V druhém kroku se každý sudý procesor ...totéž…
- Po n krocích (maximálně) jsou hodnoty seřazeny

```
Algoritmus:
for k = 1 to ⌈n/2⌉ do
  for i = 1, 3, ..., 2.(n/2)-1 do in parallel
    if y_i > y_i+1 then y_i ↔ y_i+1 endif
  endfor
  for i = 2, 4, ..., 2.((n-1)/2) do in parallel
    if y_i > y_i+1 then y_i ↔ y_i+1 endif
  endfor
endfor
```


4
3
1
2
3
4
1
2
3
1
4
2
1
3
2
4
1
2
3
4

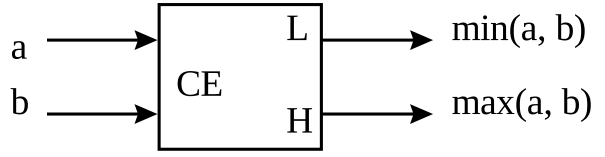


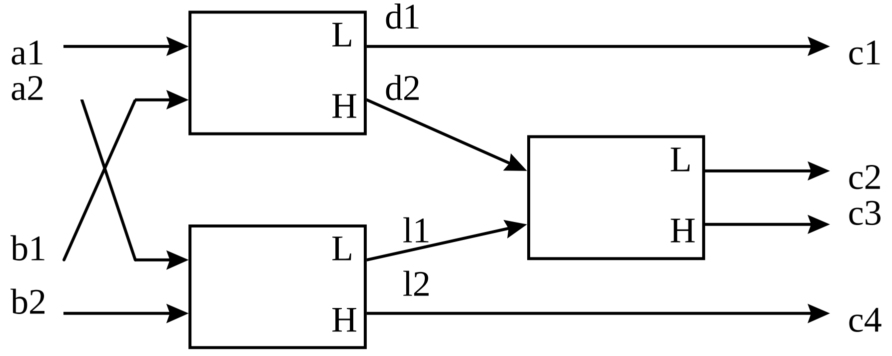
Analýza

- Každý z kroků (1) a (2) provádí jedno porovnání a dva přenosy - konstantní čas
- Složitost: t(n) = O(n)
- Cena: c(n) = t(n) . p(n) = O(n) . n = O(n^2) *což není optimální*
- Algoritmus má časovou složitost t(n) = O(n), což je to nejlepší, čeho lze při lineární topologii dosáhnout

Odd-even merge sort

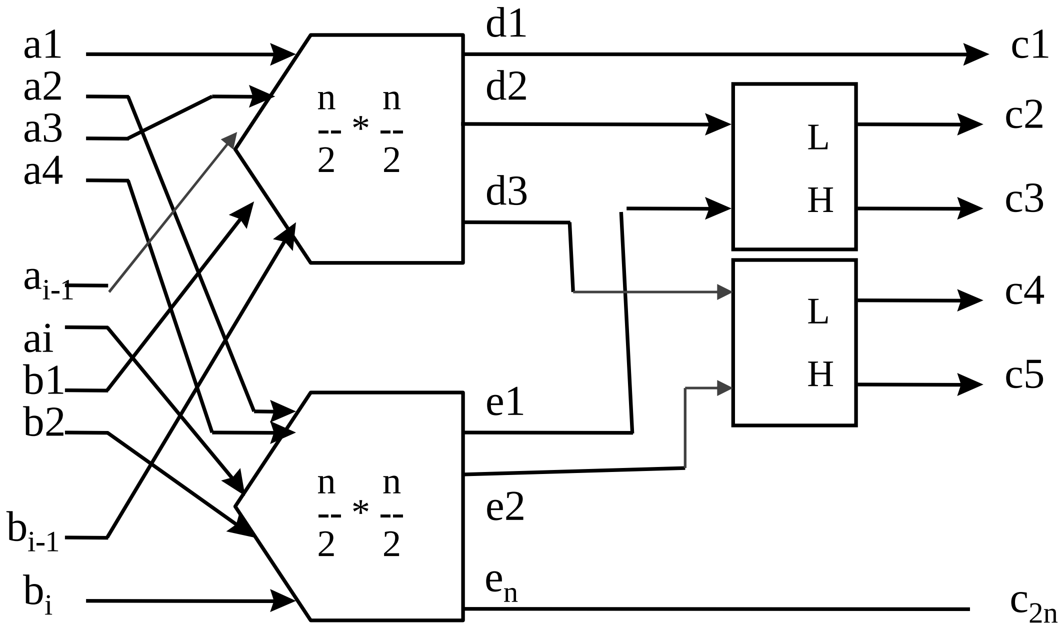
- Řadí se speciální sítí procesorů
  - Každý procesor má dva vstupní a dva výstupní kanály
  - Každý procesor umí porovnat hodnoty na svých vstupech, menší dá na výstup L(low), a větší dá výstup H (high)
- Síť 1x1 2x2





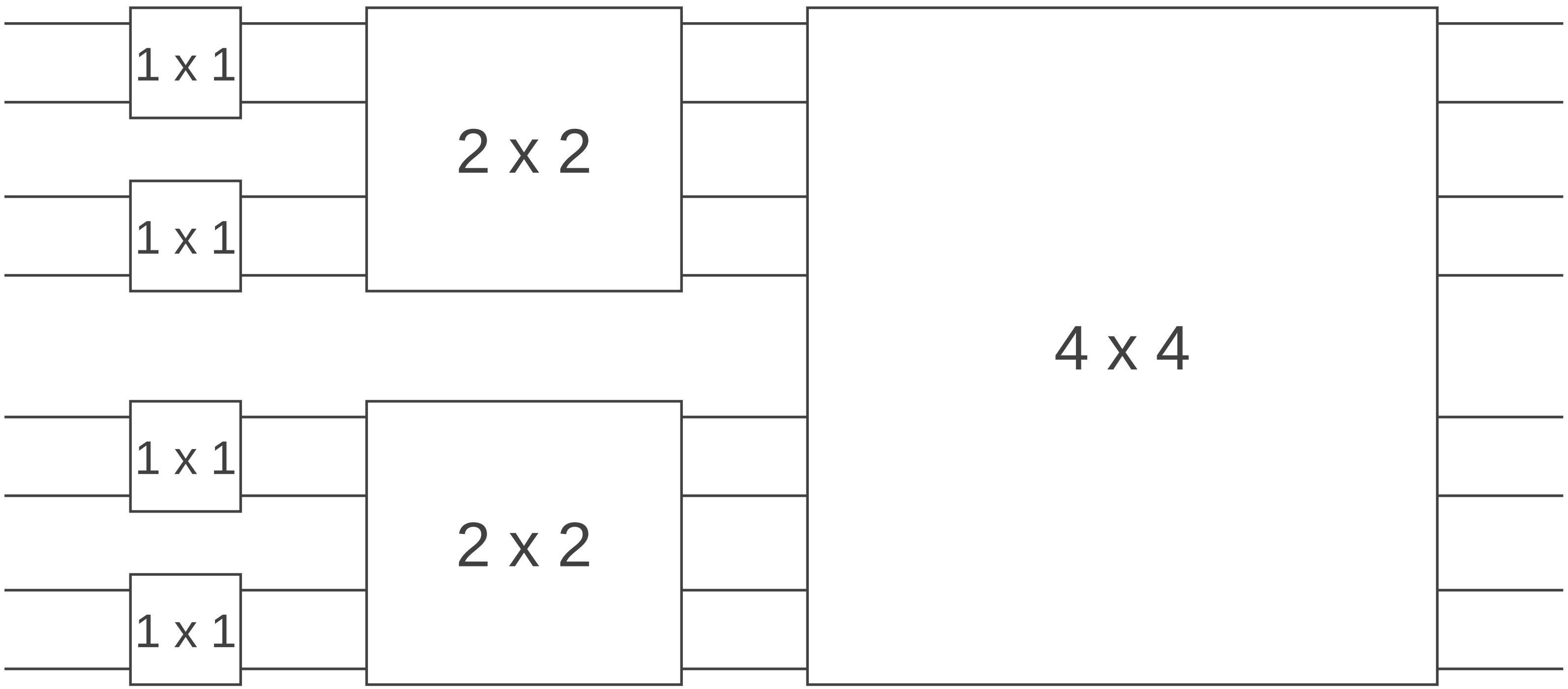
- Seřazené posloupnosti {a1, a2}, {b1, b2} jsou spojeny do seřazené posloupnosti {c1, c2, c3, c4}

