Sortovacie siete

- **PRAM** = **Parallel Random Access Machine**
  - model s (neobmedzenou) zdieľanou pamäťou
  - **synchrónny model** – spoločné hodiny
  - **SIMD** – v každom kroku sa vykonáva rovnaká inštrukcia
  - vstup aj výstup v zdieľanej pamäti
  - každý procesor má ID a pozná celkový počet procesorov
  - pamäťové modely: EREW, CREW, CRCW (common, arbitrary, priority)
  - zrýchlenie: $S_p(n) = T^*(n) / T_p(n)$
  - cena: $C_p(n) = p.T_p(n)$

Work-time zápis algoritmov

- WT presentation framework:
  - zápis **abstrahujúci** od detailov PRAM implementácie
  - **neviaže sa na fixný počet** použitých procesorov
  - **postupnosť časových jednotiek**, kde každá môže obsahovať ľubovoľne veľa konkurentných operácií
  - akoby sekvenčný algoritmus s konštrukciou „paralelný for“, ktorá vykoná zadaný príkaz konkurentne pre všetky hodnoty **i**

```
for i := 1 to N pardo
    prikaz;
```

WT-charakteristiky

- **T(n)** – počet časových jednotiek, ktoré vykoná algoritmus na vstupe veľkosti n
- **W(n)** – celkový počet operácií, ktoré vykoná algoritmus na vstupe veľkosti n (práca algoritmu)

```
for i:=1 to n pardo
    A[i] := i;
```

Počet časových jednotiek: **1**
Počet vykonaných operácií: **n**

- **WT-rozvrhovanie** – každý WT-program vieme „skompilovať“ na program pre p-procesorový PRAM s časom behu: $T_p(n) <= W(n)/p + T(n)$

WT a optimalita

- **T(n)** – počet časových jednotiek, ktoré vykoná algoritmus na vstupe veľkosti n
- **W(n)** – celkový počet operácií, ktoré vykoná algoritmus na vstupe veľkosti n (práca algoritmu)
- **Optimalita:**
  - optimálny algoritmus: $W(n) \in O(T^*(n))$
  - WT-optimálny algoritmus:
    $W(n) \in O(T^*(n))$ a $T(n)$ sa nedá zlepšiť
  - ak je algoritmus optimálny, potom $S_p(n) \in \Theta(p)$ pre $p \in \Theta(T^*(n)/T(n))$

Bitonická postupnosť

- x[0], x[1], …, x[n-1] je bitonická postupnosť práve vtedy, ak existujú i, j také, že:
  - x[i mod n] <= x[(i+1) mod n] <= …<=x[j mod n]
  - x[(j+1)mod n]>=x[(j+2)mod n]>= …>=x[(i+n-1) mod n]
- Inak:
  - existuje také cyklické posunutie postupnosti x, že postupnosť je najprv neklesajúca a potom nerastúca
- Príklad: 18 5 4 8 13 94 50 32 30 18 5 4 8 13 94 50 32 10

4 8 13 94 50 32 30 18 5

Unikátne prekríženie

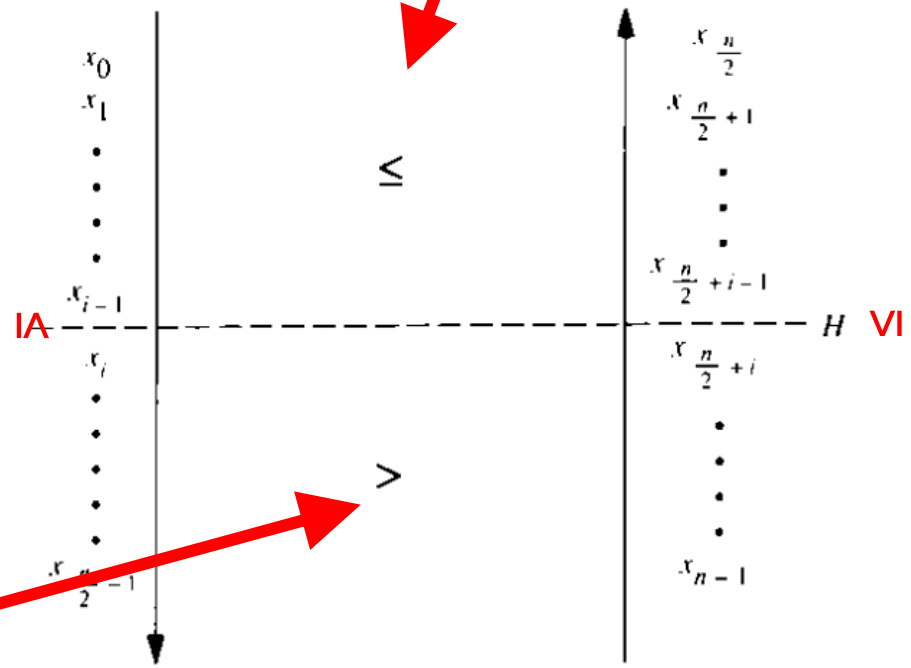
- Ak x je bitonická postupnosť s párnym počtom prvkov taká, že
  - x[0] <= x[1] <=…<=x[n/2-1]
  - x[n/2] <= x[n/2+1] <=…<=x[j], x[j+1]>=…>=x[n-1]

