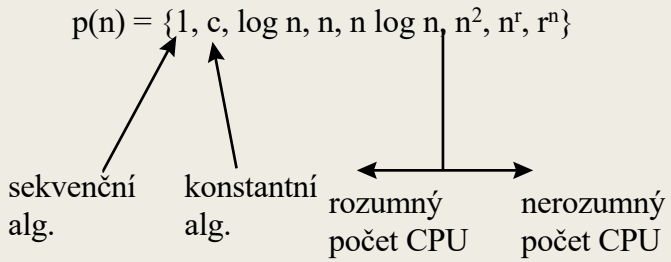
ANALÝZA ALGORITMŮ



- **Počet procesorů**
  - *(p) potřebných k řešení úlohy v závislosti na velikosti instance n*
- **Čas** řešení potřebný k řešení úlohy v jednotkách (krocích) t(n)
- **Cena** paralelního řešení: c(n) = p(n) . t(n)
  - *Algoritmus s optimální cenou: $c(n)_{optim} = t_{seq}(n)$ čas pro úlohu při sekvenčním výpočtu*
- **Zrychlení** x **Efektivnost**
  - *zrychlení $t_{seq}(n) / t(n)$*
  - *Efektivnost $t_{seq}(n) / c(n)$ - beru v úvahu i počet procesorů*
    - <1 neoptimální (přidá se režie)
    - =1 optimální
    - >1 ?

VYHLEDÁVACÍ ALGORITMY

Vyhledávání

- Vstup: sekvence **X** = $\{x_1, x_2, \ldots, x_n\}$ a prvek $\underline{x}$
- Úlohou je nalézt $x = x_k$ a jeho pozici $\underline{k}$
- Optimální sekvenční algorimus má složitost …

a) Pokud X není uspořádáno –

t(n)=**O**(n) **c**(n)=**O**(n)
(prohledání až všech n prvků)

b) Pokud X je uspořádáno – binární vyhledávání
t(n)=**O**(log n) **c**(n)=**O**(log n)
(prohledání až log n prvků)

Vyhledávání v neseřazené posloupnosti

Algorithm
procedura SEARCH(S, x, k) { sequence, element, result}
1. **for** i = 1 **to** N **do in parallel**
   read x
   endfor
2. **for** i = 1 **to** N **do in parallel**
   Si = {$s_{(i-1).(n/N)}+1$, $s_{(i-1).(n/N)}+2$, ..., $s_{i.(n/N)}$}
   SEQUENTIAL SEARCH (S_i, x, k_i)
   - *paralell Search calls SEQUENTAL Search*
endfor
3. **for** i = 1 **to** N **do in parallel**
   if k_i > 0 **then** k = k_i **endif**
   endfor

Analýza

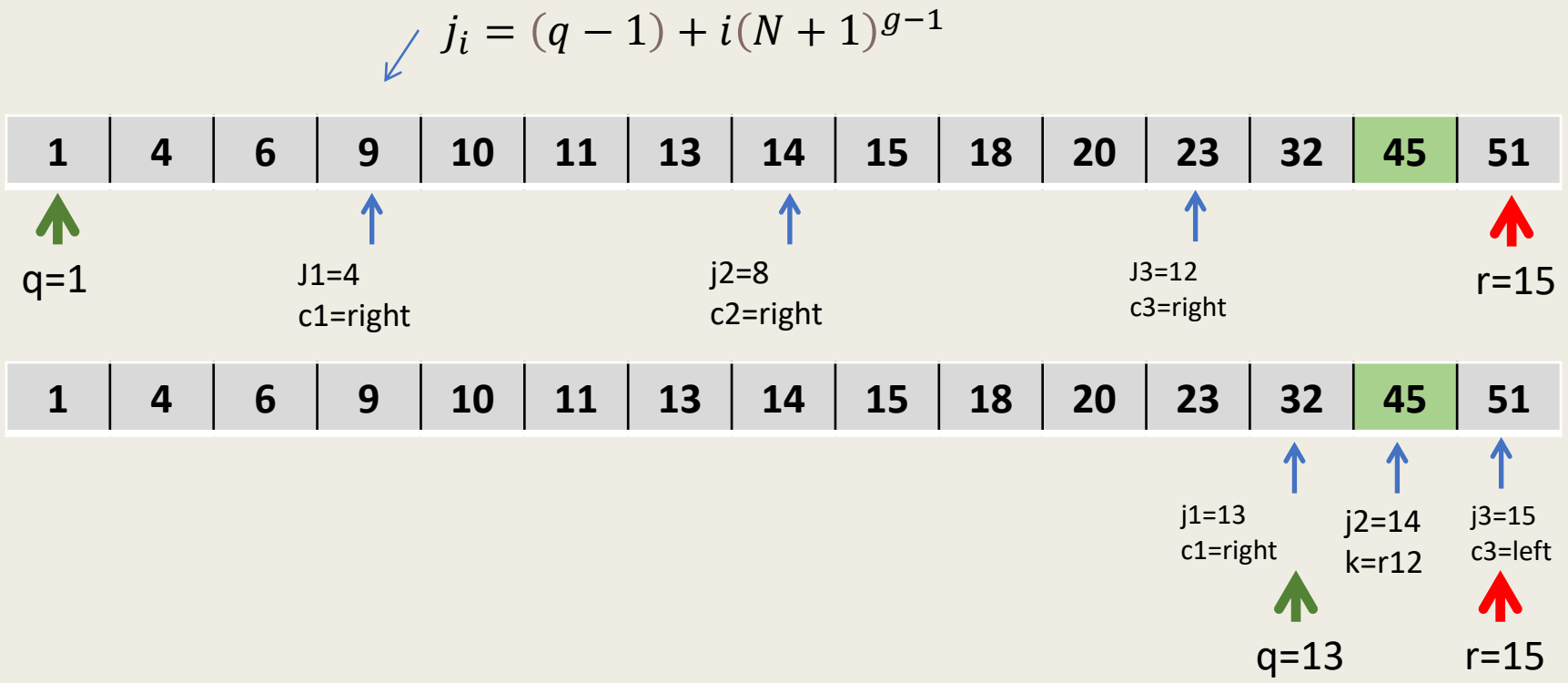
- **EREW**
  - *1. krok = O(log n)* *2. krok = O(n/N)* *3.krok = O(log N)*
  - *t(n) = O(log N + n/N)* *c(n) = O(N.log N + n)*
  - *Ve fázi 1. musí každý uzel znát hledaný prvek*
- **CREW**
  - *1. step = O(1)* *2. step = O(n/N)* *3.step= O(log N)*
  - *t(n) = O(log N + n/N)* *c(n) = O(N.log N + n)*
- **CRCW**
  - *1. steo = O(1)* *2. step = O(n/N)* *3.step = O(1)*
  - *t(n) = O(n/N)* *c(n) = O(n) což je* ***optimální***

N-ární vyhledávání na CREW

- Vyhledává v seřazené posloupnosti
- Princip algritmu:
  - Při použití binárního vyhledávání bychom mohli rozlišit, ve které polovině se prvek nachází (s využitím jednoho procesoru).
  - S **N** procesory můžeme provést **N+1** vyhledávání. V jednom kroku můžeme zjistit, v které části se prvek nachází
  - Pokud se liší ukazatel u dvou následujících prozkoumaných prvků (vlevo, vpravo), upraví se rozsah vyhledávání
- **g** je nejmenší číslo takové, že $n \leq (N+1)^g - 1$
- Potřebujeme udělat g = [log(n+1)/log(N+1)] kroků

N-ární CREW vyhledávání, příklad 1

- S={1, 4, 6, 9, 10, 11, 13, 14, 15, 18, 20, 23, 32, 45, 51}, |S|=15, hledáme 45
- Počet procesorů N=3 … tedy g=2 jelikož $15 \leq (3+1)^2 - 1$



procedure CREW SEARCH (S, x, k)

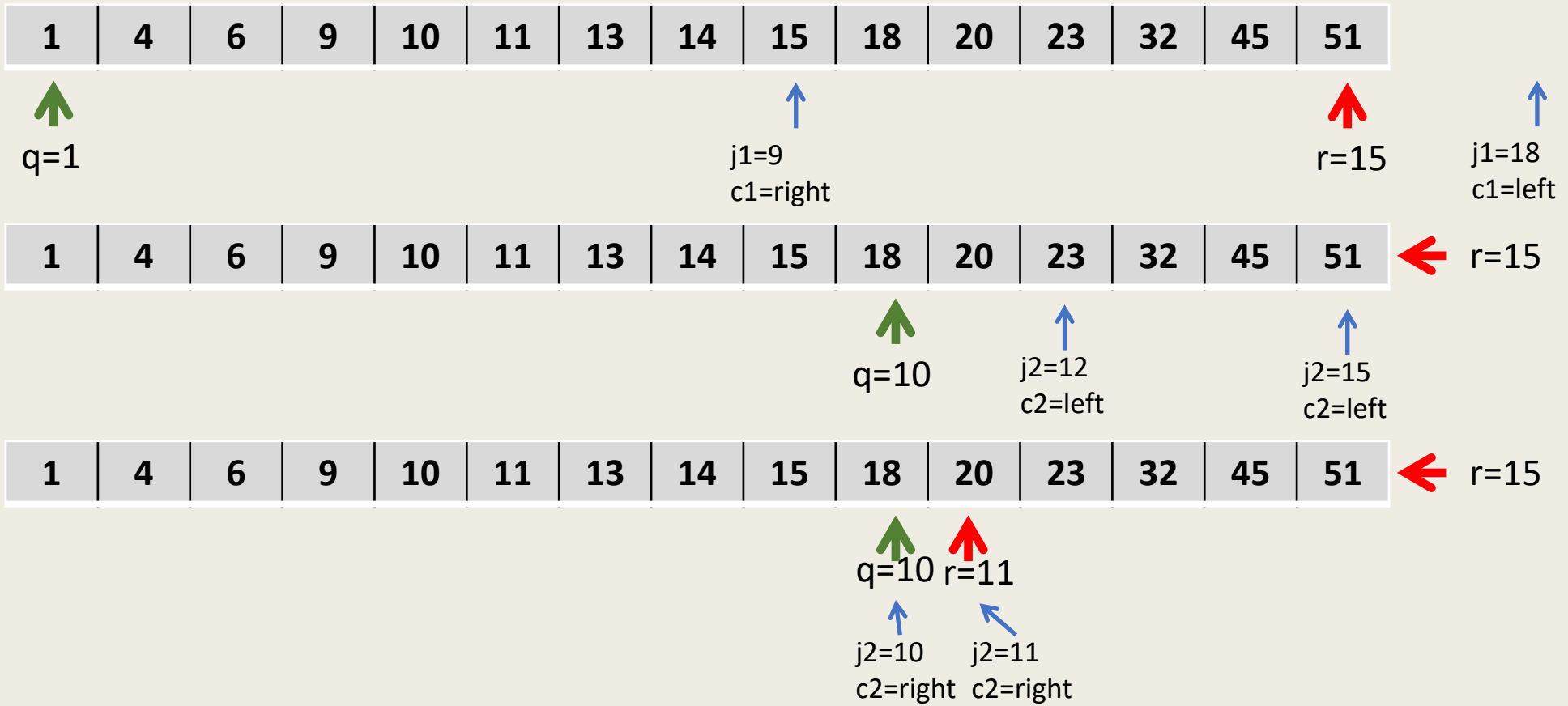
Step 1: {Initialize indices of sequence to be searched}
(1.1) $q \leftarrow 1$
(1.2) $r \leftarrow n$.

Step 2: {Initialize results and maximum number of stages}
(2.1) $k \leftarrow 0$
(2.2) $g \leftarrow \lceil \log(n+1)/\log(N+1) \rceil$.