- **Větší sítě n x n**
  - Liché prvky $\{a_1, a_3, ..\}\{b_1, b_3, ...\}$ jsou spojeny sítí n/2 x n/2 do sekvence $\{d_1, d_2, d_3, ...\}$ a podobně sudé prvky do sekvence $\{e_1, e_2, ...\}$
  - Finální sekvence $\{c_1, c_2, ...\}$
    - » $c_1 = d_1$
    - » $c_{2i} = \min(d_{i+1}, e_i)$
    - » $c_{2i+1} = \max(d_{i+1}, e_i)$
    - » $c_{2n} = e_n$
- Síť n x n:



Řazení:

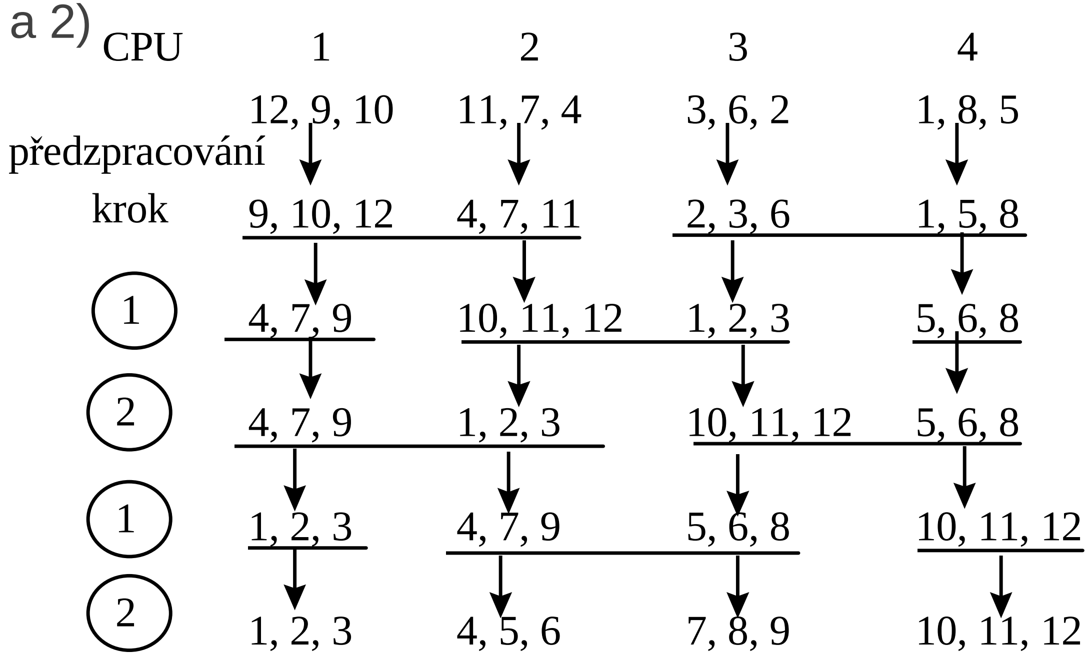
– Kaskádou sítí 1x1, 2x2, 4x4, ...



- Analýza
  - Řadíme posloupnost o délce $n=2^m$
  - 1.fáze potřebuje 2^{m-1} CE
  - 2.fáze potřebuje 2^{m-2} sítí 2x2 po 3 procesorech
  - 3.fáze 2^{m-3} sítí 4x4 po 9 procesorech
  - 4.fáze 2^{m-4} sítí po 25 procesorech
  - atd.
- Časová složitost
  - $t(n) = O(m^2) = O(\log^2 n)$
- Cena
  - $c(n) = O(n.\log^4 n)$ což není optimální

Merge-splitting sort

- Lineární pole procesorů p(n) < n
- Je variantou algoritmů lichý-sudý, kde každý procesor obsahuje několik čísel
- Porovnání a výměna je nahrazena operacemi merge-split
- Každý CPU se stará o více prvků
- Každý procesor obsahuje n/p čísel
- Po $\lceil p/2 \rceil$ iteracích (krok 1 a 2) je posloupnost seřazena



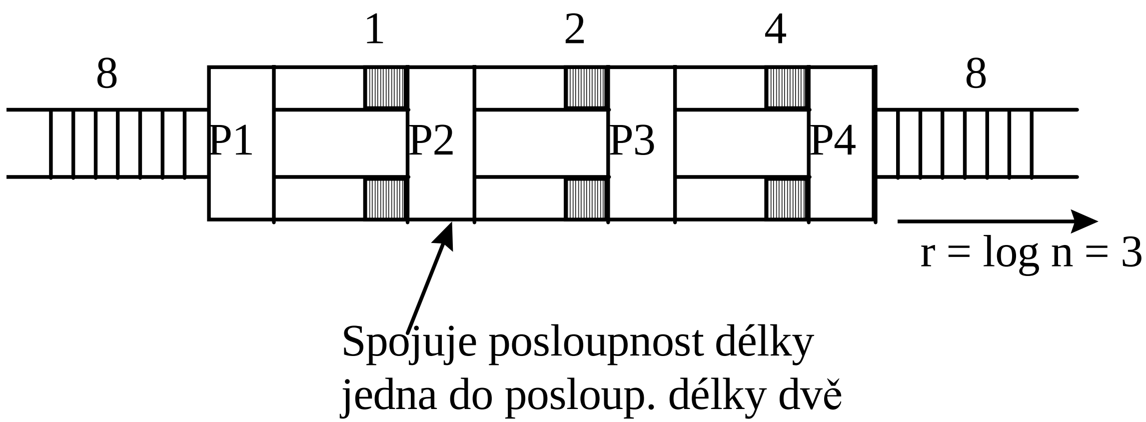
```
Algoritmus
for i = 1 to p do in parallel
        procesor Pi seřadí svou posloupnost sekvenčním algoritmem
endfor
for k = 1 to ⌈p/2⌉ do
        1) for i = 1, 3, ..., 2.⌊p/2⌋ do in parallel
                spoj Si a Si+1 do setříděné sekvence Si’
                Si = první polovina Si’
                Si+1 = druhá polovina Si’
           endfor
        2) for = 2, 4, ... , 2.⌊p/2⌋ do in parallel
                ....
           endfor
endfor
```