potom x možno rozdeliť na 2 oblasti:

Každá hodnota vľavo je menšia alebo rovná ako každá hodnota vpravo.

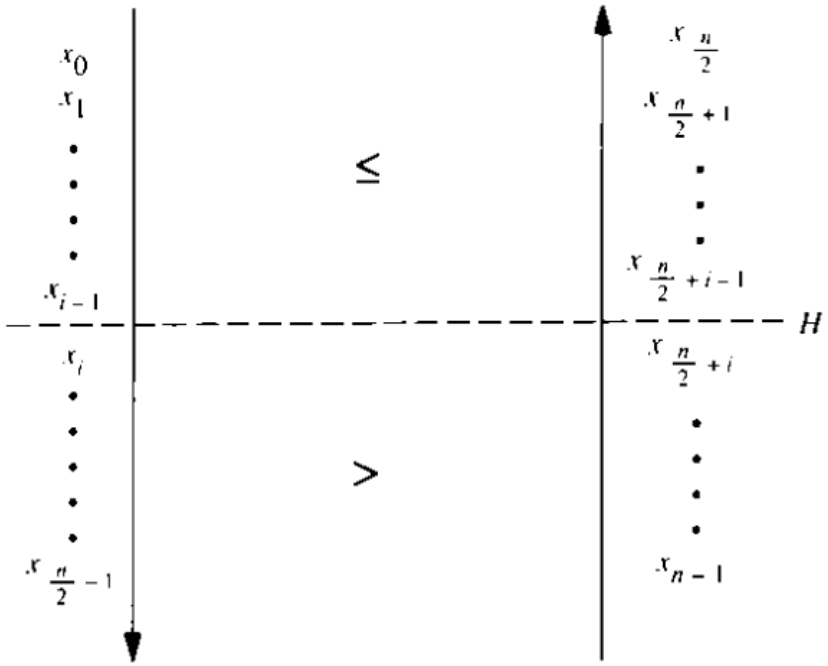
Vľavo: Každá hodnota nad H je menšia alebo rovná ako každá hodnota pod H.

Každá hodnota vľavo je väčšia ako každá hodnota vpravo.



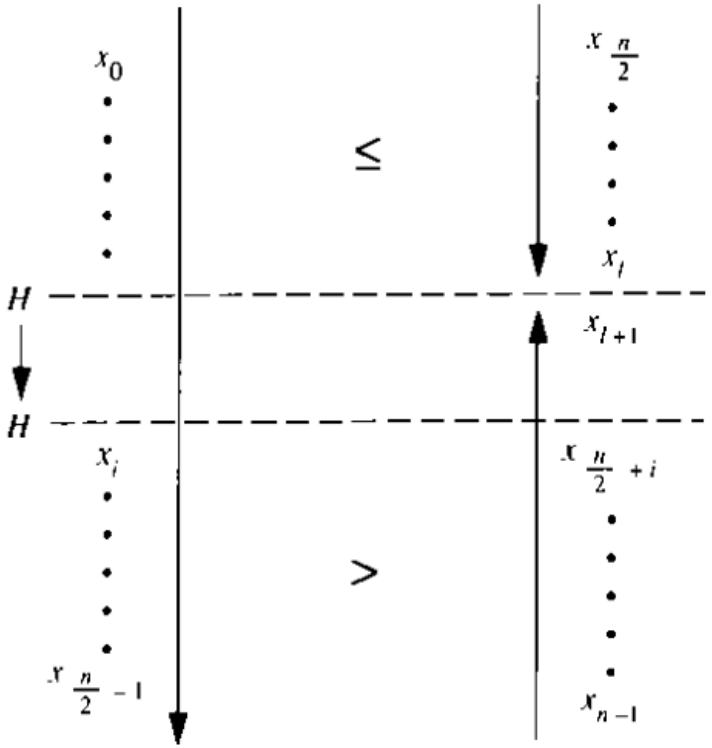
Unikátne prekríženie

- Unique cross-over property
- nech i je prvé také, že x[i] > x[i+n/2], potom H je medzi x[i-1] a x[i]
- všetky požiadavky unikátneho prekríženia sú splnené