Step 3: while $(q \leqslant r$ and $k = 0)$ do
(3.1) $j_0 \leftarrow q - 1$
(3.2) for i = 1 to N do in parallel
(i) $j_i \leftarrow (q-1) + i(N+1)^{g-1}$
{P_i compares x to s_j and determines the part of the sequence to be kept}
(ii) if $j_i \leqslant r$
then if $s_{j_i} = x$
then $k \leftarrow j_i$
else if $s_{j_i} > x$
then $c_i \leftarrow$ left
else $c_i \leftarrow$ right
end if
end if
else (a) $j_i \leftarrow r + 1$
(b) $c_i \leftarrow$ left
end if
{The indices of the subsequence to be searched in the next iteration are computed}
(iii) if $c_i \neq c_{i-1}$ then (a) $q \leftarrow j_{i-1} + 1$
(b) $r \leftarrow j_i - 1$
end if
(iv) if (i = N and $c_i \neq c_{i+1}$) then $q \leftarrow j_i + 1$
end if
end for
(3.3) $g \leftarrow g - 1$.
end while. □

N-ární CREW vyhledávání, příklad 2

- S={1, 4, 6, 9, 10, 11, 13, 14, 15, 18, 20, 23, 32, 45, 51}, |S|=15, hledáme 21
- Počet procesorů N=2 … tedy g=3 jelikož 15<=(2+1)^3 -1



Vyhledávání, analýza

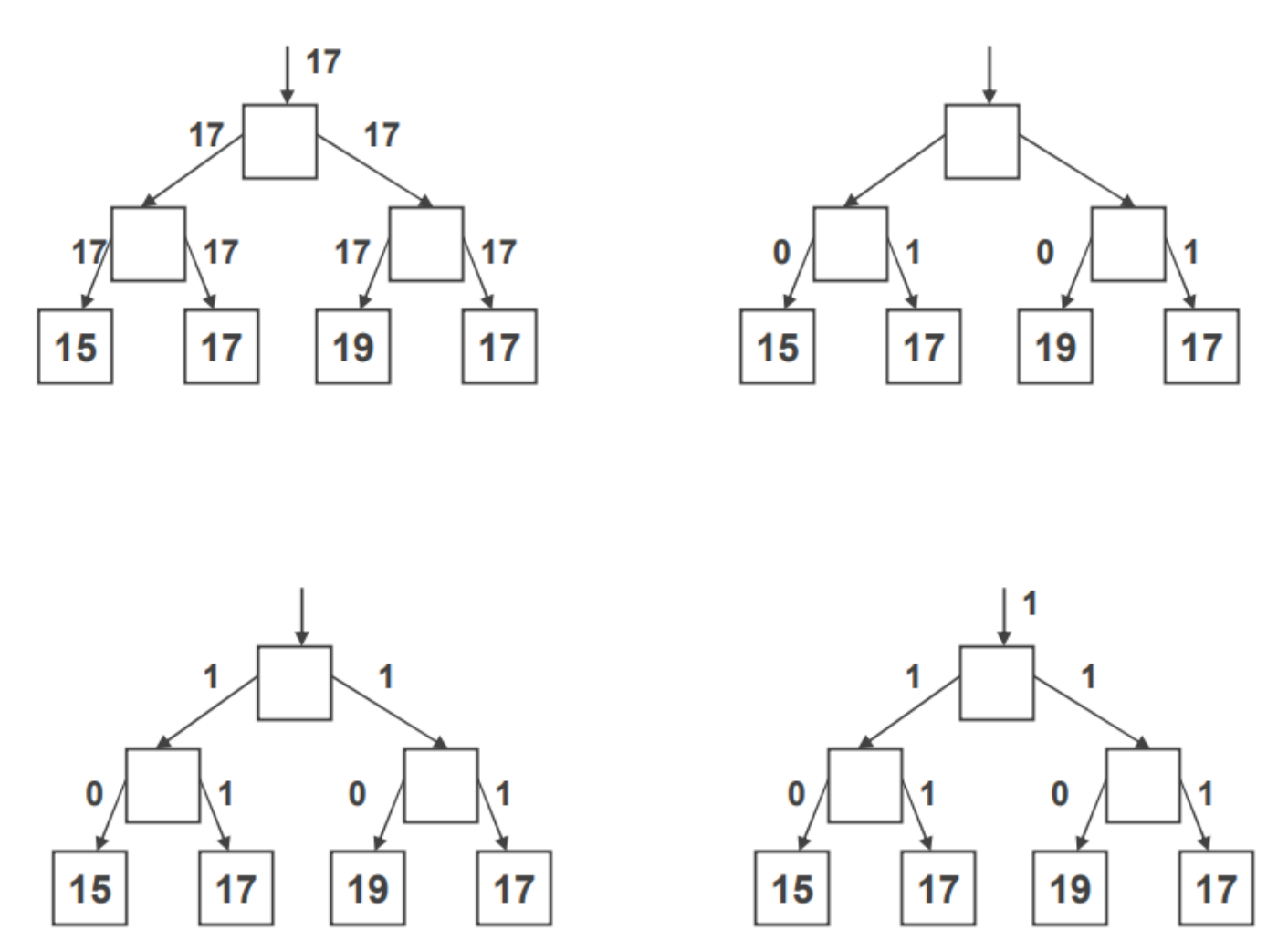
Potřebujeme **CREW PRAM**

- $t(n) = O(\log(n+1)/\log(N+1)) = O(\log_{N+1}(n+1))$
- $c(n) = O(N.\log_{N+1}(n+1))$ což ***není optimální***

Vyhledávání na stromě

- Topologie – Binární strom s 2n-1 procesory
- Algoritmus
  - *1. Kořen stromu načte hledanou hodnotu a předá ji synům, a ty dále až k listovým uzlům*
  - *2. Listy obsahují prvek nebo prvky, ve kterých se snaží hodnotu vyhledat, výsledkem je 0 or 1.*
  - *3. Všechny tyto výsledky jsou předány kořenu*
    - každý nelistový uzel spočte logický součet hodnot potomků a přepošle výsledek výš
    Kořen obdrží 0 – nenalezeno, 1- nalezeno

Tree Search, příklad



Tree Search, analýza

Krok (1) má složitost O(log n),
krok (2) má konstantní složitost,
krok (3) složitost O(log n).

$t(n) = O(\log n)$ $p(n) = 2.n-1$

$c(n) = t(n).p(n) = O(n.\log n)$ *což* ***není optimální***

Parallel splitting (Paralelní rozdělení)

- Úloha:

  Vstup: Sekvence čísel **S** a číslo m

  Výstup: Tři sekvence **L**, **E** a **G** takové, že

  $$\mathrm{L} = \{s_i \in S : s_i < m\}$$
  $$\mathrm{E} = \{s_i \in S : s_i = m\}$$
  $$\mathrm{G} = \{s_i \in S : s_i > m\}$$

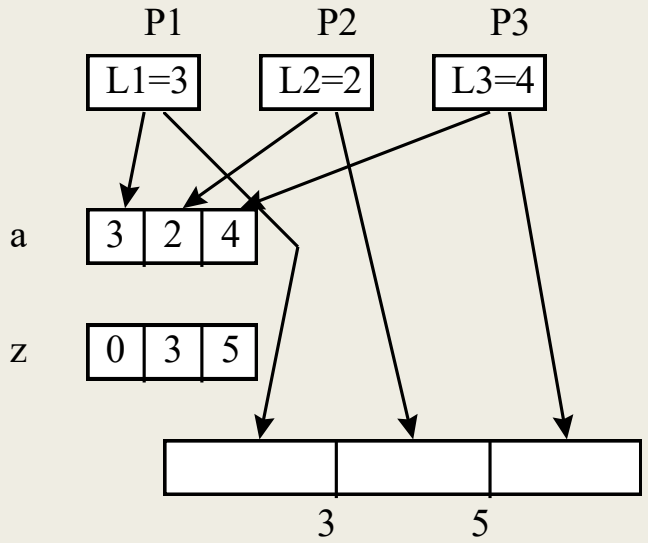
- Složitost sekvenčního algoritmu je O(n)
- Paralelní řešení – máme N procesorů, které rozdělí sekvenci S na podsekvence S_i délky n/N

Parallel splitting, Algoritmus

(i) $\underline{m}$ zasláno všem z N procesorů (BROADCAST)
(ii) Každý processor $\underline{P_i}$ rozdělí posloupnost S_i na posloupnosit L_i, E_i, G_i
(iii) Všechny sekvence L_i jsou spojeny do L
$a_i = |L_i|$
Pro všechna i, $1 < i < N$ se spočte suma:

$x_i = \sum_{j=1}^{i} a_j$ a_j a $x_0 = 0$

(iv) Každý procesor zapíše svou sekvenci Li paralelně id indexu $z_{i-1}+1$
Obdobně pro sekvence E_i a G_i, indexy jsou počítány od konce předešlých, tedy k y_i se přičte $x_{|N|}$ a k z_i se přičte $y_{|N|}$



Parallel splitting, Příklad

- Vstup: 13, 1, 12, 14, 3, 6, 8, 10, 2, 15, 7, 11, 4, 5, 9, číslo m = 8
- Počet procesorů N=3
- Rozdělíme na 3 posloupnosti o 5 prvcích a hledáme medián |Si|=5

13, 1, 12, 14, 3	6, 8, 10, 2, 15	7, 11, 4, 5, 9
L_1={1, 3}	L_2={2, 6}	L_3={4, 5, 7}
E_1={}	E_2={8}	E_3={}
G_1={12, 13, 14}	G_2={10, 15}	G_3={9, 11}

$a_1=|L_1|=2$ $a_2=|L_2|=2$ $a_3=|L_3|=3$

$$x_0 = 0 \; x_1 = \sum_{i=1..1} a_i = 2, x_2 = \sum_{i=1..2} a_i = 4 \; x_3 = \sum_{i=1..3} a_i = 7$$

$b_1=|E_1|=0$ $b_2=|E_2|=1$ $b_3=|E_3|=0$

$$y_0 = x_3 = 7 \;\; y_1 = \sum_{i=1..1} b_i + x_3 = 7 \;\; y_2 = \sum_{i=1..2} b_i + x_3 = 8 \;\; y_3 = \sum_{i=1..3} b_i + x_3 = 8$$

$c_1=|G_1|=3$ $c_2=|G_2|=2$ $c_3=|G_3|=2$

$$z_0 = y_3 = 8 \;\; z_1 = \sum_{i=1..1} c_i + y_3 = 11 \;\; z_2 = \sum_{i=1..2} c_i + y_3 = 13 \;\; z_3 = \sum_{i=1..3} c_i + y_3 = 15$$

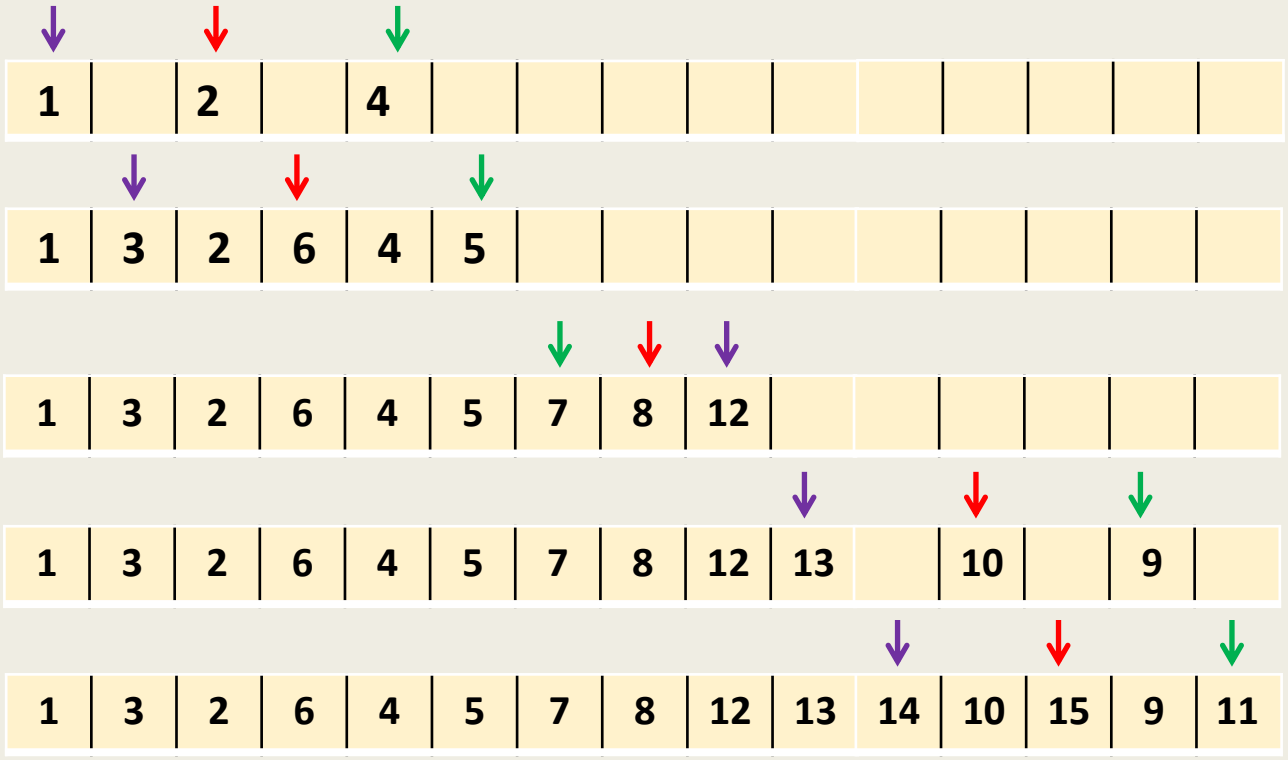
Parallel splitting, Příklad

L_1={1, 3}
E_1={}
G_1={12, 13, 14}

L_2={2, 6}
E_2={8}
G_2={10, 15}

L_3={4, 5, 7}
E_3={}
G_3={9, 11}

$\boldsymbol{x_0 = 0}$ $\boldsymbol{x_1 = 2}$ $\boldsymbol{x_2 = 4}$ $x_3 = 7$
$\boldsymbol{y_0 = 7}$ $\boldsymbol{y_1 = 7}$ $\boldsymbol{y_2 = 8}$ $y_3 = 8$
$\boldsymbol{z_0 = 8}$ $\boldsymbol{z_1 = 11}$ $\boldsymbol{z_2 = 13}$ $z_3 = 15$



Parallel splitting, Analýza

- Analýza
  - *(i) Krok (i) trvá O(log N)*
  - *(ii) Rozdělení optimálním sekvenčním algoritmem má složitost O(n/N)*
  - *(iii) Hodnoty x, y a z se dají spočítat sumou prefixů (ukážeme později) v O(log N)*
  - *(iv) Následné spojení má složitost O(n/N)*