- Analýza
  - » předzpracování optimálním alg. — O((n/p)log (n/p))
  - » přenos S_{i+1} do P_i — O(n/p)
  - » spojení S_i a S_{i+1} do S_i’ optimálním alg. — 2.n/p
  - » přenos S_{i+1} do P_{i+1} — O(n/p)
  - » krok 1 nebo 2 — O(n/p)

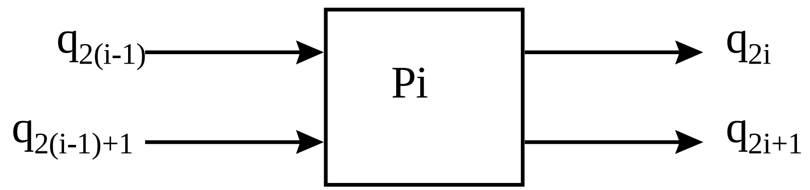
  t(n) = O[(n/p) log (n/p)]+O(n) = O((n log n)/p) + O(n)
  c(n) = t(n).p = O(n.log n) + O(n.p)
  což je optimální pro p ≤ log n

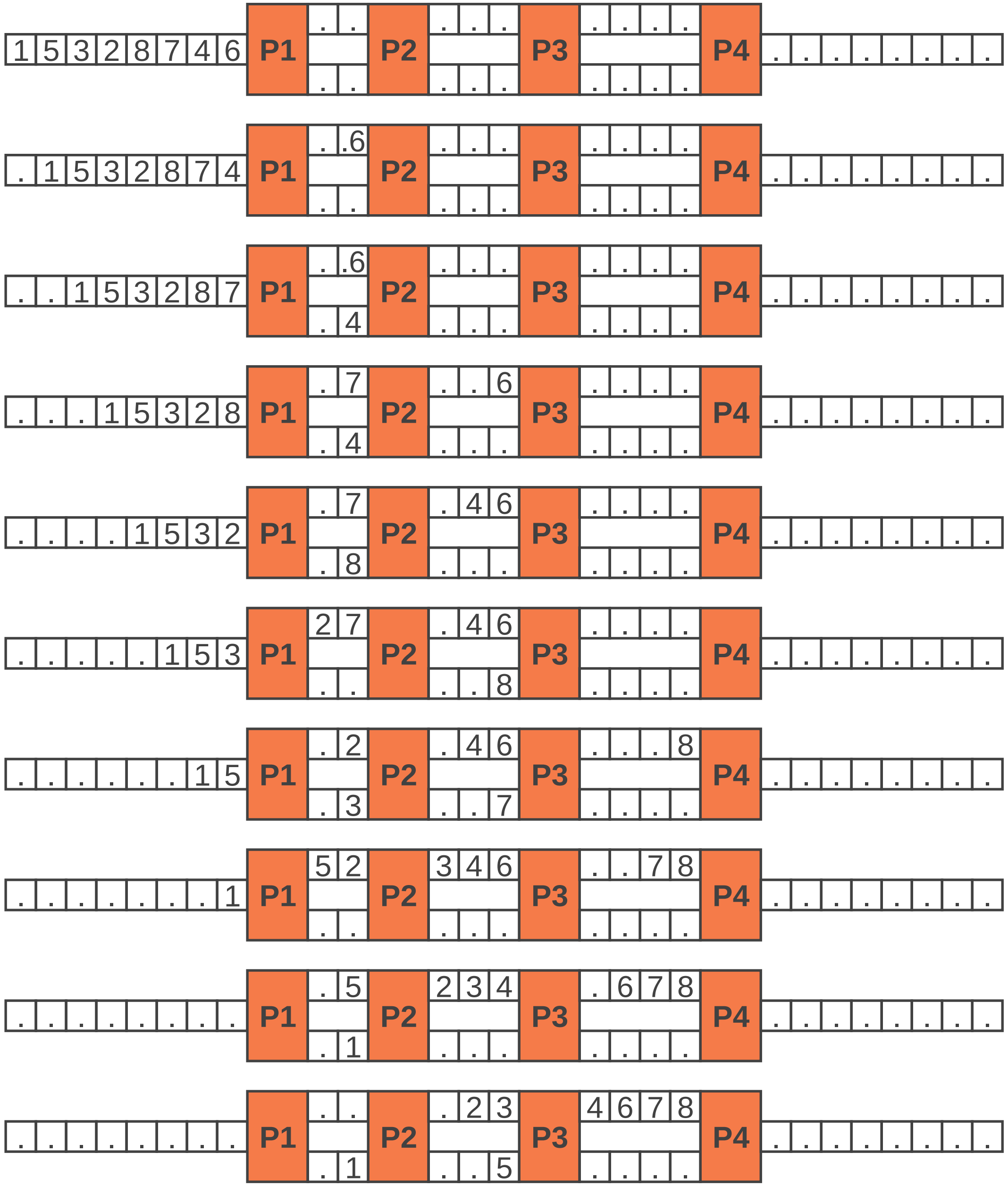
Pipeline Merge sort

- Lineární pole procesorů p(n) = log n + 1
- Data nejsou uložena v procesorech, ale postupně do nich vstupují
- Každý procesor spojuje dvě seřazené posloupnosti délky 2^{i-2}



Označení front :




P1
P2
P3
P4
1 5 3 2 8 7 4 6
5 2
3 4 6
7 8
2 3 4
6 7 8
4 6 7 8

Analýza:

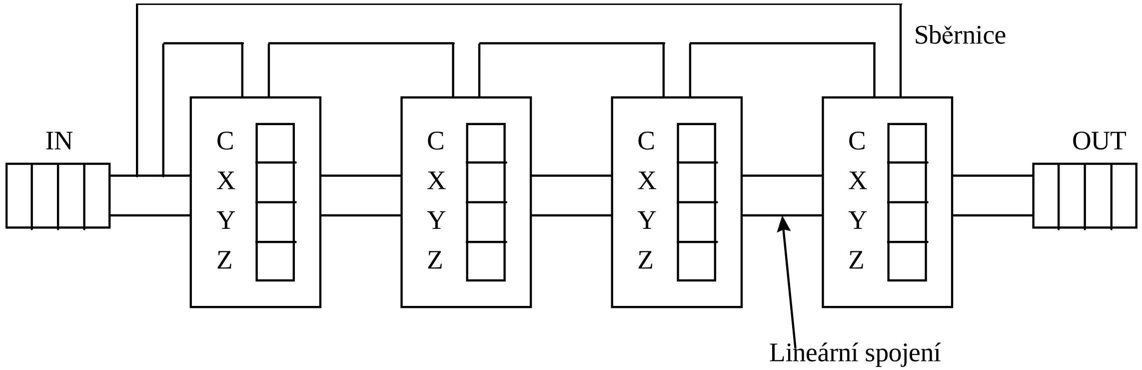
- Procesor P_i začne, když má na jednom vstupu posloupnost délky 2^{i-2} a na druhém 1, tedy začne $2^{i-2}+1$ cyklů po procesoru P_{i-1}.
- P_i tedy začne v cyklu