Unikátne prekríženie všeobecne

- Každá bitonická postupnosť párnej dĺžky má vlastnosť unikátneho prekríženia.
- Dôkaz:
  - predpoklad: i = 0 a j > n/2
  - H nie je „nad“ x[j] - začneme s H medzi x[j] a x[j+1] a pokiaľ x[i] <= x[i+n/2], posúvame ju „dole“
  - cyklické posunutie postupnosti zachováva vlastnosť unikátneho prekríženia



Využitie vlastnosti prekríženia

Nech x je bitonická postupnosť párnej dĺžky a súčasne

- L[i] = min(x[i], x[i+n/2]) pre i=0, …, n/2-1
- R[i] = max(x[i], x[i+n/2]) pre i=0, …, n/2-1

potom L aj R sú bitonické a každý prvok L je menší ako všetky prvky R

Dôkaz:

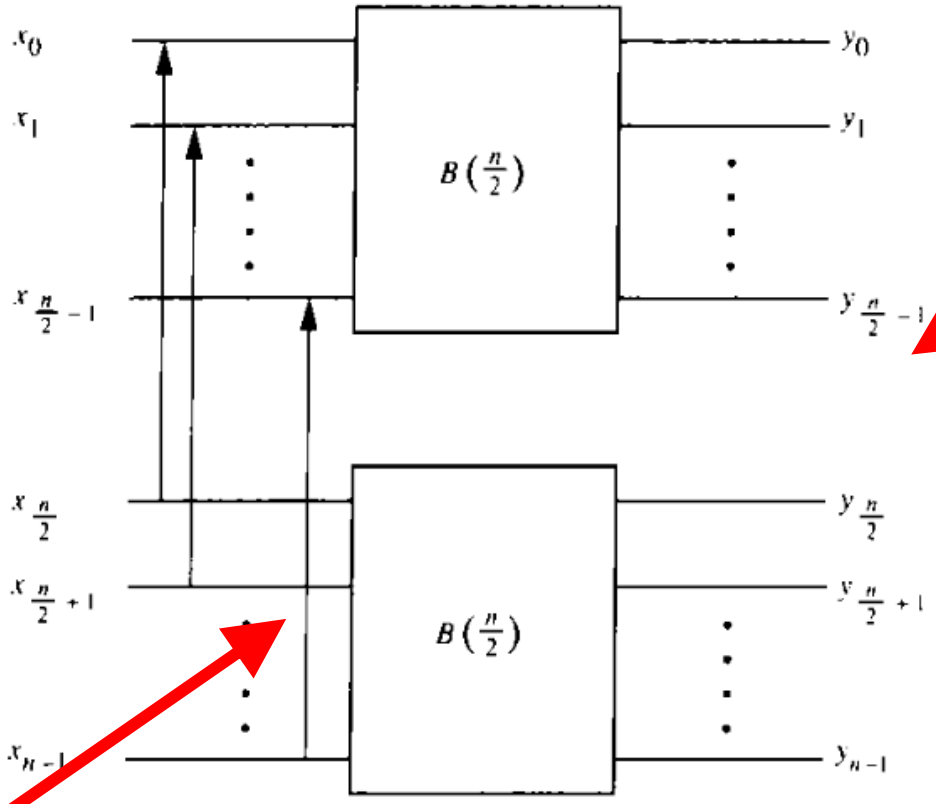
- stačí si všimnúť, že obe postupnosti sú súvislé podpostupnosti originálnej bitonickej postupnosti (s cyklickým posunutím)
- z unikátneho prekríženia aj každý prvok L je menší ako ktorýkoľvek prvok z R (pozri obrázok ...)

Bitonické triedenie

- **Vstup**: bitonická postupnosť x
- **Výstup**: usporiadaná postupnosť x
- **Algoritmus**:

```
for i := 0 to (n/2) - 1 pardo
begin
    L[i] := min(x[i], x[i+n/2]);
    R[i] := max(x[i], x[i+n/2]);
end;
rekurzívne (a paralelne) usporiadaj bitonické postupnosti L a R
x := L | R
```