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

Nalezení k-tého nejmenšího prvku, sekvenční řešení

```
procedure SEQUENTIAL_SELECT(S, k)
(1) if |S| < Q then sort S and count down
        else divide S into |S|/Q sequences S_i of length Q
(2) // Sort every sequence S_i and find its median M[i]
     for i=1 to |S|/Q do
        M[i] = SEQUENTIAL_SELECT(S_i, |S_i|/2)
     end for
(3) // Find "median of medians" m
    m = SEQUENTIAL_SELECT(M, |M|/2)
(4)     L = {s_i ϵ S: s_i < m}
        E = {s_i ϵ S: s_i = m}
        G = {s_i ϵ S: s_i > m}
(5) if |L| > k then SEQUENTIAL_SELECT(L,k) // element must be in L
    else if |L| + |E| > k then return m    // element must be in E
    else SEQUENTIAL_SELECT(G, k-|L|-|E|)   // element must be in G
```

- Pokud k=|S|/2 jedná se o medián
- t(n)=O(n)

Nalezení k-tého nejmenšího prvku, paralelní řešení

```
procedure PARALLEL SELECT(S, k)
(1) if |S| < 4 then find imediatly the k-th element
    else divide S into N subsequences Si of length n/N and every
         sequence Si  assert to a processor Pi
(2) for i=1 to N do in parallel
       M[i] = SEQ.SELECT(Si, |Si|/2)
     end for
(3) m = PARALL.SELECT(M, |M|/2)  with the lowest number
                                                of processors
(4) PALALLEL SPLITTING(S, m)
       L = {si ϵ S: si < m}
       E = {si ϵ S: si = m}
       G = {si ϵ S: si > m}
(5) if |L| > k then PAR.SELECT(L,k)
       else if |L| + |E| > k then return m
              else PAR.SELECT(G, k-|L|-|E|)
```

- Pokud k=|S|/2 jedná se o medián
- t(n)=O(n)

Příklad

- Vstup: 13, 1, 12, 14, 3, 6, 8, 10, 2, 15, 7, 11, 4, 5, 9
- Počet procesorů N=4
- Rozdělíme na 3 posloupnosti o 5 prvcích a hledáme medián |**Si**|=15, |**Si**| / 2 = 7

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

  M[1]=12 M[2]=8 M[3]=7

  12, 6, 7, 5 -> sekvenčně M=7

Použijeme Parallel Splitting

|L|= 6 |M|=1 |G|= 8

//.pokud bychom hledali 8. prvek, pokračujeme hledáním prvního v G

Časová složitost podrobněji

- Při volbě pivotů a rozdělení, hledáme v posloupnosti s asi 70% uzlů
- Celková práce přes rekurze se dá odhadnout řadou $\mathrm{n} + \left(\frac{7}{10}\right)\mathrm{n} + \left(\frac{7}{10}\right)^2\mathrm{n} \ldots$

  To je geometrická řada s kvocientem $\gamma = 0.7$

  Její součet je $\sum_{i=1}^{\infty} n\gamma^i = \frac{n}{1-\gamma} = \frac{n}{1-0.7} = \frac{n}{0.3} \approx 0.33n$

  Tedy třída časové složitosti je je $O(n)$

ALGORITMY ŘAZENÍ

Paralelní řadící algoritmy

- Porovnávací algoritmy
  - *Odd – Even Transposition Sort*
- Výčtové řadící algoritmy
  - *Na mřížce*
  - *Na lineárním poli*
- Algoritmy slučováním
  - *Odd – Even Merge Sort*
  - *Pipeline Merge Sort*
  - *Merge – Splitting Sort*
- Algoritmy založené na výběru minima
  - *Minimum Extraction Sort*
- Algoritmus rozdělením
  - *Bucket Sort*
  - *Median Finding and Splitting (+ nalezení Mediánu)*

Odd – Even Transposition Sort

- Topologie:

  lineární pole n procesorů p(n) = n

- Princip:

  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, platí-li $y_i > y_{i+1}$, procesory vymění své hodnoty

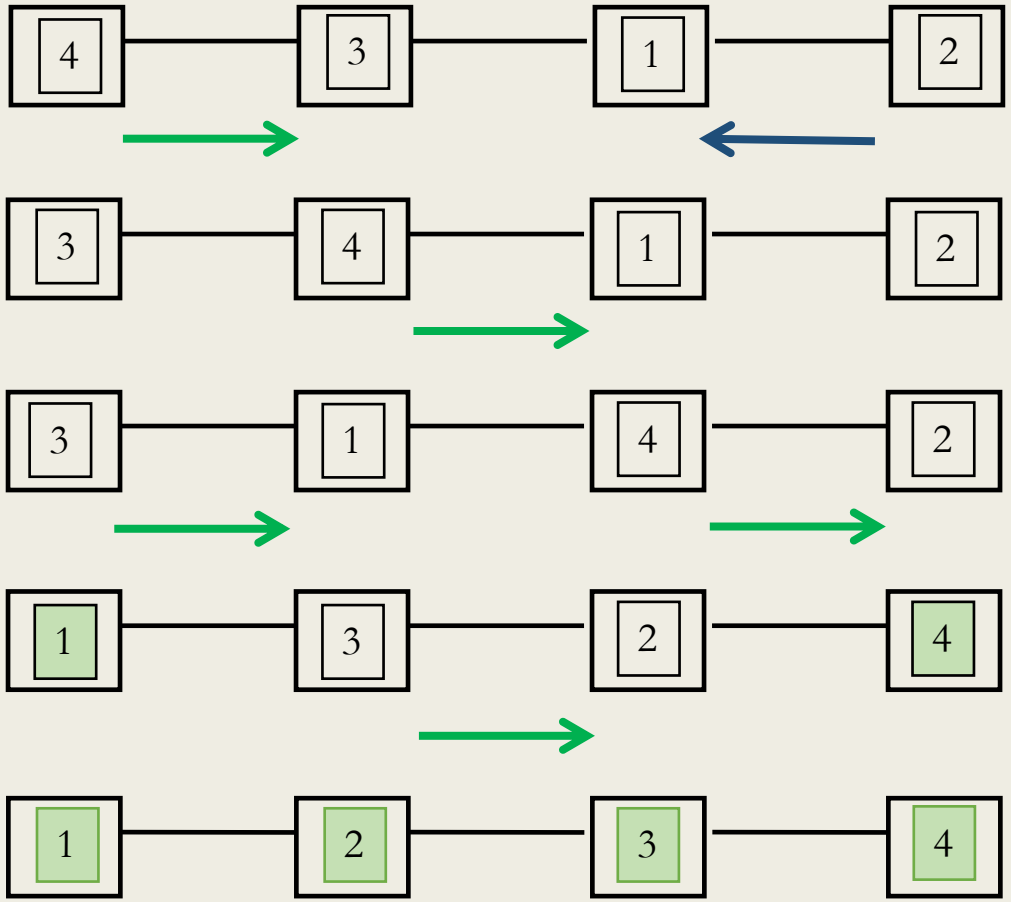
  v druhém kroku se každý sudý procesor … to samé

  po n krocích (maximálně) jsou hodnoty seřazeny

- Algoritmus

```
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
```

Odd – Even Transposition Sort, příklad



1. krok, liší

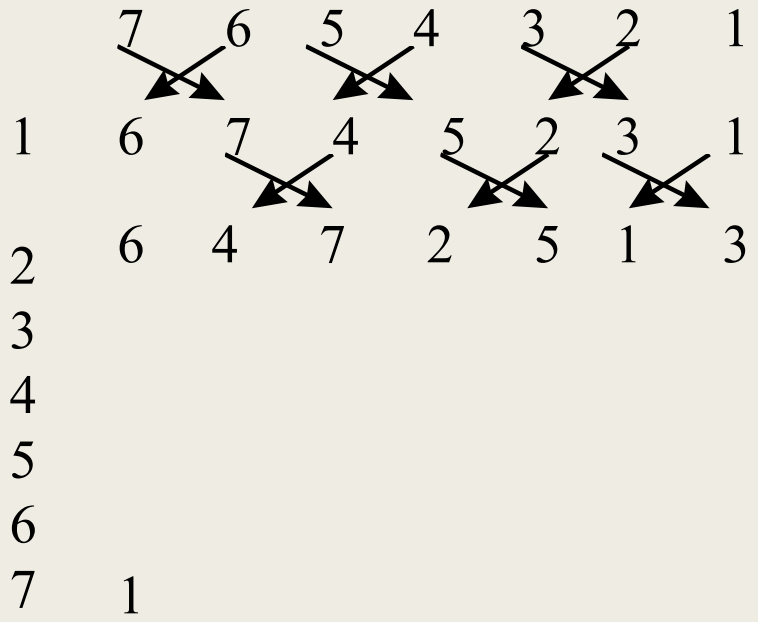
1. krok, sudí

2. krok, liší

2. krok, sudí

Odd – Even Transposition Sort, 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.c



Enumeration Sort (Mřížka)

- Topologie:

  dvojrozměrné pole **n x n** procesorů, vertikálně i horizontálně propojeny do stromové struktury

- Uzel

  může uložit dva prvky do svých registrů A a B

  může porovnat A a B a uložit výsledek do registru RANK

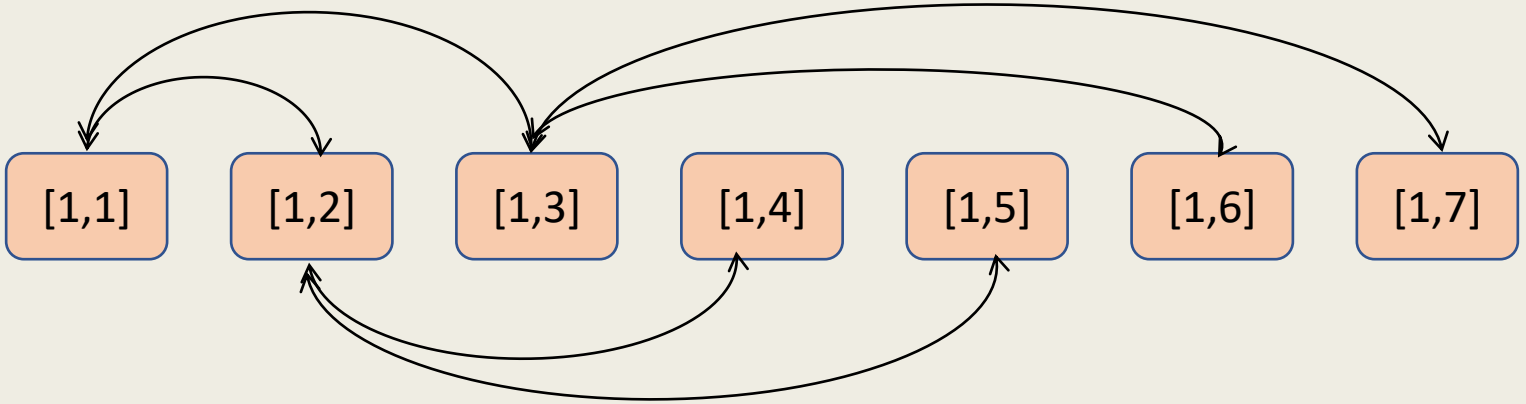
  pomocí stromového propojení může zadat obsah kteréhokoli registru jinému procesoru

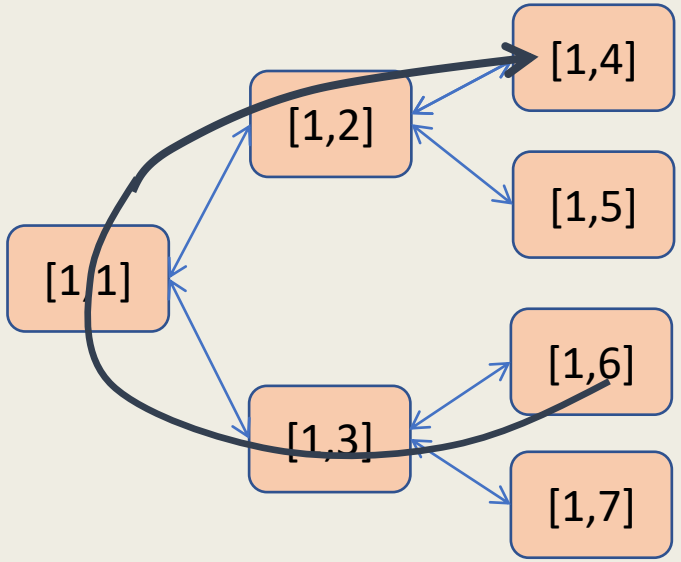
  může přičítat k registru RANK v RANK bude 1, nebo 0

- Princip:

  správná pozice každého prvku ve výstupní seřazené posloupnosti je dána počtem prvků, které jsou menší než tento prvek

Komunikace na stromě (připomenutí)