$$1+\sum_{j=0}^{i-2} 2^j + 1 = 2^{i-1} + i - 1$$

a skončí v cyklu $(n-1) + 2^{i-1} + i - 1$

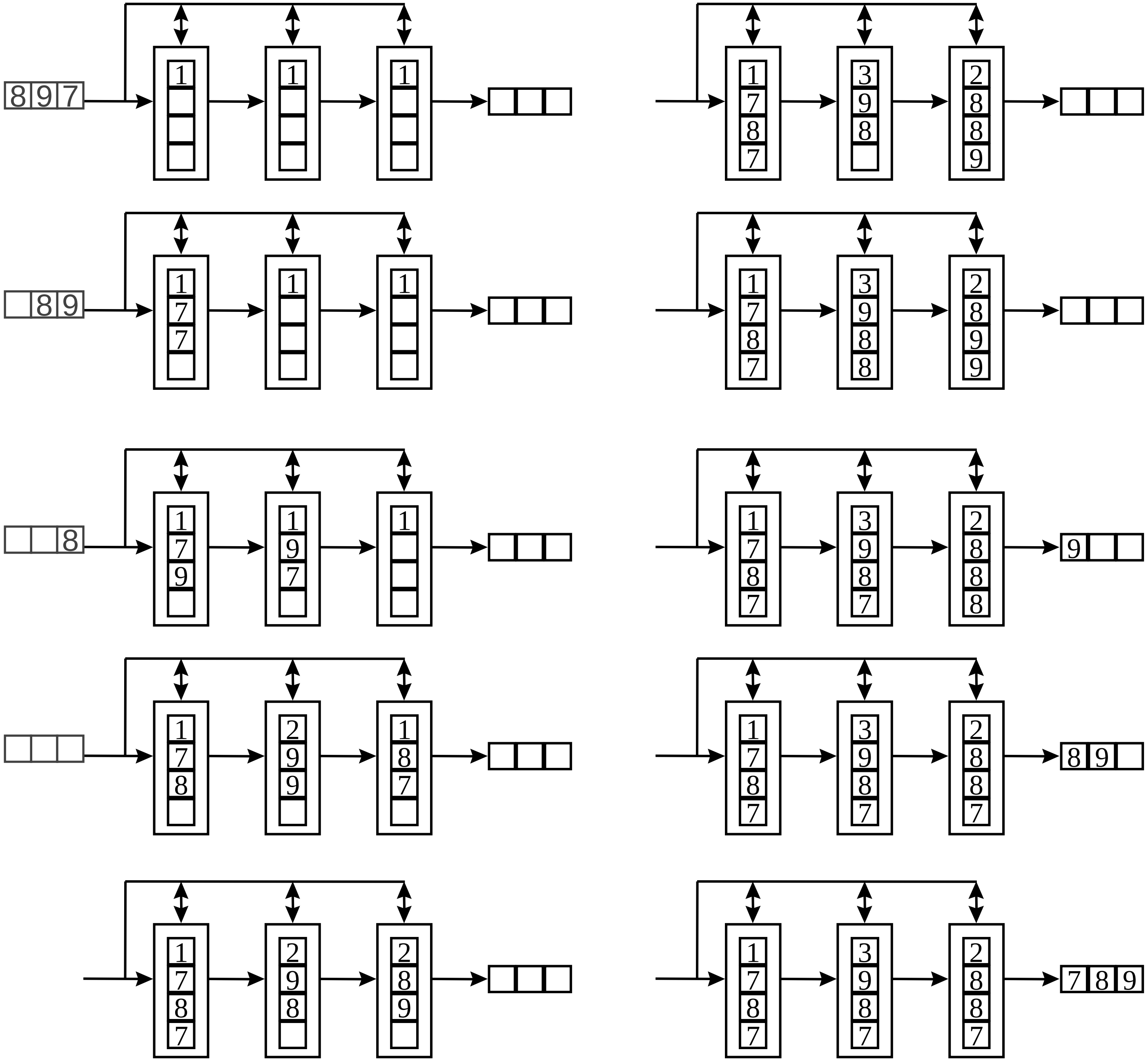
- Algoritmus skončí za:
  - $n + 2^r + r - 1 = 2n + \log n - 1$ cyklů, $t(n) = O(n)$
  - $c(n) = t(n).p(n) = O(n).(\log n + 1) = O(n.\log n)$ což je optimální

Enumeration sort



- Lineární pole n procesorů, doplněných společnou sběrnicí, schopnou přenést v každém kroku jednu hodnotu
- X_i - prvek x_i
- Y_i - postupně prvky $x_1 \ldots x_n$
- C_i - počet prvků menších než x_i
  (t.j. kolikrát byl $Y_i \leq X_i$)
- Z_i - seřazený prvek Y_i

- Algoritmus:
  - 1) Všechny registry C se nastaví na hodnotu 1
  - 2) Následující činnosti se opakují 2n krát $1 \leq k \leq 2n$
    - Pokud vstup není vyčerpán, vstupní prvek $\underline{x}_i$ se vloží do X_i (sběrnicí) a do Y_1 (lineárním spojením) a obsah všech registrů Y se posune doprava
    - Každý procesor s neprázdnými registry X a Y je porovná, a je-li X > Y inkrementuje C
    - Je-li k > n (t.j. po vyčerpání vstupu) procesor P_{k-n} pošle sběrnicí obsah svého registru X procesoru P_{Ck-n}, který jej uloží do svého registru Z
  - 3) V následujících <u>n</u> cyklech procesory posouvají obsah svých registrů Z doprava a procesor P_n produkuje seřazenou posloupnost