Bitonické triedenie a komparátory



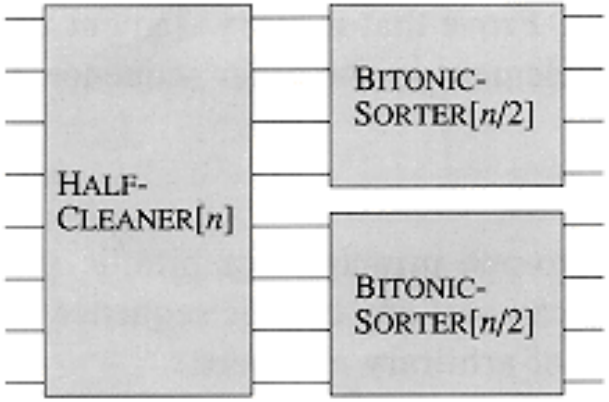
B(n) – triediaca sieť pre n-prvkovú bitonickú postupnosť

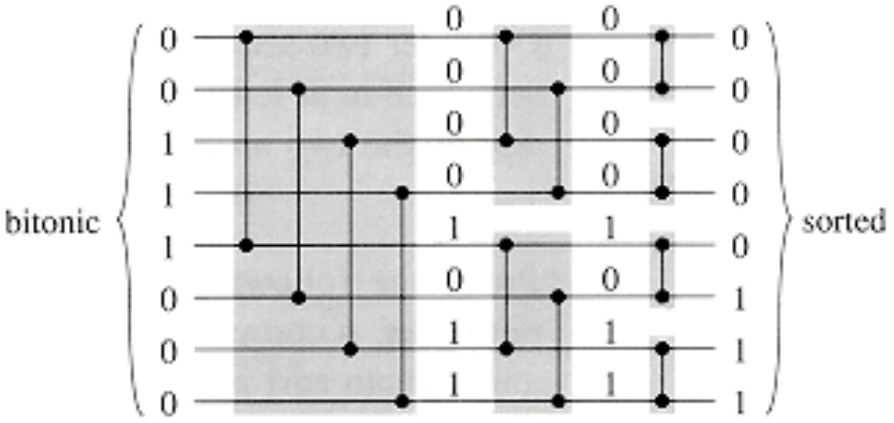
Komparátor vezme 2 hodnoty, porovná ich a ak treba, tak ich navzájom vymení (menšia v smere šípky)

B(n) sa skladá z dvoch **B(n/2)** a **n/2** komparátorov

Bitonické triedenie a komparátory

- Každý komparátor v triediacej sieti zodpovedá jednotke práce
- Komparátory porovnávajú hodnoty paralelne – každá úroveň komparátorov predstavuje časovú jednotku





Bitonické triedenie - analýza

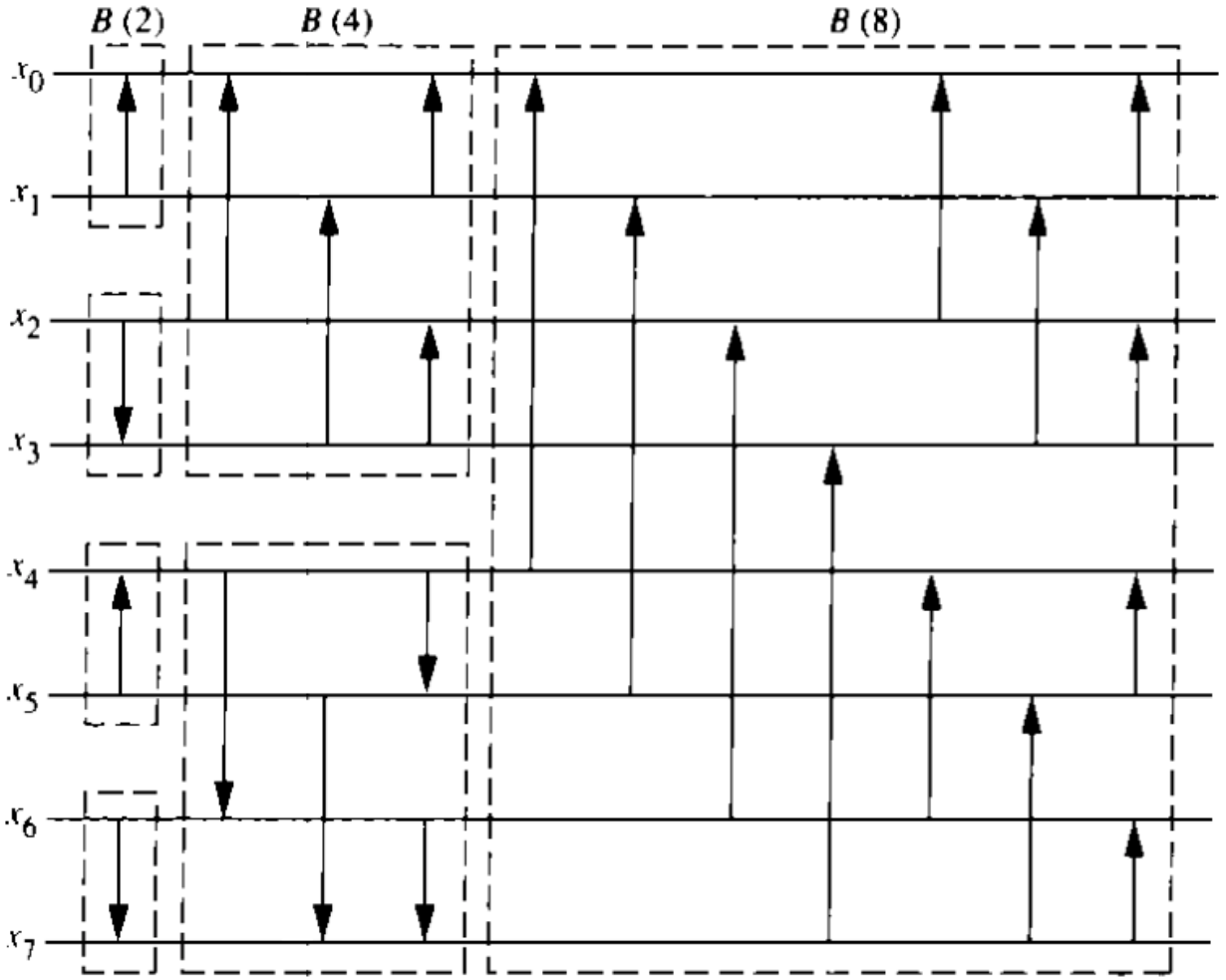
- Čas: $T(n) \in O(\log(n))$
  - každým rekurzívnym volaním vzniknú polovičné problémy
- Práca: $W(n) \in O(n.\log(n))$
  - na každej úrovni máme n/2 komparátorov, resp. porovnaní