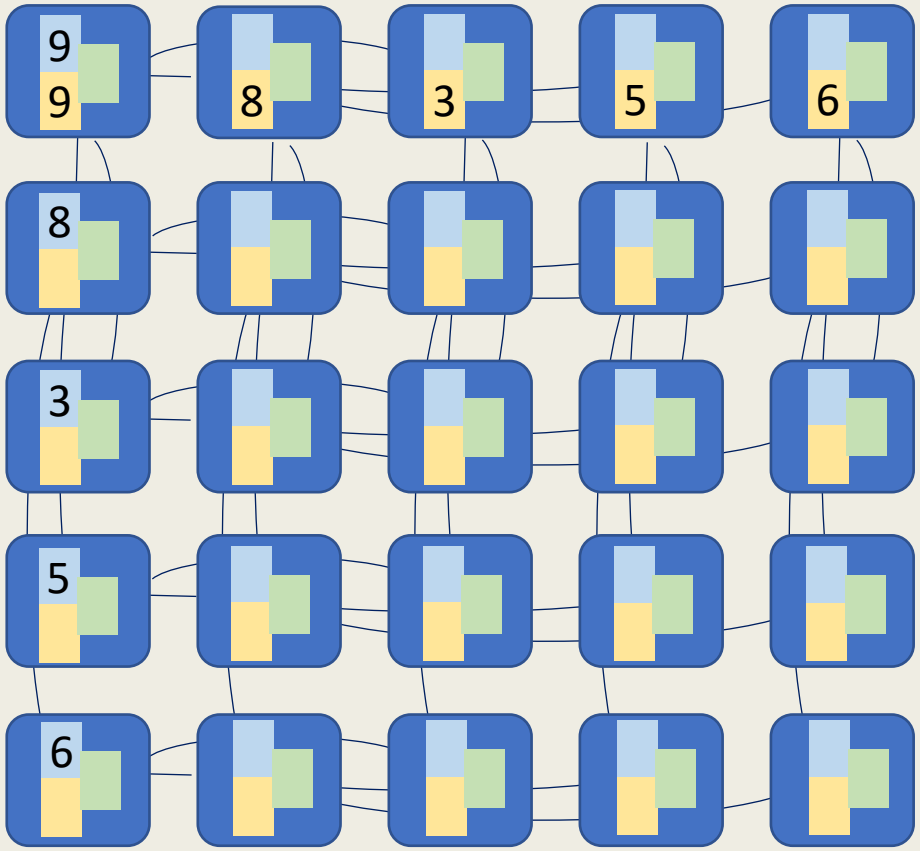
Pro **n** uzlů výška stromu **log n**
Přenesení hodnoty mezi libovolnými dvěma uzly nanejvýš v **2*(log n - 1)** krocích

Enumeration Sort (Mřížka), Algoritmus

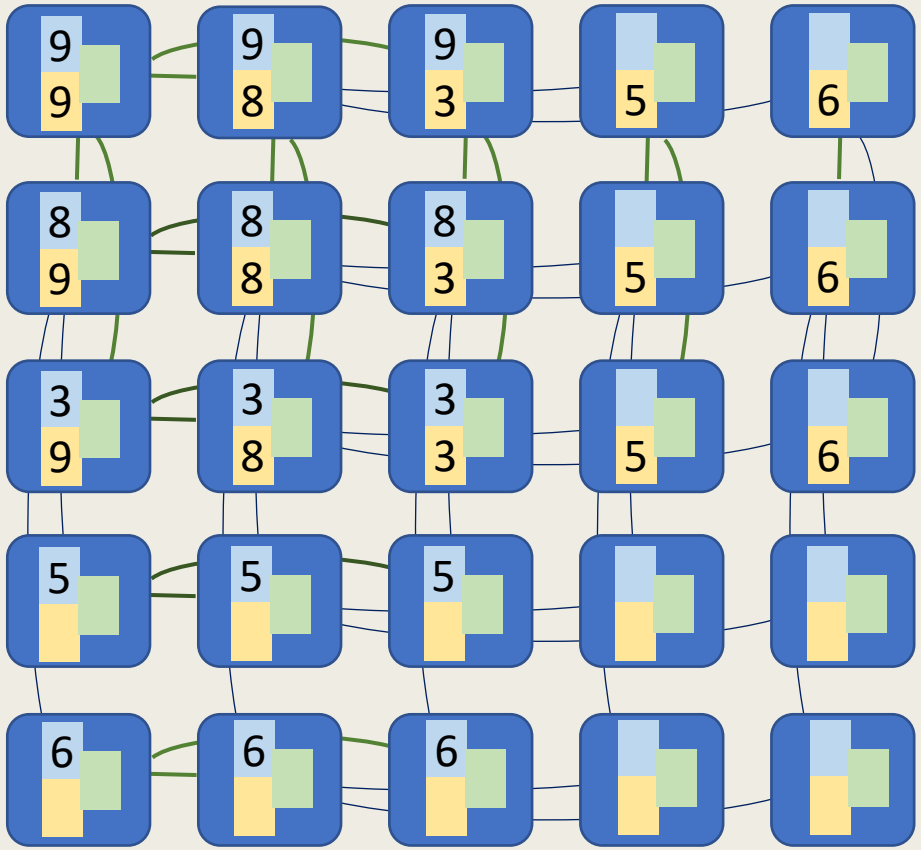
1) každý prvek je porovnán se všemi ostatními pomocí jedné řady procesorů
2) správná pozice prvku je RANK(x_i) = 1 + počet menších prvků
3) každý prvek je zadán na správné místo

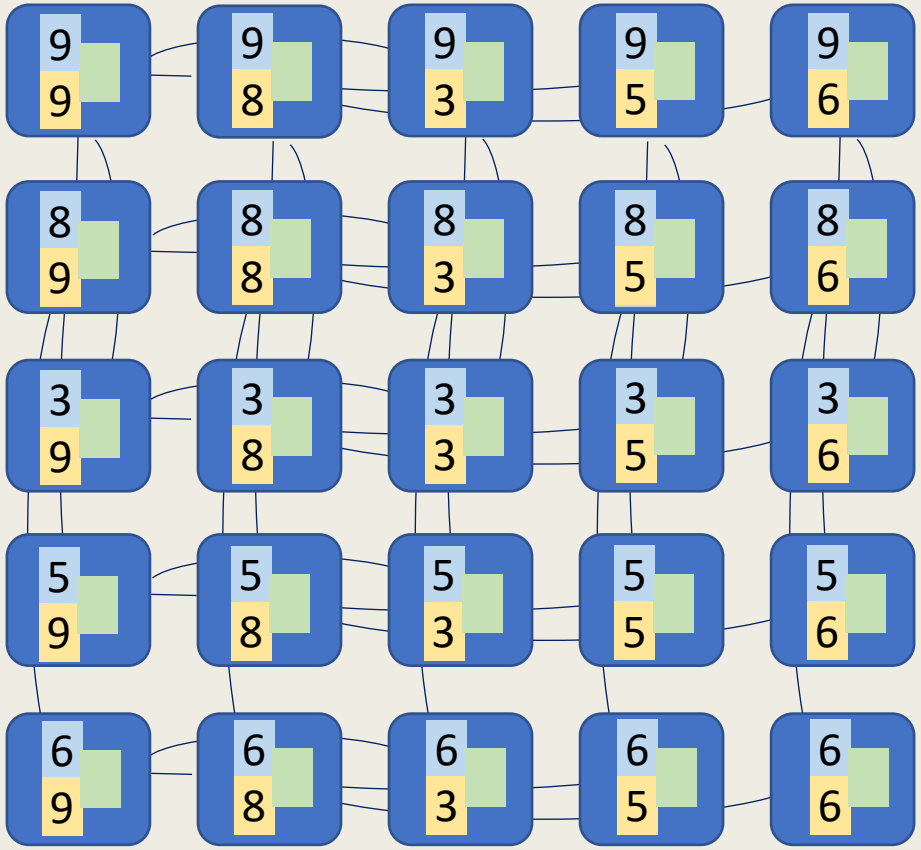
```
1) for i=1 to n do in parallel
        1.1) každý procesor P(i, j) v řadě i získá x_i a x_j
                  (i, j = 1...n) a uloží je do A(i, j) a B(i, j)
                  prvky se roznáší do procesorů
        1.2) if B(i, j) < A(i, j) then RANK(i, j) = 1
                                  else 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(x_i) jako RANK(i,1) += 1
   endfor
3) for i=1 to n do in parallel
        if  RANK(i, 1)=j then x_i je přesunuto z A(i, j) do A(j, 1)
        endif
   endfor
```