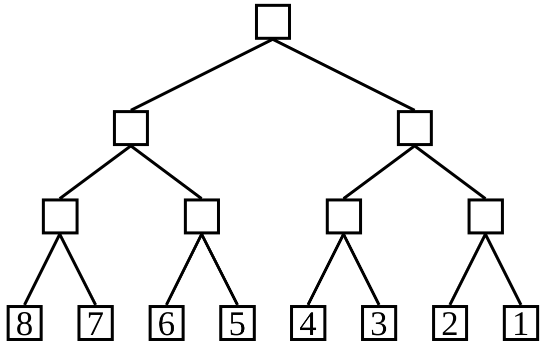
C
X
Y
Z

Formální algoritmus

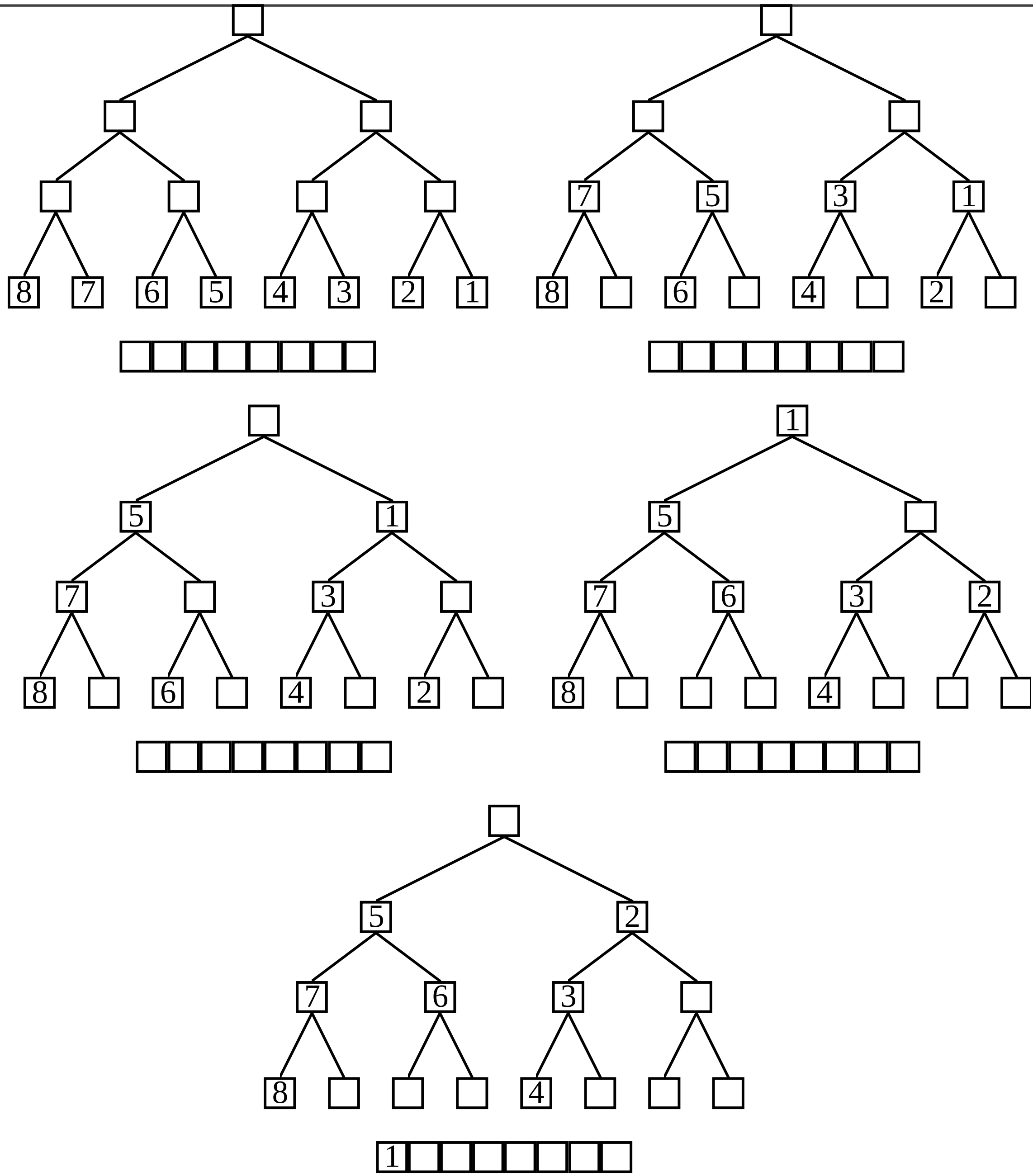
```
1) for i = 1 to n do in parallel
        Ci = 1
   endfor
2) for k = 1 to 2n do
        if k ≤ n then h = 1 else h = k - n endif
        for i = h to n do in parallel
          if (Xi nonempty and Yi nonempty) and Xi > Yi then Ci = Ci + 1 endif
         endfor
        for i = h to n - 1 do in parallel
          if Yi nonempty then Yi+1 = Yi endif
         endfor
        if k ≤ n then Y1 = nextinput, Xk = nextinput endif
        if k > n then ZCk-n = Xk-n endif
   endfor
3) for k = 1 to n
        output = Zn
        for i = k to n-1 do in parallel
          Zi+1 = Zi
        endfor
   endfor
```

- Analýza
  - Krok 1) je v konstantním čase, krok 2) trvá 2n cyklů, krok 3) trvá n cyklů
  - $t(n) = O(n)$
  - $c(n) = t(n).p(n) = O(n).n = O(n^2)$, což není optimální
- Poznámky
  - » Výpočet složitosti platí pouze za předpokladu, že přenos hodnoty sběrnicí trvá konstantní dobu bez ohledu na fyzickou vzdálenost procesorů
  - » Algoritmus není schopen řadit vstup obsahující stejné hodnoty (*Navrhněte modifikaci)*

Minimum Extraction sort



- Strom s <u>n</u> listy, (log n) + 1 úrovněmi a 2n-1 procesory
- Každý listový procesor obsahuje jeden řazený prvek
- Každý nelistový procesor umí porovnat dva prvky
- <u>Algoritmus:</u>
  - Každý list obsahuje jeden prvek
  - Každý nelistový procesor porovná hodnoty svých dvou synů a menší z nich pošle svému otci po (log n) + 1 krocích se minimální prvek dostane do kořenového procesoru
  - Každým dalším krokem se získá další nejmenší prvek

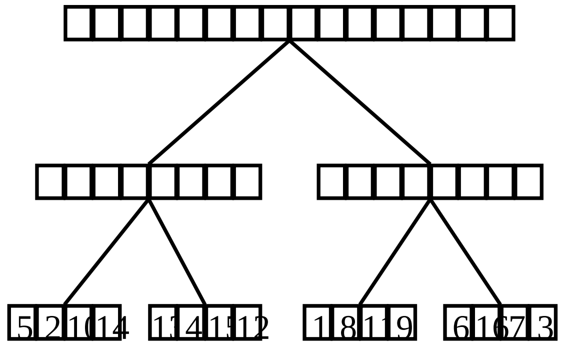


- <u>Algoritmus</u>

```
1) for all leafs do in parallel
          processor reads one element
    end for
2) for i=1 to 2n+(log n)-1 do
      for all nonleafs do in parallel
        if root and nonempty then output number
        else if nonempty then nothing
             else                                  {i.e. empty nonleaf}
               if  children empty then nothing
               else
                  if one child empty then get number from child
                  else                             {no child empty}
                    get smaller number from both children
                  endif
               endif
             endif
        endif
      endfor
   endfor
```

- Analýza
  - Jelikož strom má (log n)+1 úrovní, první prvek se získá po (log n)+1 krocích. Kořenový procesor potřebuje jeden krok na porovnání a jeden na uložení výsledku do paměti. Každý ze zbylých n-1 prvků spotřebuje 2 kroky.
- $t(n) = 2.n + (\log n) - 1 = O(n)$
- $p(n) = 2.n - 1$
- $c(n) = t(n).p(n) = O(n^2)$ což není optimální