- Ako triediť nielen bitonické postupnosti?

Bitonický MergeSort

- Idea: výsledkom rekurzívnych volaní v MergeSorte sú utriedené monotónne postupnosti
  - ich spojenie je bitonická postupnosť, ktorej usporiadaním dostaneme výsledok zlúčenia vstupných postupností
  - bitonické triedenie použijeme na zlučovanie
- **$T(n) \in O(\log^2(n))$**
- **$W(n) \in O(n.\log^2(n))$**
- EREW PRAM

Bitonický MergeSort - sieť komparátorov



Triediaca sieť (Sorting Network)

- vstup triediacej siete je postupnosť $x_1, ..., x_n$
- po konečnom čase sa výstup ustáli na usporiadanú postupnosť $y_1, ..., y_n$

- pre monotónnu funkciu f $(r \leq s \Rightarrow f(r) \leq f(s))$
  - $\min(f(r), f(s)) = f(\min(r, s))$
  - ak postupnosť $x_1, ..., x_n$ je vstup triediacej siete a $y_1, ..., y_n$ je usporiadaný výstup, potom pre vstup $f(x_1), ..., f(x_n)$ je výstup $f(y_1), ..., f(y_n)$ tiež usporiadaný (pre $i < j$ platí $\min(f(y_i), f(y_j)) = f(\min(y_i, y_j)) = f(y_i)$)

Ak triediaca sieť triedi každú postupnosť núl a jednotiek, potom triedi každú postupnosť.

Dôkaz sporom. Nech sieť vie triediť všetky 0-1 postupnosti, ale existuje postupnosť $x_1, x_2, \ldots, x_n$, ktorú utriediť nevie. Potom vo výstupnej postupnosti $y_1, y_2, \ldots, y_n$ existuje y_k, že $y_k > y_{k+1}$