Enumeration Sort (Mřížka), 1. fáze

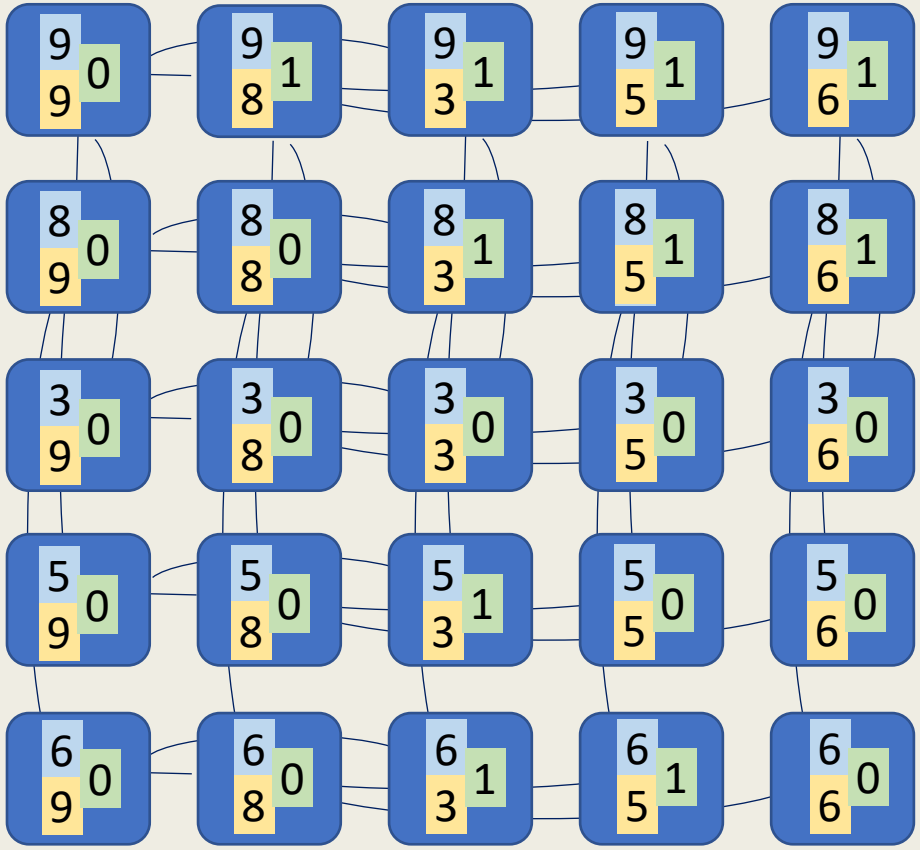


Enumeration Sort (Mřížka), 1. fáze





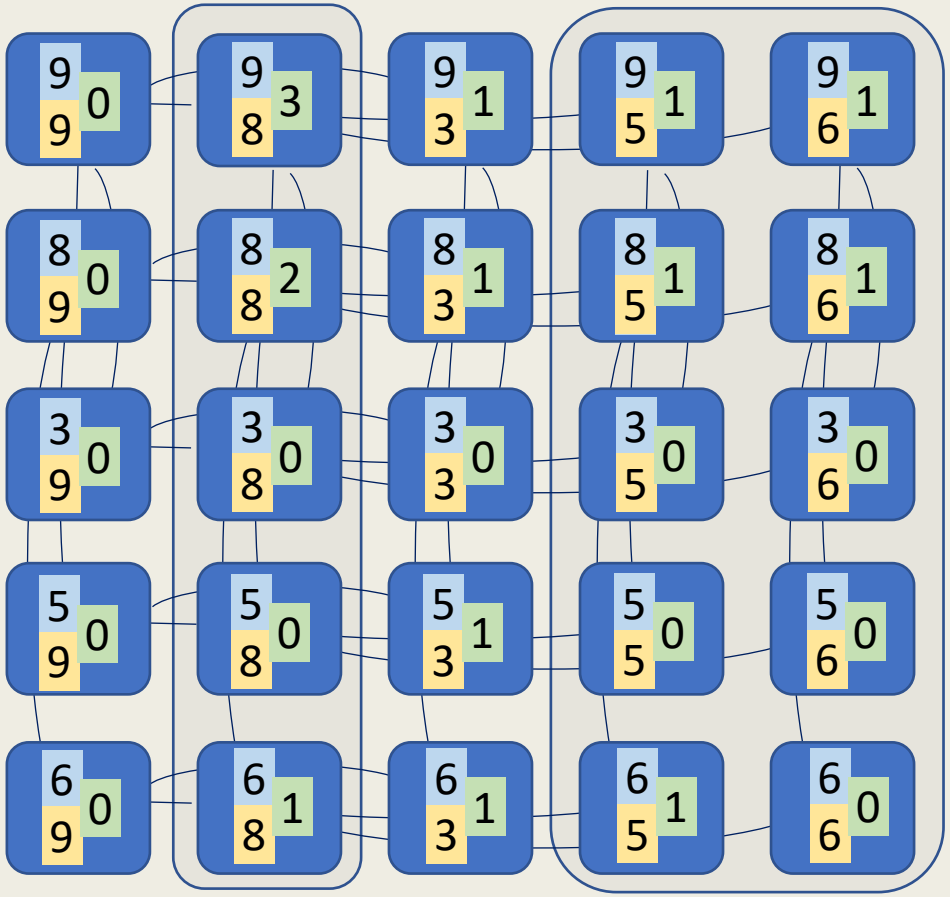
Enumeration Sort (Mřížka), 2. fáze



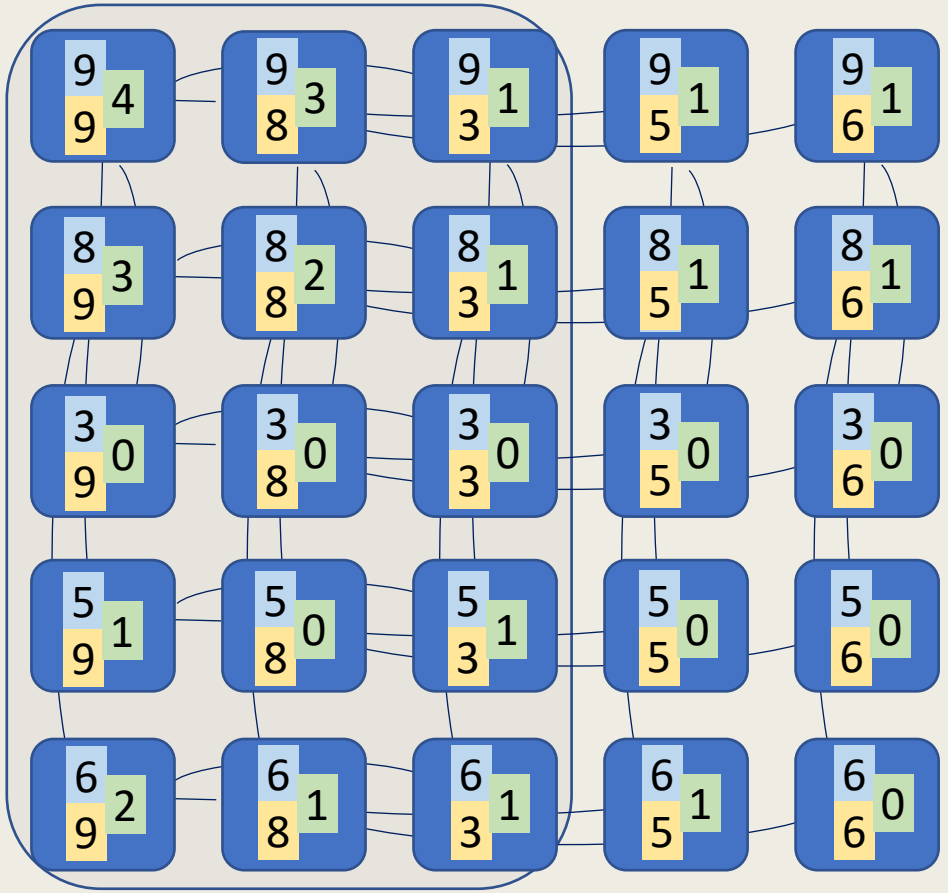

```
if B(i, j) < A(i, j)
 then
   RANK(i, j) = 1
  else
   RANK(i, j) = 0

 for k = ((log n) - 1) downto 1 do
  for j = 2^(k-1) to 2^k-1 do in parallel
        RANK(i,j) += RANK(i,2j)+ RANK(i,2j+1)
  endfor
 endfor
```

Enumeration Sort (Mřížka), 2. fáze

Po sečtení druhé úrovně stromu

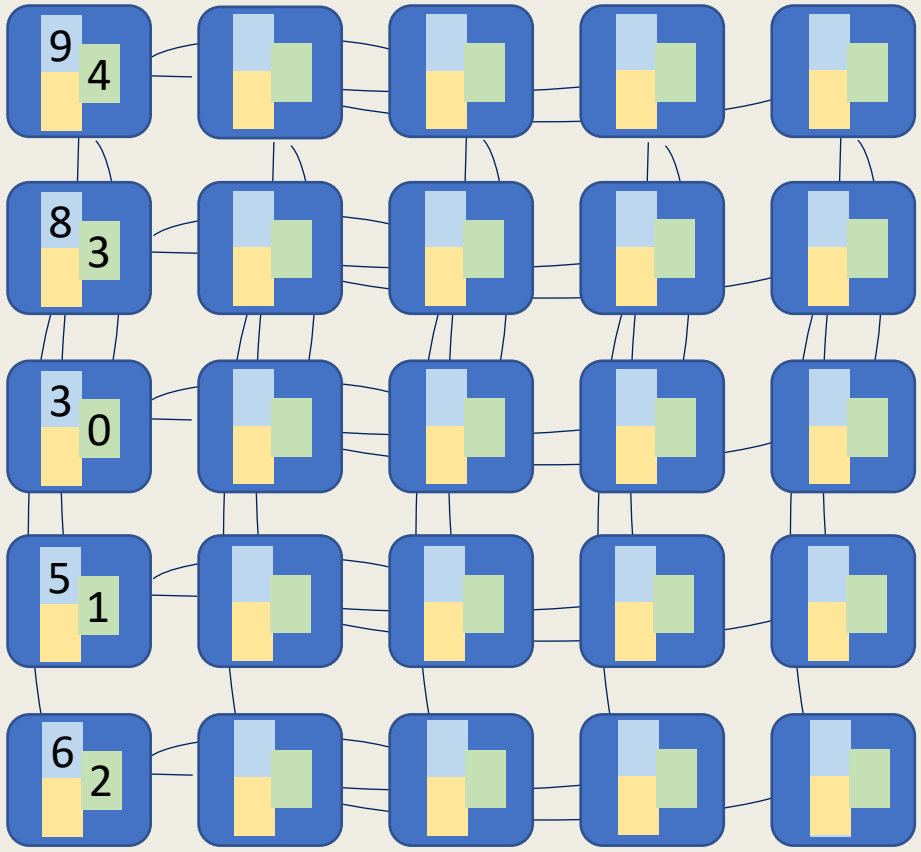


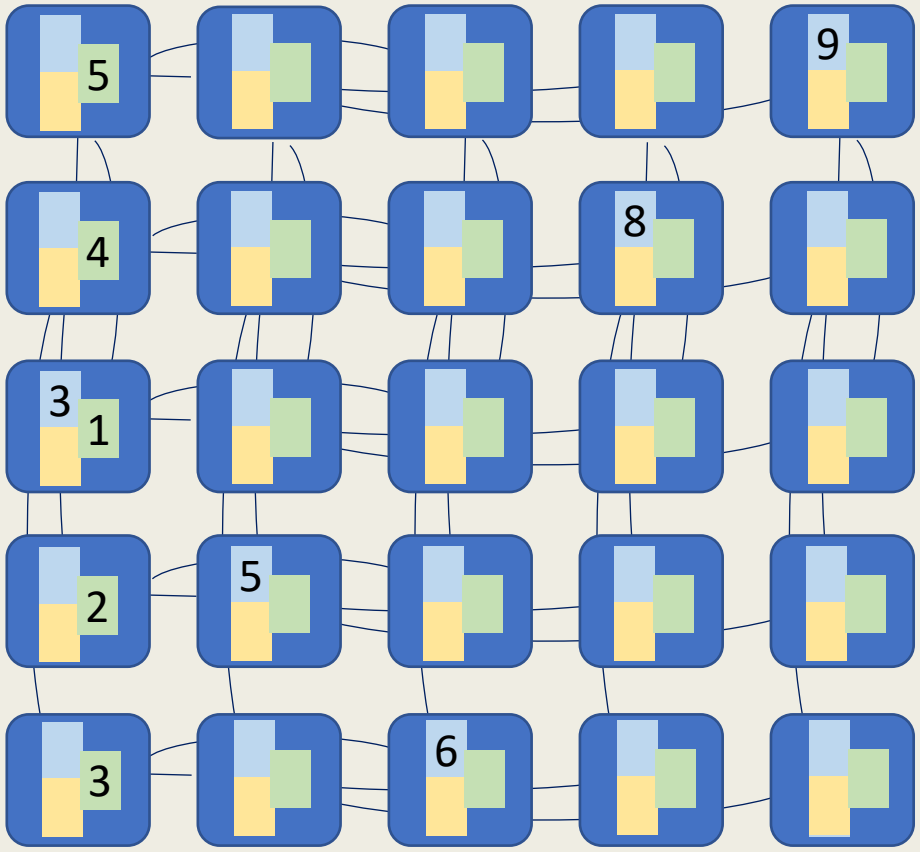
Po sečtení první úrovně a kořenu



Enumeration Sort (Mřížka), 3. fáze

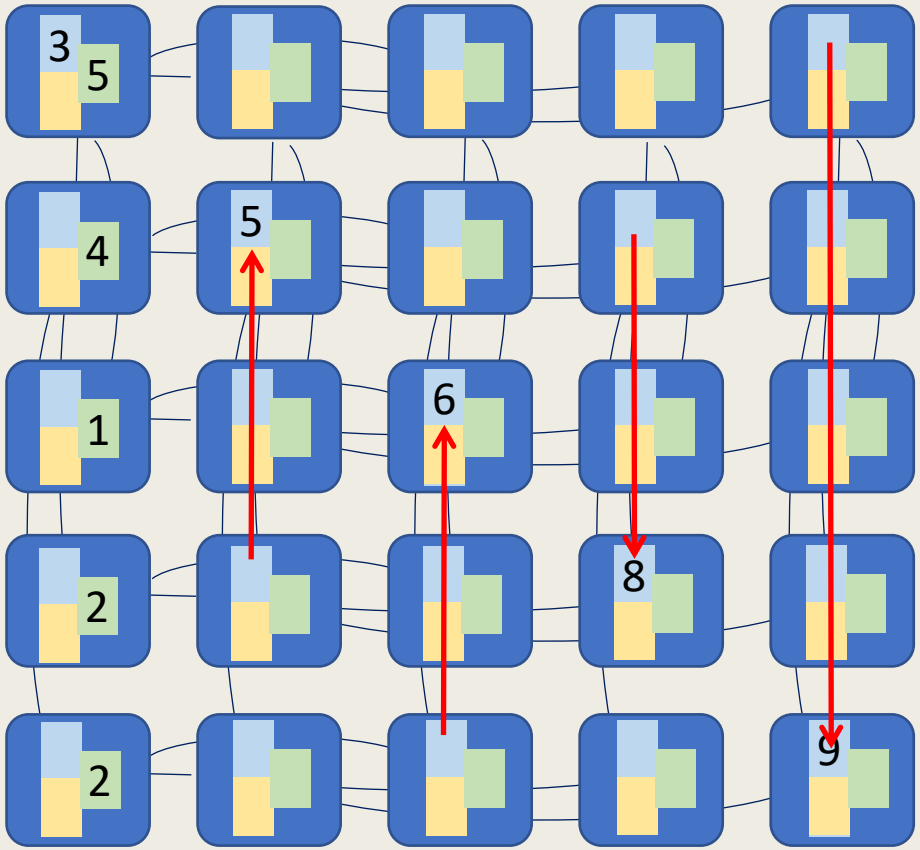
V každém řádku se přesune hodnota registru A v prvním sloupci do sloupce stejného řádku daného Hodnotou registru C. Tedy v prvním řádku se hodnota 9 do 5. sloupce. Stejně tak i pro ostatní řádky. (v nejvíce **log n – 1** krocích).

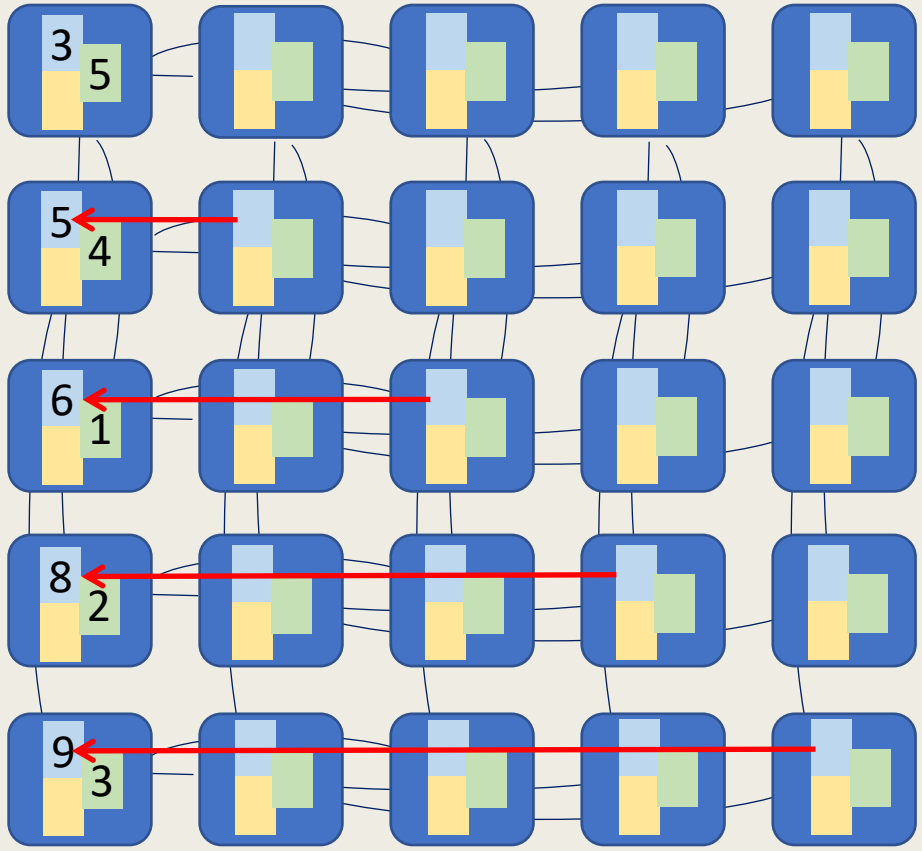




Enumeration Sort (Mřížka), 3. fáze

V každém řádku se přesune hodnoty z aktuálního registru, kam byla hodnota přesunuta, vertikálně
Na pozici diagonály v nejvíce 2*(**log n – 1)** krocích, následně pak všechny přesune do prvního sloupce.
v nejvíce **log n – 1** krocích