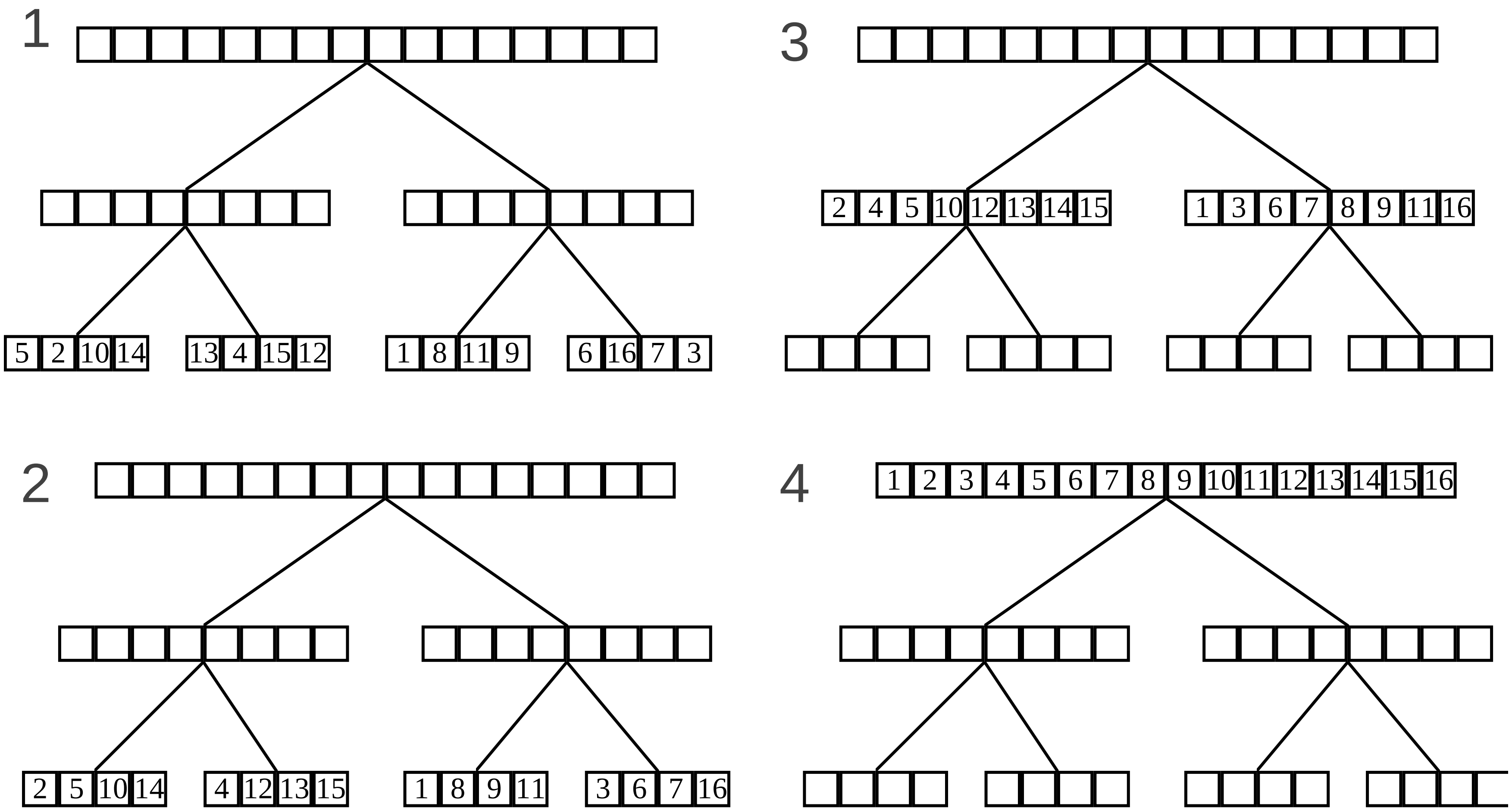
Bucket sort



- Řazení stromem procesorů s <u>m</u> listovými procesory $n = 2^m$
- Strom obsahuje 2^{m-1} procesorů, takže $p(n) = (2. \log n) - 1$
- Každý listový procesor obsahuje n/m řazených prvků a umí je seřadit optimálním sekvenčním algoritmem (např. heapsort)
- Každý nelistový procesor umí spojit dvě seřazené posloupnosti optimálním sekvenčním algoritmem
- <u>Algoritmus</u>
  - Řazené prvky se rovnoměrně rozdělí mezi listové procesory
  - Každý list seřadí svou posloupnost

```
for j = 1 to log m do
        for all processors at level (log m)-j do in parallel
                procesor spojí posloupnosti svých synů
        endfor
    endfor
```

  - Kořenový procesor uloží výslednou posloupnost do paměti

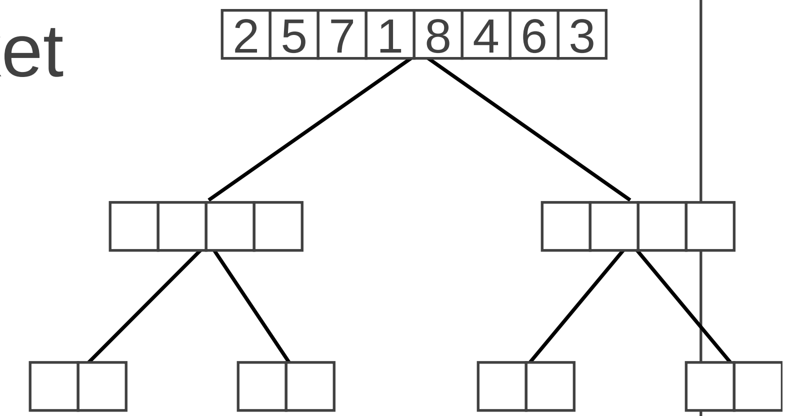

1
5 2 10 14
13 4 15 12
1 8 11 9
6 16 7 3
2
2 5 10 14
4 12 13 15
1 8 9 11
3 6 7 16
3
2 4 5 10 12 13 14 15
1 3 6 7 8 9 11 16
4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

- <u>Analýza</u>
  - 1. Každý listový procesor čte n/(log n) prvků, takže složitost je O(n/(log n))
  - 2. Při použití optimálního algoritmu O(r. log r) = O((n/log n)).log(n/(log n)) = O(n)
  - 3. Při j-té iteraci každý procesor na úrovni i = (log m) - j spojí dvě posloupnosti o délce $n/2^i$. Při použití např. straight merge každá j-tá iterace zabere $k.n/2^i$ kroků, kde <u>k</u> je konstantní. Krok 3 tedy trvá

$$\sum_{i=1}^{(\log m)-1} (k.n)/2^i = O(n)$$

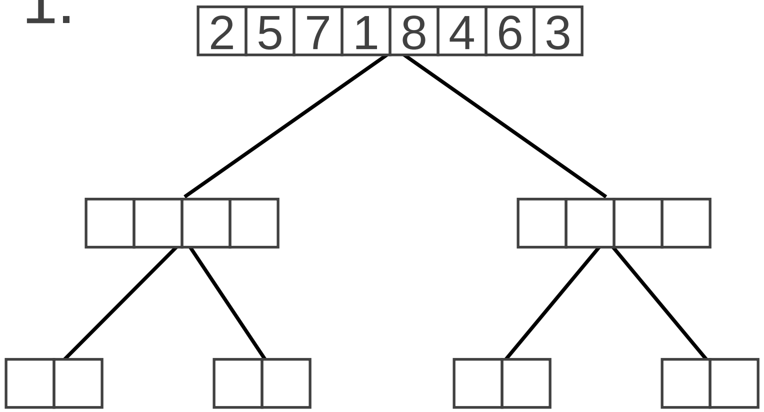
  - 4. O(n)
- Celkově tedy
  - t(n) = O(n) p(n) = O(log n)
  - c(n) = O(n.log n) → optimální

Median Finding and Splitting

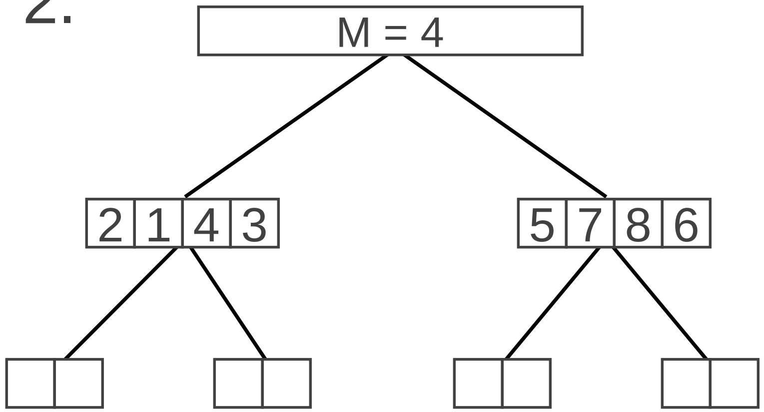


- Stejná architektura jako Bucket Sort
  - Strom procesorů s <u>m</u> listy, $n = 2^m$
  - Procesor na úrovni <u>i</u> zpracovává $n/2^i$ prvků
  - Každý list umí optimální sekvenční sort
- Nový požadavek
  K*aždý nelistový procesor umí nalézt medián v optimálním čase (např. algoritmus Select s O(n) složitostí)*