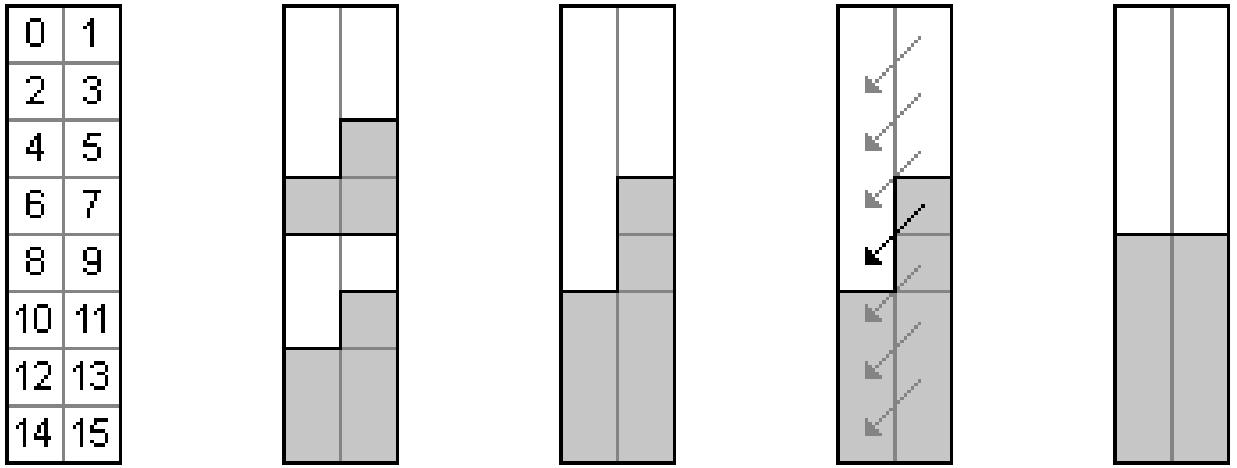
- definujme **monotónnu** funkciu f:
  - $f(x) = 0$, ak $x < y_k$
  - $f(x) = 1$, ak $x >= y_k$
- 0-1 vstup $f(x_1),\ldots, f(x_n)$ usporiada sieť do $f(y_1), \ldots, f(y_n)$
- avšak $f(y_k) = 1$ a $f(y_{k+1}) = 0$, čo je spor s predpokladom, že sieť triedi všetky 0-1 postupnosti

Even-odd zlučovanie (Batcher)

- **Vstup:** postupnosť A taká, že A[0], …, A[n/2-1] je usporiadaná a aj A[n/2], …, A[n] je usporiadaná
- **Výstup:** usporiadaná postupnosť A
- Algoritmus:
  - Rekurzívne vykonaj zlučovanie prvkov na párnych indexoch A[0], A[2], …
  - Rekurzívne vykonaj zlučovanie prvkov na nepárnych indexoch A[1], A[3], …
  - Compare(A[i], A[i+1]) pre i=1, 3, 5, …

Even-odd zlučovanie

- Dôkaz indukciou na $n=2^i$ s využitím 0-1 princípu:



Mapovanie indexov

Vstup – sivé políčka sú 1

Po rekurzívnom aplikovaní zlučovania na párne a nepárne indexy – rozdiel vo „výškach“ je nanajvýš 2