Enumeration Sort (Mřížka), analýza

- Analýza
  - $t(n) = O(\log n)$ $p(n) = n^2$
  - $c(n) = O(n^2 . \log n)$ což ***není optimální***

- Diskuse
  - *Algoritmus je extrémně rychlý O(log n), což znamená zrychlení O(n) krát oproti optimálnímu sekvenčnímu algoritmu.*
  - *Žádný paralelní alg. 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)*

Enumeration Sort (Lineární)

- Topologie:

  Lineární propojení n uzlů pro řazení posloupnosti délky n

- Uzel

  může uložit dva prvky do svých registrů X a Y, RANK (označíme C) a výstupní registr Z

  může porovnat A a B a uložit výsledek do registru RANK

- Princip:

  Hodnoty vstupují do registru X postupně po sběrnici

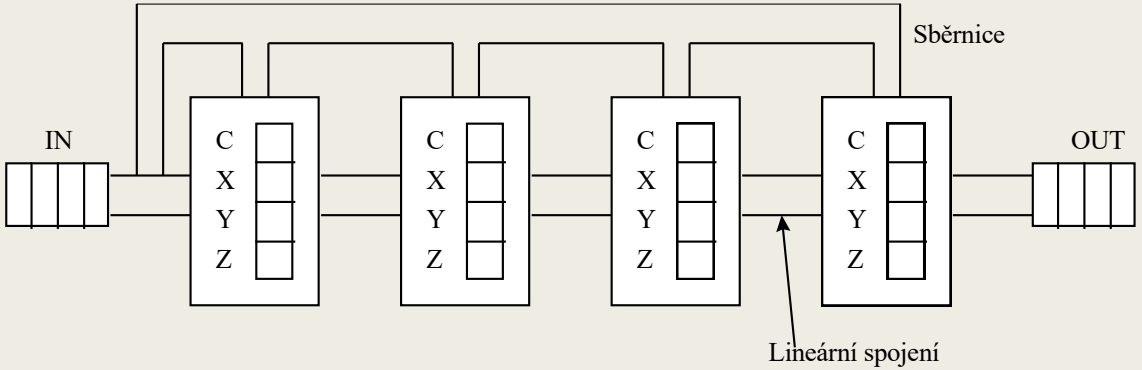
  Hodnoty se přesouvají po registrech Y přes lineární propojení

  Hodnoty jsou umisťovány do registru Z_{ci} po sběrnici

  Každý uzel porovnává hodnoty X a Y, pokud jsou k dispozici, a případně inkrementuje C

  Pokud již byl porovnán každý pár, hodnota z Xi je přesunuta do příslušného Zci

Enumeration Sort (Lineární), Algoritmus



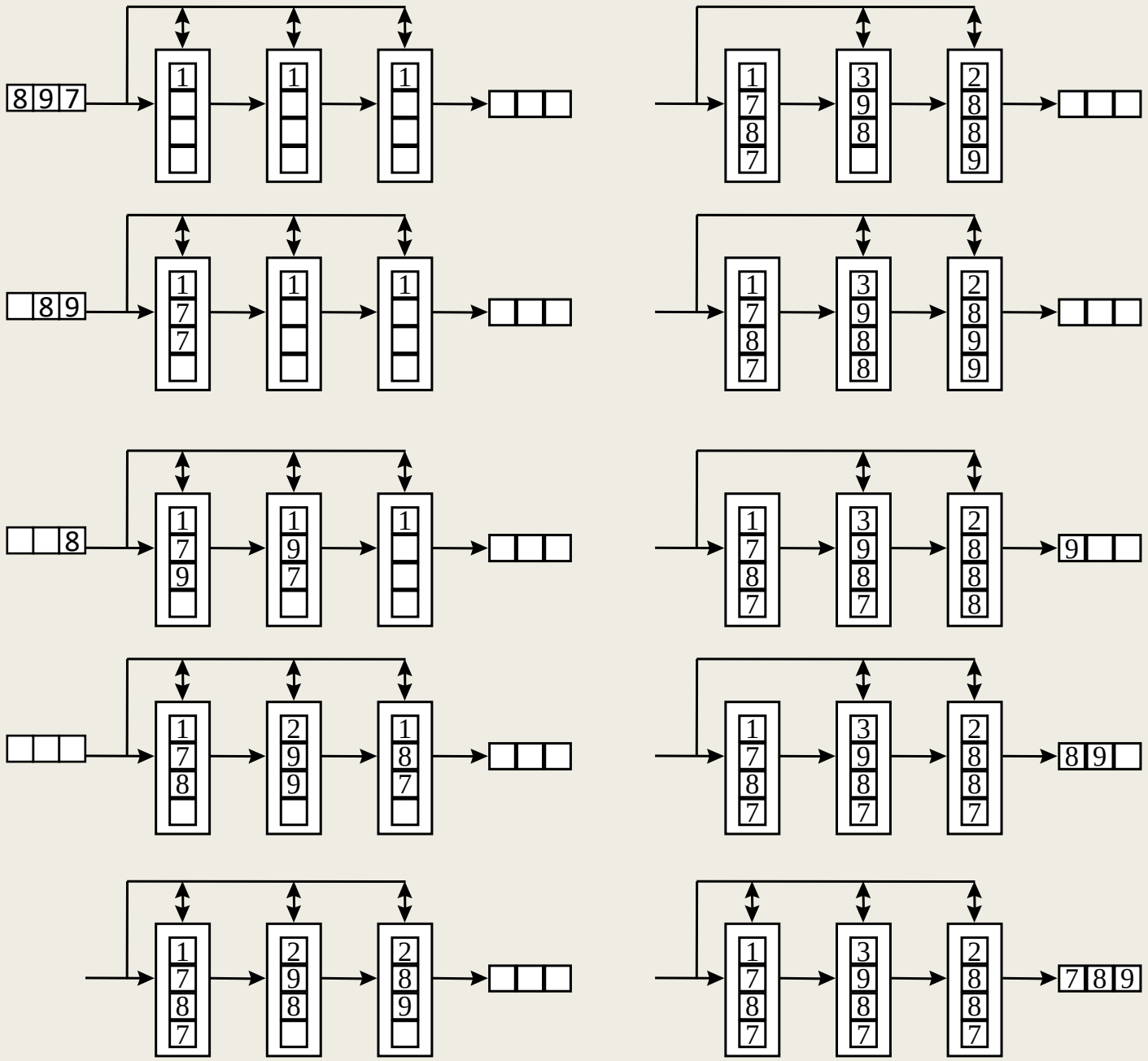
Algoritmus:

1) všechny registry C se nastaví na hodnotu 1
2) následující činnosti se opakují 2n krát 1 <= k <= 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 $\underline{n}$ cyklech procesory posouvají obsah svých registrů doprava a procesor P_n produkuje seřazenou posloupnost


8 9 7
1
1
1
1 7 8 7
3 9 8
2 8 8 9
8 9
1 7 7
1
1
1 7 8 7
3 9 8 8
2 8 9 9
8
1 7 9
1 9 7
1
1 7 8 7
3 9 8 7
2 8 8 8
9
1 7 8
2 9 9
1 8 7
1 7 8 7
3 9 8 7
2 8 8 7
8 9
1 7 8 7
2 9 8
2 8 9
1 7 8 7
3 9 8 7
2 8 8 7
7 8 9

Enumeration Sort (Lineární), Analýza

- Analýza
  - *Bod 1) je v konstantním čase*
  - *bod 2) trvá 2n cyklů,*
  - *bod 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

- Topologie

  Stromová topologie

- Uzly

  Listové uzly obsahují řázenou posloupnost, do nelistových je vybírána hodnota z potomkovských (umí porovnávat tyto hodnoty), kořenový uzel posílá hodnoty na výstup

- Princip

  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

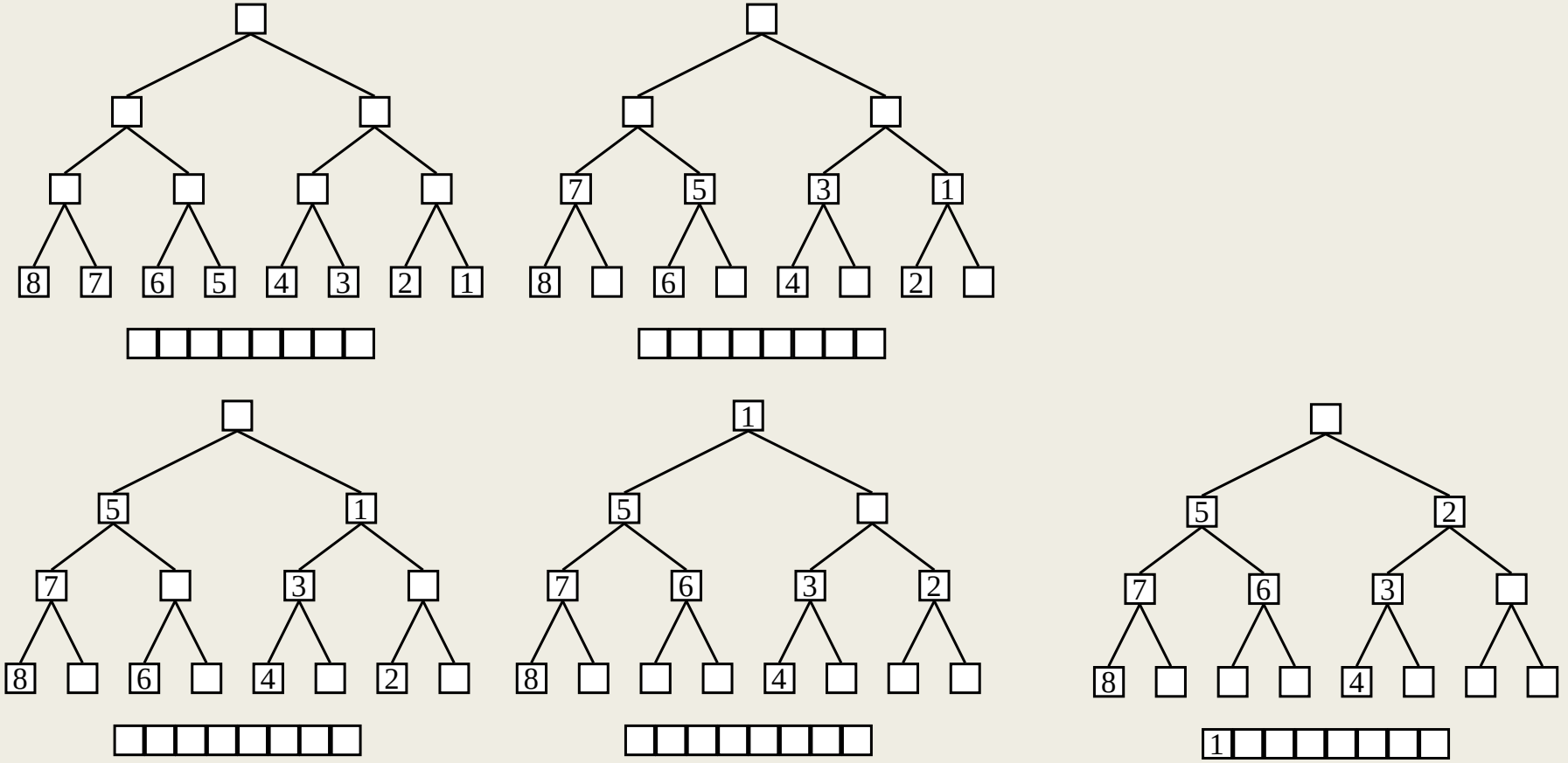
  každým dalším krokem se získá další nejmenší prvek

Minimum Extraction sort, Algoritmus

```
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 childs
                 endif
              endif
            endif
       endif
     endfor
   endfor
```

Zde je třeba pozorně synchronizovat

Minimum Extraction sort, Příklad



Minimum Extraction sort, 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

- **Topologie**

  řazení stromem procesorů s m listovými procesory $n = 2^m$ (tedy m=log n)

  strom obsahuje 2m - 1 procesorů, takže p(n) = (2. log n) - 1

- **Uzly**

  každý listový procesor obsahuje **n/m** řazených prvků a umí je seřadit optimálním sekvenčním algoritmem

- **Princip**

  každý nelistový procesor umí spojit dvě seřazené posloupnosti optimálním sekvenčním algoritmem

Bucket Sort, Algoritmus

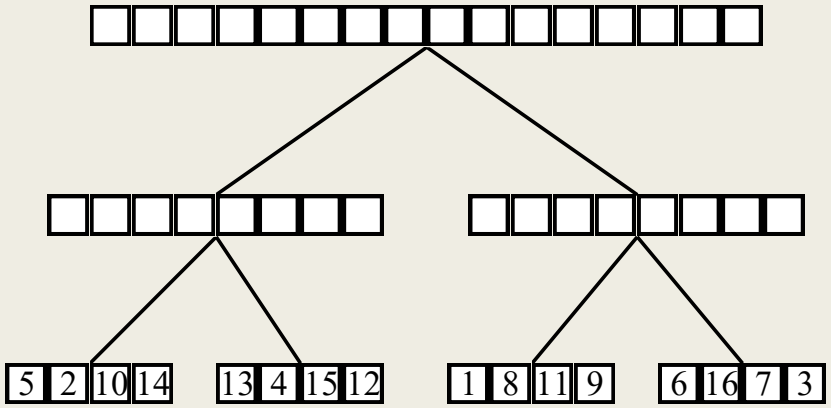
Řazené prvky se rovnoměrně rozdělí mezi listové procesory