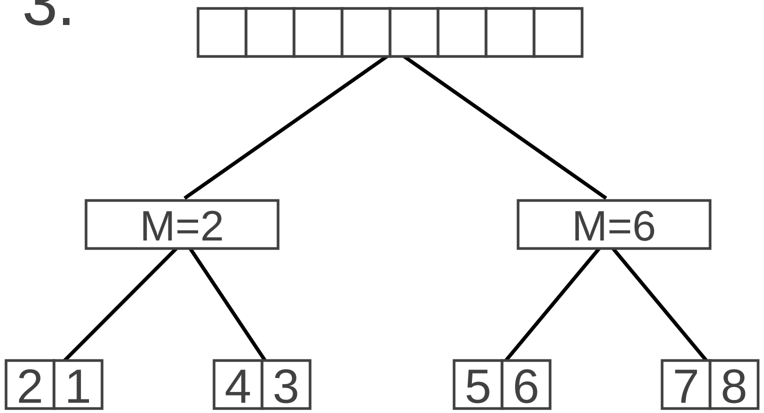
1.



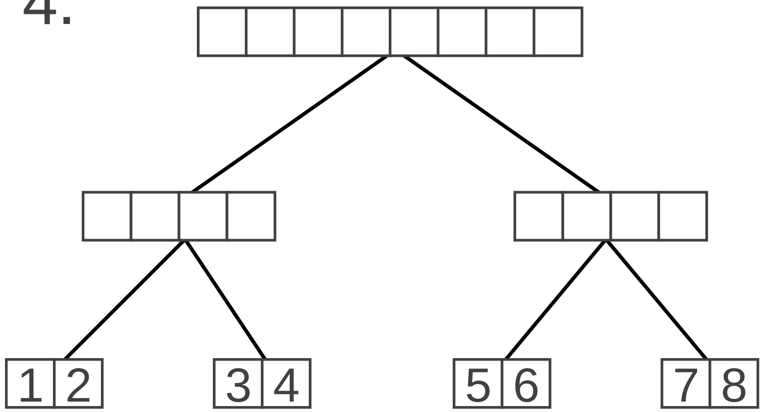
2.



3.



4.



- <u>Algoritmus</u>

```
1.Kořen načte řazenou sekvenci S
2.for i=0 to (log m) - 1 do
        for all processors at level i do in parallel
        2.1) Nalezni ve své sekvenci medián M (sekvenčně)
        2.2) Pro každý prvek x této sekvence
                if x ≤ M then pošli x levému synovi
                        else pošli x pravému synovi
                endif
        endfor
  endfor
3.for all leaf processors do
        3.1) setřiď svou sekvenci sekvenčním algoritmem
        3.2) ulož ji do výstupní vyrovnávací paměti
  endfor
```

- Analýza
  - 1.O(n)
  - 2.Procesor na i-té úrovni hledá medián v posloupnosti délky $n/2^i$, což mu při optimálním algoritmu trvá $O(n/2^i)$. Rozdělení posloupnosti trvá rovněž $O(n/2^i)$, tedy celkem krok 2:

$$\sum_{i=0}^{(\log m)-1} n/2^i = O(n)$$

  - 3.každý procesor řadí posloupnost délky n/log n což mu trvá O(n/(log n))/log(n/(log n))) a výstup uloží v čase O(n/log n)), takže složitost kroku 3 je O(n)

$t(n) = O(n)$ $p(n) = O(\log n)$
$c(n) = O(n.\log n)$ $\rightarrow$ což je optimální

- Diskuse
- Algoritmus se nechová dobře, pokud se v řazené posloupnosti vyskytují stejné prvky

Select (median)