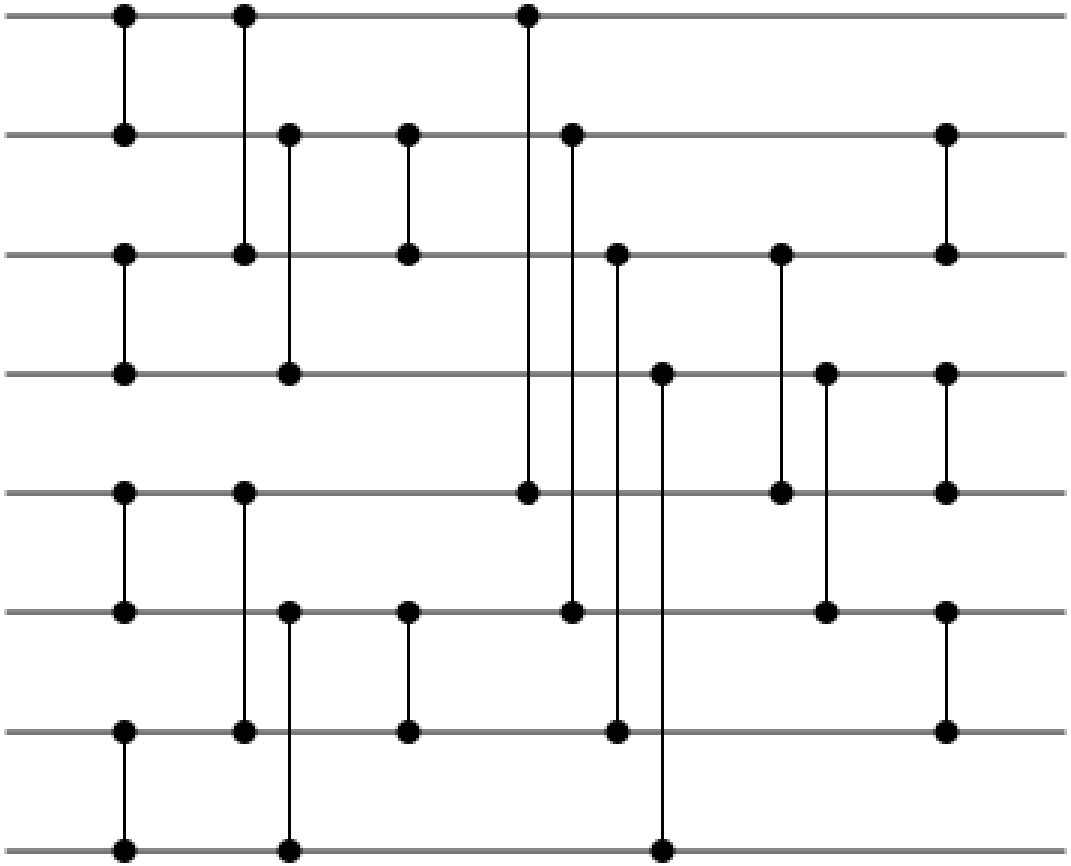
Porovnanie za sebou idúcich dvojíc

Even-odd zlučovanie

- Čas: **$T(n) \in O(\log(n))$**
  - $T(n) = 1 + T(n/2)$
- Práca: **$W(n) \in O(n.\log(n))$**
  - $W(n) = n/2 + 2.W(n/2)$

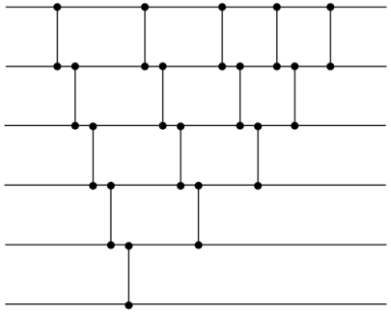
- Dôsledok pre Even-odd MergeSort:
  - Čas: **$O(\log^2(n))$**
  - Práca: **$O(n.\log^2(n))$**

Even-odd MergeSort – sieť komparátorov

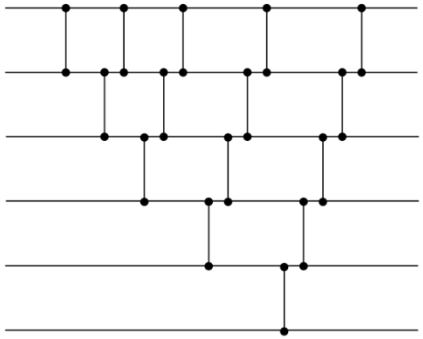


Prebublávanie a insert sort – paralelná sieť

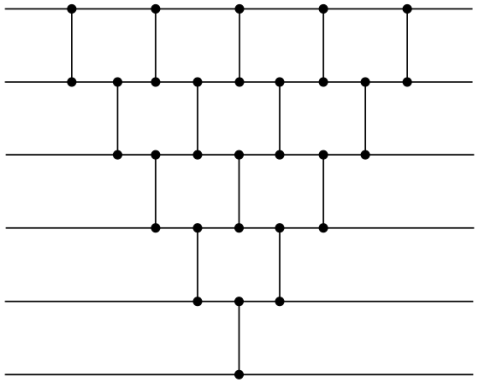
- sieť komparátorov pre sekvenčné prebublávanie



- sieť komparátorov pre sekvenčný insert sort



- sieť komparátorov pre paralelné prebublávanie/insert sort (rovnaká)



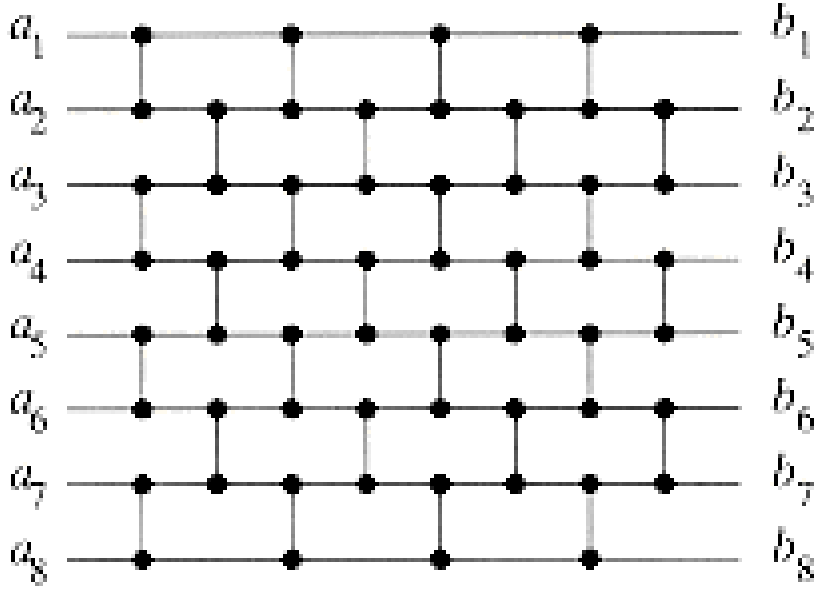
Oblivious porovnávací algoritmus

- Nevšímavý (oblivious) porovnávací algoritmus nezávisí na konkrétnych hodnotách
  - v každom kroku má množinu dvojíc pamäťových miest
  - ak (x, y) je taká dvojica, potom ak x > y, tak obsah pamäťových miest sa vymení