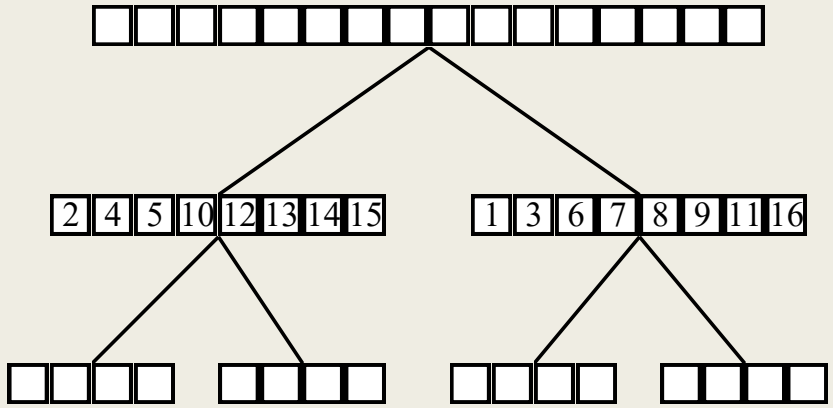
Každý list seřadí svou posloupnost

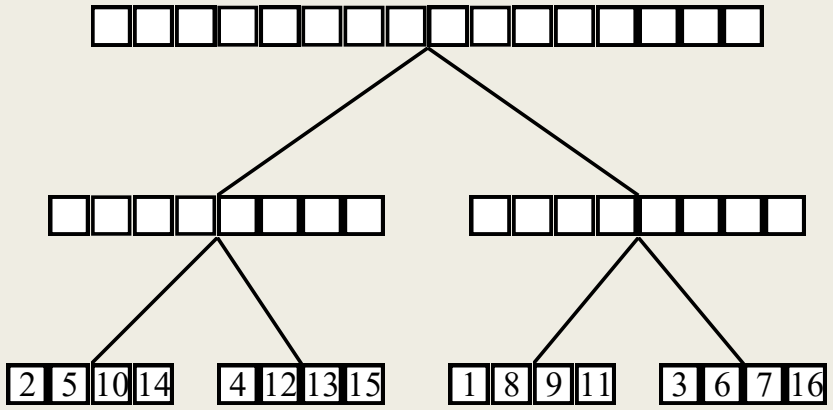
```
for j = 1 to log m do
      for all processors at level (log m)-j do in parallel
            procesor spojí posloupnosti svých synů
      endfor
endfor
```

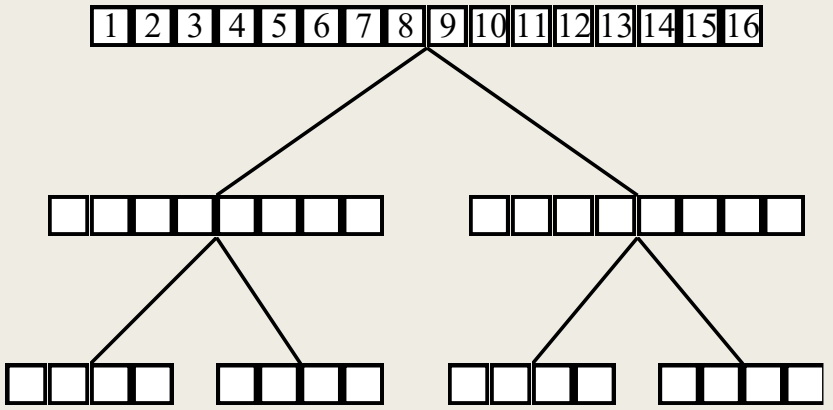
Kořenový procesor uloží výslednou posloupnost do paměti

Bucket Sort, příklad









Bucket Sort, Analýza

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 $\frac{n}{2^i}$. Při použití *merge* každá j-tá iterace zabere $\frac{kn}{2^i}$ kroků, kde k je konstantní. Krok 3 tedy trvá

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

4.Třída výpisu výsledné posloupnosti na výstup je je O(n)

Celkově tedy

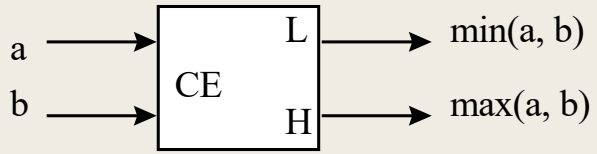
t(n) = O(n) *p(n) = O(log n)*

c(n) = O(n.log n) □ *což je optimální*

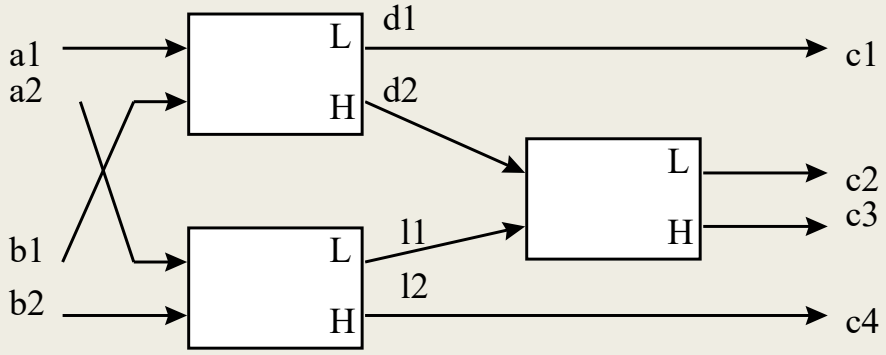
Odd – Even Merge Sort

- Algoritmus založený na slučování
- Topologie
  - *Speciální topologie pro slučování. Jednotka CE uspořádá dvě hodnoty na vstupech na výstup*
  - *Spojováním jednotek do bloků dokáže uspořádat dvě uspořádané vstupní posloupnosti v jednu*

Síť 1x1



Síť 2x2



Odd – Even Merge Sort, obecné bloky

- K sestavení obecného bloku n x n potřebujeme
  - *Dva bloky n/2 x n/2*
  - *N-1 CE bloků*
- 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 $\{c_1, c_2, ...\}$

$c_1 = d_1$

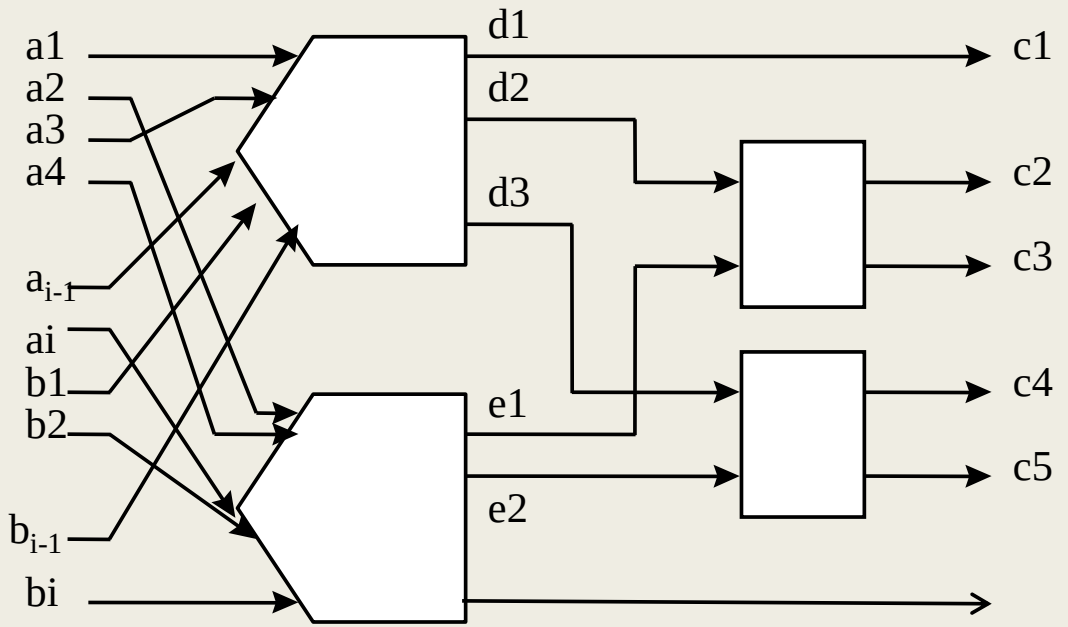
...

$c_{2j} = min\ (d_{j+1}, e_j)$

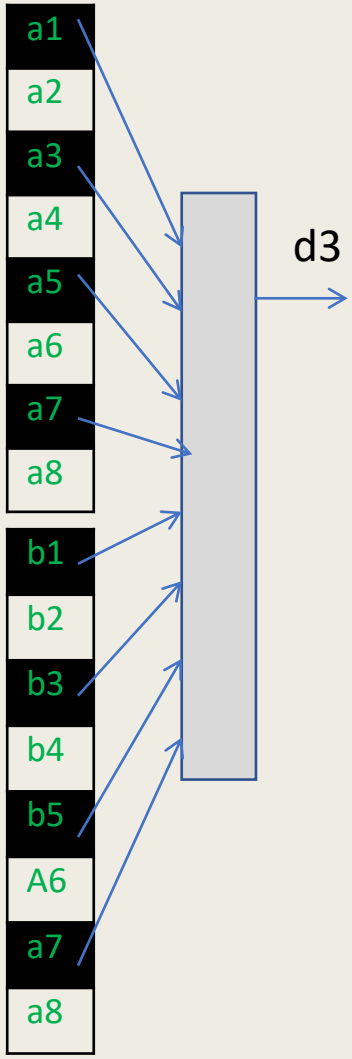
$c_{2ij1} = max\ (d_{j+1}, e_j)$

...

$c_{2n} = e_n$



Odd – Even Merge Sort, ověření funkčnosti



Prvek di je i tý prvek z n prvků ve výsledné posloupnosti první řadičky. Tedy existuje
i-1 menších lichých prvků v celkové posloupnosti a n-i větších lichých prvků v posloupnosti. Jelikož každý lichý prvek má sudého ‚sourozence', který v této řadičce nebyl obsloužen, je 2*(i-1) menších prvků celkem a 2*(n-i) větších prvků celkem a tedy prvek di musí být buď na pozici 2*(i-1) nebo 2*(i-1)+1

<u>d3 může být</u>
a1 s d1=b1 a d2=b3
pak jej předchází b1,b2,b3 a možná i b4 (4. nebo 5.)
a3 s předchůdci a1 a b1
pak jej předchází a1,a2,b1 a možná i b2 (4. nebo 5.)
a5 s d1=a1 a d2=a3
pak jej předchází a1,a2,a3 a a4 (4.)

… obdobně může být i b1, b3 a b5

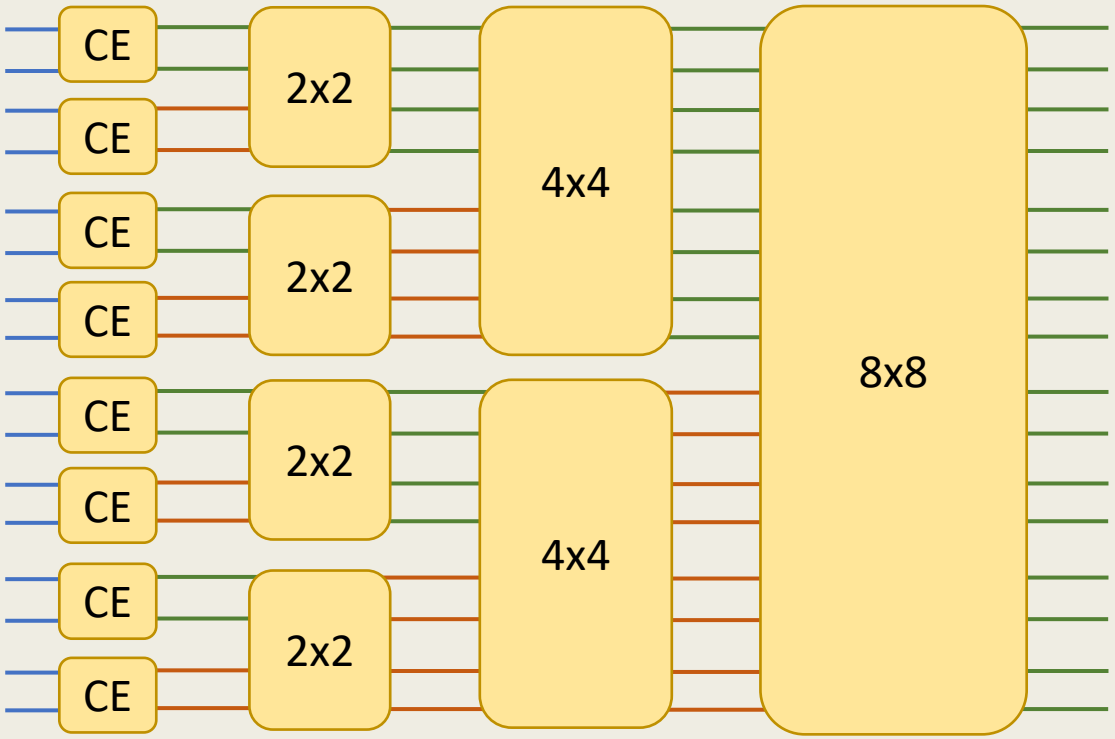
Odd – Even Merge Sort, struktura

- Řadíme po fázích

  dvojice vstupů v uspořádanou posloupnost délky 2 (CE)

  dvojice uspořádaných dvojic v uspořádanou čtveřici (Síť 2x2)