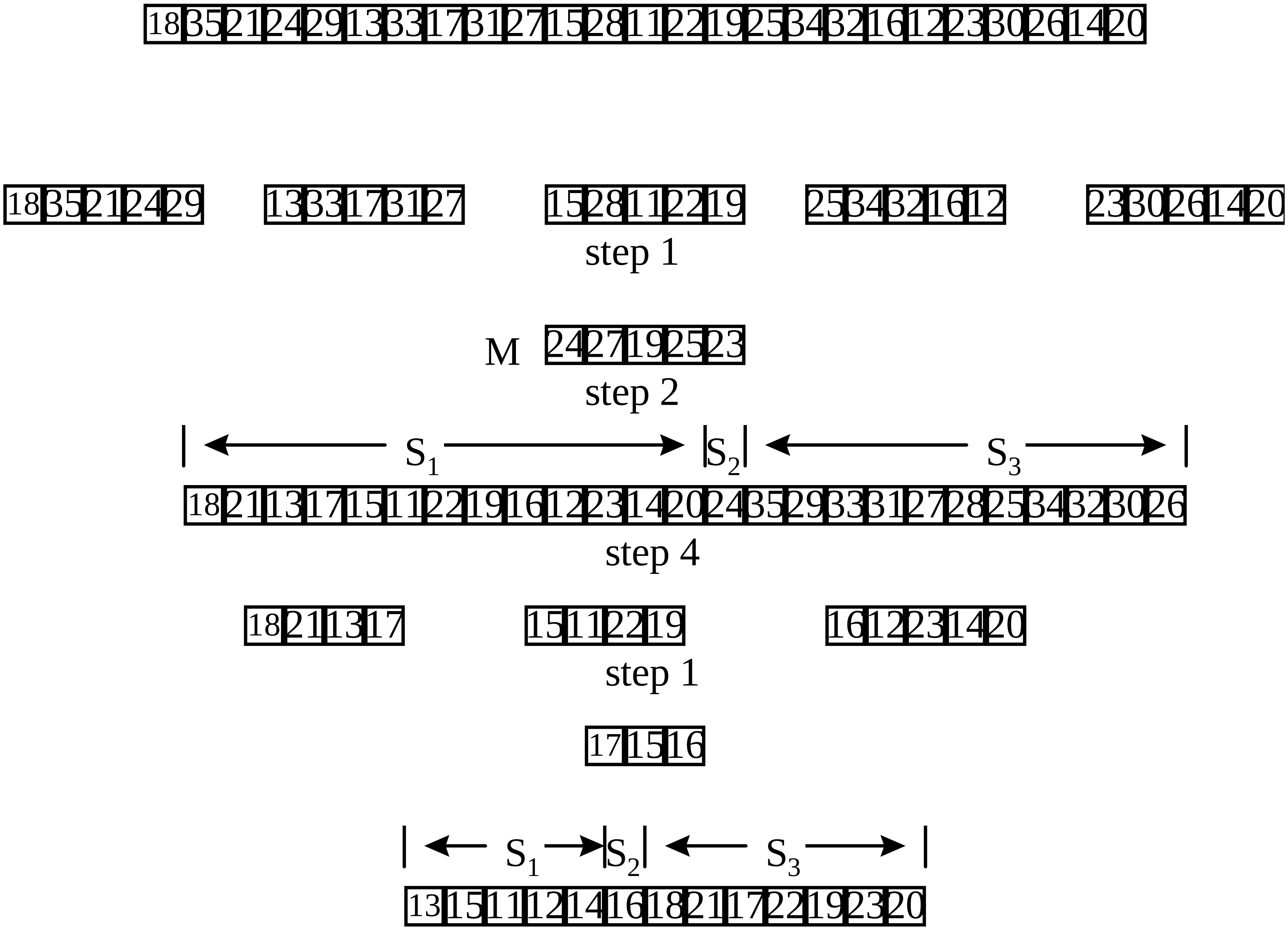
Petr Hanáček
Upd 2005,7

Sequential select

- Hledá k-tý nejmenší prvek v posloupnosti S
- Je-li k = |S| / 2, jde o medián

```
Algoritmus
procedure SEQUENTIAL_SELECT(S, k)
(1) if |S| ≤ Q then seřaď S a odpočítej
       else rozděl S na |S|/Q posloupností Si o délce Q prvků
(2) // Seřaď každou posloupnost Si a nalezni její medián M[i]
     for i=1 to |S|/Q do
       M[i] = SEQUENTIAL_SELECT(Si, |Si|/2)
     end for
(3) // Nalezni "medián mediánů" m
    m = SEQUENTIAL_SELECT(M, |M|/2)
(4) L =  {si ∈ S: si < m}
    E =  {si ∈ S: si = m}
    G = { si ∈ S: si > m}
(5) if |L| > k then SEQUENTIAL_SELECT(L,k) // prvek musí být v L
    else if |L| + |E| > k then return m    // prvek musí být v E
    else SEQUENTIAL_SELECT(G, k-|L|-|E|)   // prvek musí být v G
```

- Pro Q>=5   t(n) = O(n)


18 35 21 24 29 13 33 17 31 27 15 28 11 22 19 25 34 32 16 12 23 30 26 14 20
18 35 21 24 29
13 33 17 31 27
15 28 11 22 19
25 34 32 16 12
23 30 26 14 20
step 1
M
24 27 19 25 23
step 2
S_1
S_2
S_3
18 21 13 17 15 11 22 19 16 12 23 14 20 24 35 29 33 31 27 28 25 34 32 30 26
step 4
18 21 13 17
15 11 22 19
16 12 23 14 20
step 1
17 15 16
S_1
S_2
S_3
13 15 11 12 14 16 18 21 17 22 19 23 20

Parallel select

- Hledá k-tý nejmenší prvek v posloupnosti S
- EREW PRAM s N procesory $P_1 \dots P_N$
- Používá sdílené pole M o N prvcích

```
Algoritmus
procedure PARALLEL_SELECT(S, k)
(1) if |S| ≤ 4 then přímo nalezni k-tý prvek
    else rozděl S na N posloupností Si o délce n/N a každou přiřaď
          jednomu procesoru Pi
(2) for i=1 to N do in parallel
       M[i] = SEQUENTIAL_SELECT(Si, |Si|/2)
     end for
(3) m = PARALLEL_SELECT(M, |M|/2)  ← s menším počtem procesorů
(4) L =  {si ∈ S: si < m}
    E =  {si ∈ S: si = m }
    G = { si ∈ S: si > m}
(5) if |L| > k then PARALLEL_SELECT(L,k)
       else if |L| + |E| > k then return m
              else PARALLEL_SELECT(G, k-|L|-|E|)
```

- <u>Analýza</u>
  - $t(n) = O(n/N)$ pro $n > 4$, $N < n/\log n$
  - $p(n) = N$
  - $c(n) = t(n).p(n) = O(n)$ $\rightarrow$ což je optimální

Parallel splitting

- Krok 4 algoritmu Parallel select
- Úloha: Je dána posloupnost S a číslo m

  Mají se vytvořit tři posloupnosti:

  $$L = \{s_i \in S: s_i < m\}$$
  $$E = \{s_i \in S: s_i = m\}$$
  $$G = \{s_i \in S: s_i > m\}$$
- Složitost sekvenčního algoritmu je O(n)
- Paralelní řešení - máme N procesorů, které si sekvenci S rozdělí na podposloupnosti S_i o délce n/N

Algoritmus

(i) m se zašle všem procesorům procedurou BROADCAST

(ii) Každý procesor P_i rozdělí svoji sekvenci S_i na sekvence L_i, E_i, G_i

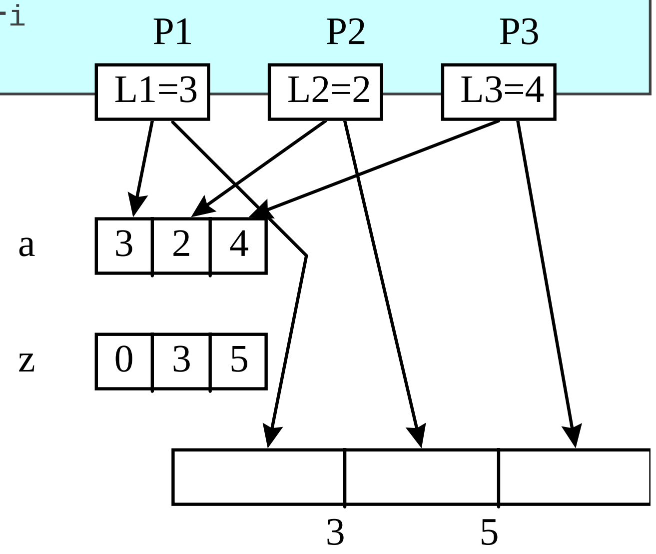
(iii) Všechny sekvence L_i jsou spojeny do sekvence L (a stejně tak E_i a G_i) následovně:

Nechť $a_i = |L_i|$

Pro všechny i, kde $1 \le i \le N$ se spočte suma:

$$z_i = \sum_{j=1}^{i} a_j \qquad a\ z_0 = 0$$

(iv) Nyní všechny procesory paralelně ukládají své posloupnosti L_i do L tak, že procesor P_i kopíruje L_i do L od pozice $z_{i-1}+1$.



- Analýza
  - (i) Krok (i) zabere čas O(log N)
  - (ii) Rozdělení se provádí optimálním sekvenčním algoritmem a zabere čas O(n/N)
  - (iii) Hodnoty z_i jsou vlastně suma prefixů a dají se procedurou ALLSUMS spočíst v čase O(log N)
  - (iv) Spojení subsekvencí se provádí paralelně s lineární složitostí a trvá O(n/N)

- Celková časová složitost
  - t(n) = O(log N + n/N) = O(n/N) pro dostatečně malé N
  - Cena c(n) = O(n/N).N = O(n) $\rightarrow$ což je optimální

Řazení na SIMD bez společné paměti

	t(n)	cena optimální?
Speciální topologie		
Enumeration Sort	O(log n)	N
Odd-even Marge Sort	$\log^2 n$	N
Lineární pole procesorů		
Odd-Even Trasposition	n	N
Merge-splitting Sort (agregovaná verze předchozího)		A
Pipeline Merge Sort	n	A
Enumeration Sort	n	N
Mřížka (mesh)		
Mesh Sort	$n^{1/2}$	N
Agregable Mesh Sort		A
Strom		
Minimum Extraction	n	N
Bucket Sort		A
agregovaná verze předchozího		
Median Finding and Splitting	n	A
Hyperkostka		
Cube Sort		
$t(n) = O(\log n) .. O(\log^2 n)$		
$p(n) = n^2 .. 2n$		