- Zodpovedajú použitiu komparátorov („hardvérovo orientované porovnávanie“)

Oblivious sorting network

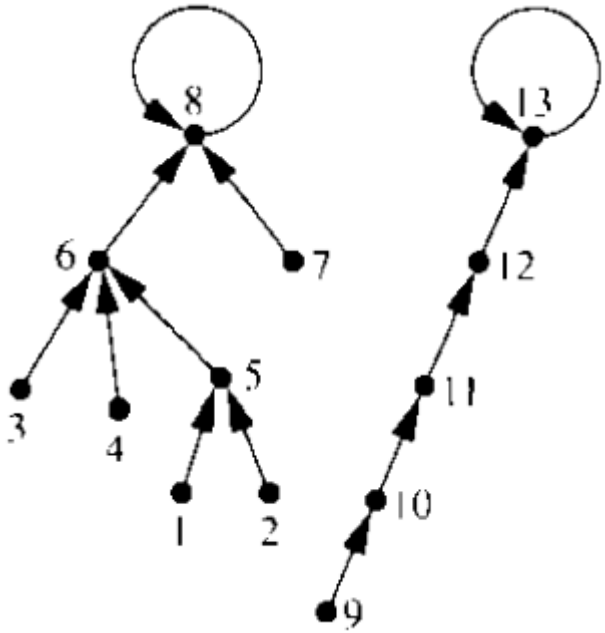
- paralelné porovnávanie všetkých susedných dvojíc
- striedanie porovnávania od nepárnych a od párnych indexov



technika „Pointer jumping“

- **Problém:**
  - Nájdenie koreňov v lese
- **Vstup:**
  - F – les zakorenených stromov
  - P[i] = *j*, ak rodičom *i* je v strome uzol *j*
  - P[i] = *i*, ak *i* je koreňom stromu
- **Výstup:**
  - S[i] – uzol, ktorý je koreňom stromu, v ktorom sa nachádza uzol *i*

nájdenie koreňov v lese



V akom čase sa dá vyriešiť problém **sekvenčne**?

Riešenie:
- obrátenie hrán O(n)
- BFS/DFS -> O(n)

Celkom: O(n)

i	1	2	3	4	5	6	7	8	9	10	11	12	13
P	5	5	6	6	6	8	8	8	10	11	12	13	13
S	8	8	8	8	8	8	8	8	13	13	13	13	13

nájdenie koreňov v lese

```
for i := 1 to n pardo
begin
   S[i] := P[i];
   while (S[i] <> S[S[i]]) do
        S[i] := S[S[i]];
end;
```

- každou iteráciou sa vzdialenosť medzi *i* a *S[i]* zdvojnásobí, ak *S[i]* nie je koreňom stromu
- T(n) = O(log(h)), kde h je najväčšia výška stromu
- W(n) = O(n.log(h))
- CREW

paralelný prefix v lese

- **Vstup:**
  - Les F
  - P[i] = *j*, ak uzol *j* je rodičom uzla *i*
  - P[i] = *i*, ak uzol *i* je koreň
  - W[i] = **ohodnotenie** uzla i
- **Výstup:**
  - S[i] = *j* ak *j* je koreňom stromu, v ktorom je uzol *i*
  - PW[i] = súčet ohodnotení uzlov na ceste z *S[i]* do *i* v strome

paralelný prefix v lese

```
for i := 1 to n pardo
begin
   S[i] := P[i];
   PW[i] := W[i]
   while (S[i] <> S[S[i]]) do
   begin
        PW[i] := PW[i] + PW[S[i]];
        S[i] := S[S[i]];
   end;
end;
```

- $T(n) \in O(\log(h))$, kde h je najväčšia výška stromu v lese F
- $W(n) \in O(n.\log(h))$
- **CREW**
- použitie: prefixové sumy v spájanom zozname

paralelný prefix v zozname

- Zoznam – „degenerovaný“ strom s koreňom v poslednom uzle
- Otočenie zoznamu, výpočet predchodcov:
  - $T(n) \in O(1)$
  - $W(n) \in O(n)$
- Aplikácia algoritmu na stromy:
  - $T(n) \in O(\log(n))$
  - $W(n) \in O(n.\log(n))$

Ďakujem za pozornosť !