  … dvojice uspořádaných 2^{n-1}-tic na uspořádanou 2^n tici



Odd – Even Merge Sort, analýza

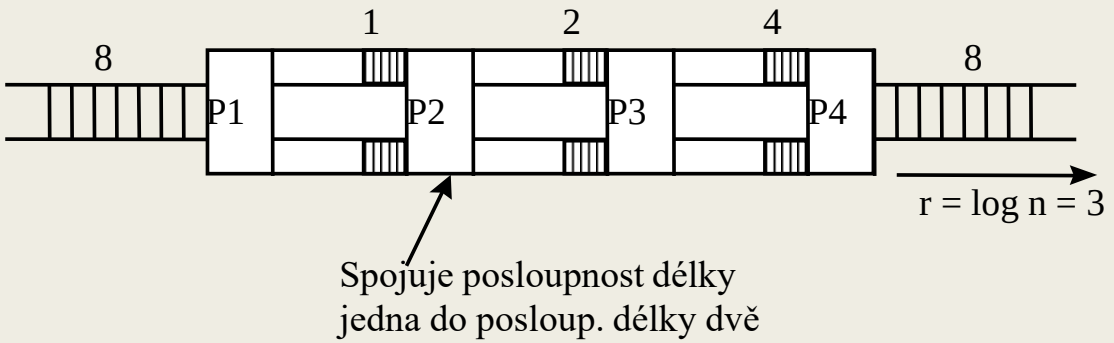
- Počet CE jednotek roste v řadě 1, 3, 9, 25 …
- Obecně $n_i=2*(n_{i-1})+2^{(i-2)}-1$ (resp p(2n) = 2p(n)+(n-1) = 1+n*log n)

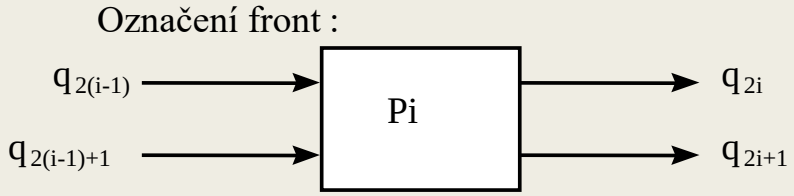
  n1=**1**, n2=2+1=**3**, n3=2*3+3=**9**, n4=2*9+7=**25**, n5=2*25+15=**65**
- Řadíme posloupnost o délce $n=2^m$
  - *1.fáze potřebuje 2^{m-1} CE*
  - *2.fáze potřebuje 2^{m-2} sítí 2x2 po 3 procesorech*
  - *3.fáze 2^{m-3} sítí 4x4 po 9 procesorech*
  - *4.fáze 2^{m-4} sítí po 25 procesorech*
  - *atd.*
  - *Suma je $O(n* \log^2 n)$ // fází je log n*
- Časová složitost (počet kroků v jednotlivých fázích ke řada 1,2,3 … log n)
  - *$t(n) = O(m^2) = O(\log^2 n)$ // součet řady*
- Cena
  - *$c(n) = O(n*\log^4 n)$ // což **není optimální***

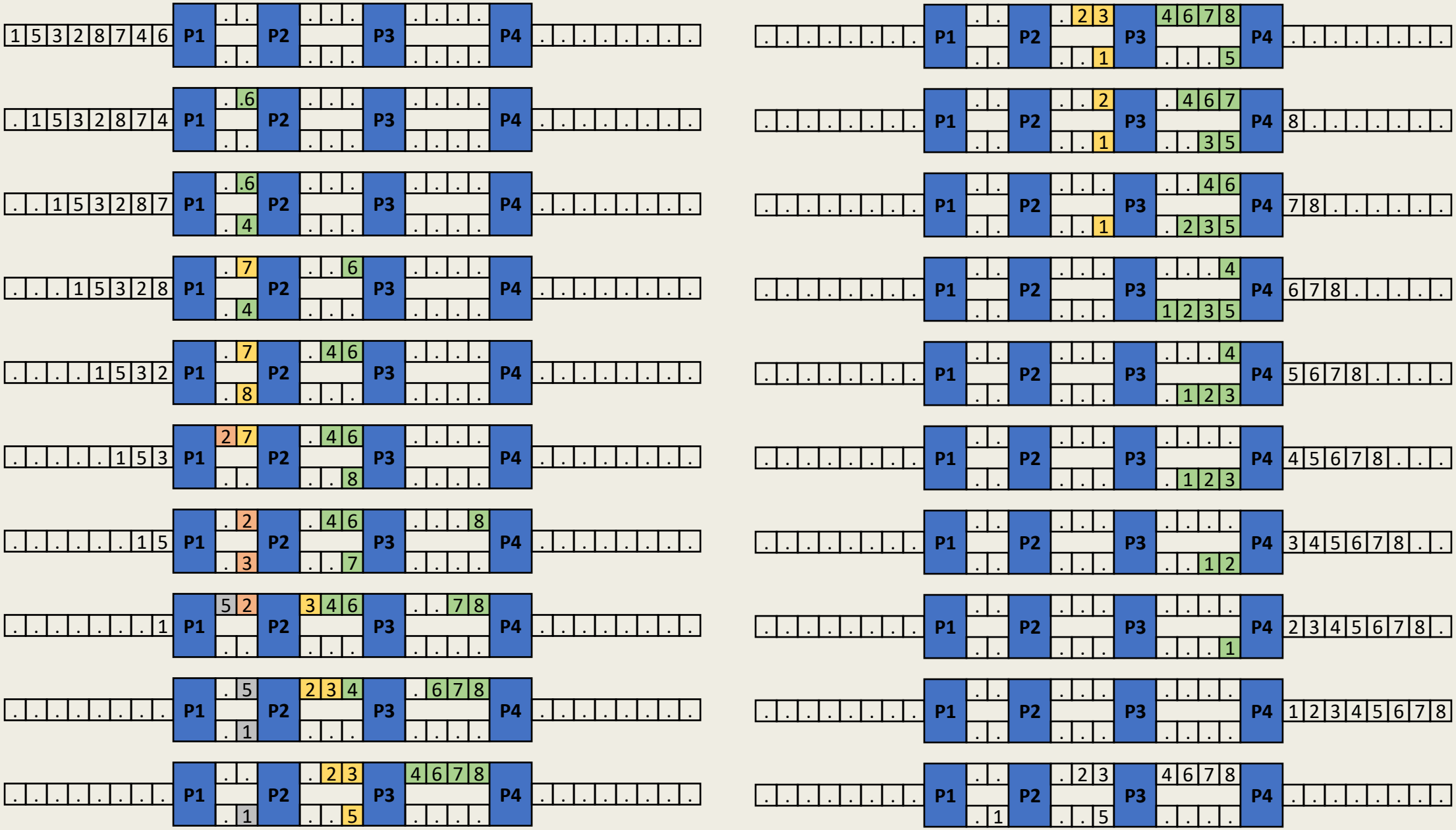
$1+16*\log_2 16$

Pipeline Merge Sort

- Topologie
  Lineárně propojené uzly
- Funkčnost uzlů
  Má dvě vstupní fronty a jednu výstupní. Spojí dvě seřazené posloupnosti délky ***n*** *v jednu seřazenou posloupnost délky* ***2n***
- Princip
  Postupně slučuje seřazené posloupnosti délky 2^i do seřazené posloupnosti délky 2^{i+1} pro i= $0 \ldots \log n$ kde n je délka vstupní posloupnosti






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

Pipeline Merge Sort, 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

  $$1 + \sum_{j=0}^{i-2}(2^j+1) = 2^{i-1} + i - 1$$ *(tj. v 3., 6., 11. ...)*

  a skončí v cyklu

  $$(n-1) + 2^{i-1} + i - 1$$

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

Median Finding and Splitting

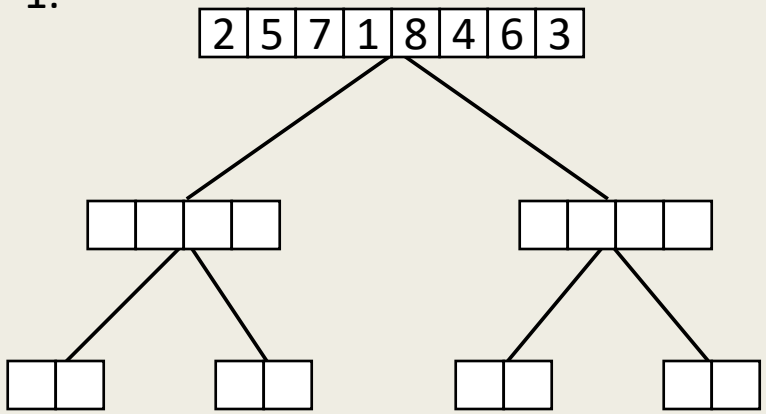
- stejná architektura jako Bucket Sort
  - strom procesorů s m listy, **$n = 2^m$, $m = \log n$**
  - procesor na úrovni i zpracovává $n/2^i$ prvků
  - každý list umí optimální sekvenční sort
- nový požadavek
  *každý nelistový procesor umí nalézt medián v optimálním čase (např. algoritmus Select s **O(n)** složitostí)*

Median Finding and Splitting, Algoritmus

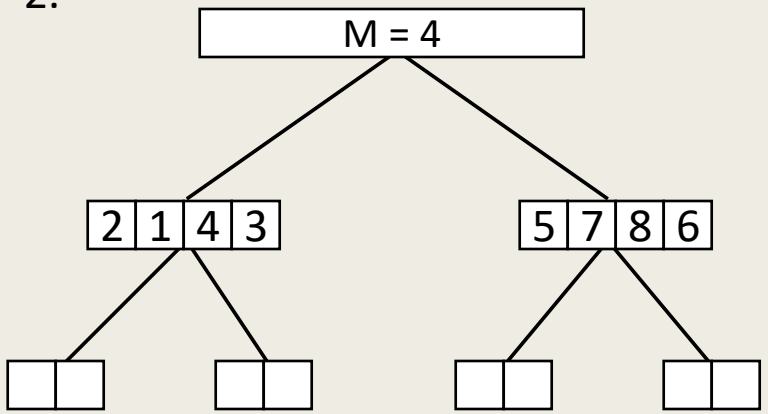
```
1.Kořen načte řazenou sekvenci S
2.for i=0 to (log m) - 1 do
        for all processors at level i do in parallel
        2.1) Nalezni ve své sekvenci medián M (sekvenčně)
        2.2) Pro každý prvek x této sekvence
                if x < M then pošli x levému synovi
                        else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

Median Finding and Splitting, Příklad

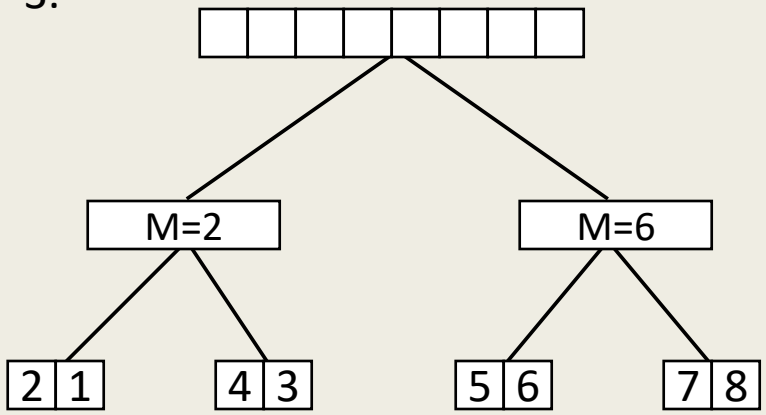
1.



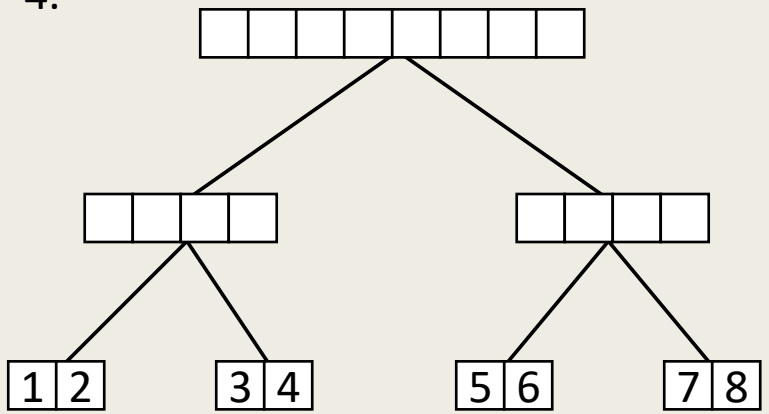
2.



3.



4.



Median Finding and Splitting, Analýza

1.První krok má složitost 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:

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

3.každý procesor řadí posloupnost délky n/log n což mu trvá

$O(\frac{n}{logn} * \log(n/(\log n)))$ a výstup uloží v čase $O(\frac{n}{logn})$, takže složitost kroku 3 je O(n)

t(n) = O(n)
c(n) = O(n.log n)

p(n) = O(log n)
☐ **což je optimální**