4) Algoritmy řazení

- Optimální sekvenční:
  - p(n) = 1
  - t(n) = O(n * log(n))
  - c(n) = O(n * log(n))

Enumeration sort (mřížka)

- Správná pozice každého prvku ve výstupní seřazené posloupnosti je dána počtem prvků, které jsou menší než tento prvek.
- Ideální algoritmus pro paralelní zpracování.

Topologie

- Na mřízce
- p(n) = n^2
- Extrémní případ, dosáhneme nejlepší časové složitosti
- Mřížka, kde procesory jsou vodorovně propojeny do stromu a svisle taky

Vlastnosti procesoru

- Může uložit dva prvky do svých registrů A, B
- Může porovnat A a B a uložit výsledek do registru RANK
- Pomocí stromového propojení může předat obsah kteréhokoliv registru jinému procesoru
- Může přičítat k registru RANK

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

Analýza

- t(n) = O(log(n))
- c(n) = O(n^2 * log(n))
- p(n) = n^2
- Není optimální

Diskuze

- Extrémně rychlý algoritmus O(log(n)), což znamená zrychlení O(n) oproti optimálnímu sekvenčnímu algoritmu.
    - Nic není rychlejší
- Spotřebovává příliš mnoho procesorů, na hranici přijatelnosti

Odd-even transposition sort

- Paralelní bubble sort
- Lineární pole n procesorů p(n) = n

Algoritmus

- 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 každý sudý procesor udělá totéž
- Po n krocích (maximálně) jsou hodnoty seřazeny

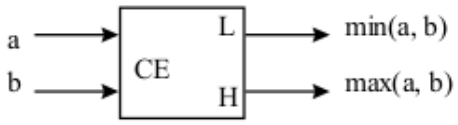
Analýza

- 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 lineární časovou složitost, což je to nejlepší, čeho lze při lineární topologii dosáhnout

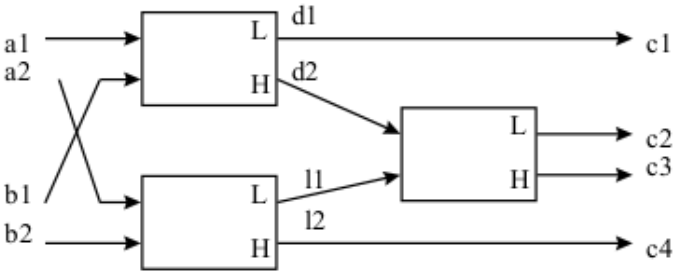
Odd even merge sort

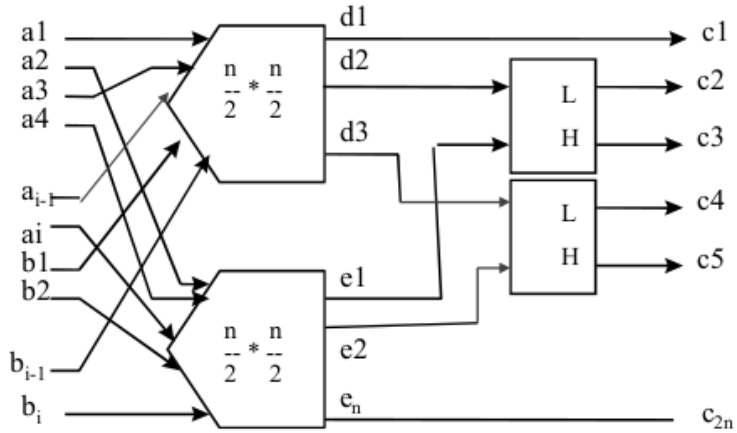
- Řadí se speciální sítí procesorů
- Každý procesor má dva vstupní a dva výstupní kanály
- Každý procesor umí porovnat hodnoty na svých vstupech, menší dá na výstup L (low), a větší dá výstup H (high).

Síť 1x1



2x2





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))

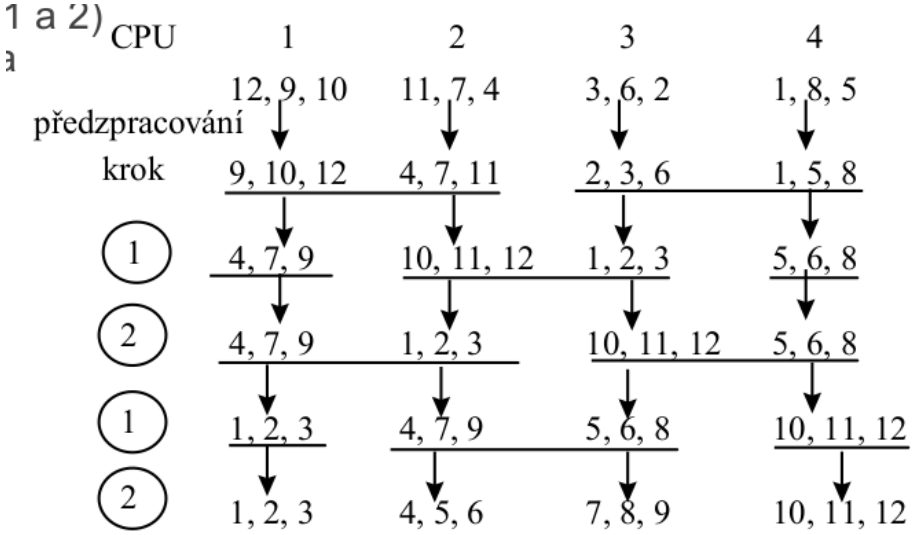
Cena

- 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
- Pomocí nastavení počtu procesorů jsme schopni nastavovat vlastnosti algoritmu

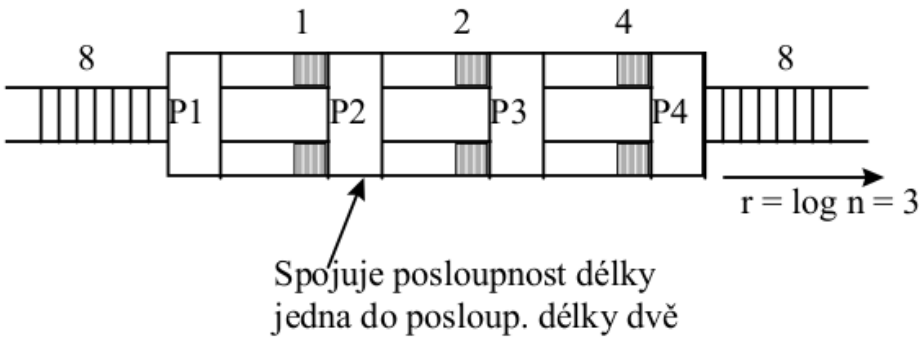


Analýza

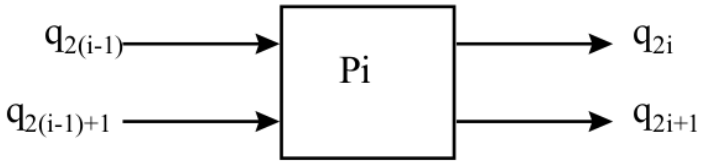
- Předzpracování optimálním alg.
  - O((n/p) * log (n/p))
- Přenos S_i+1 do P_i
  - O(n/p)
- Spojení S_i a S_i+1 do S'_i optimálním alg.
  - 2*n/p
- Přenos S_i+1 do P_i+1
  - O(n/p)
- Krok 1 nebo 2
  - O(n/p)

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

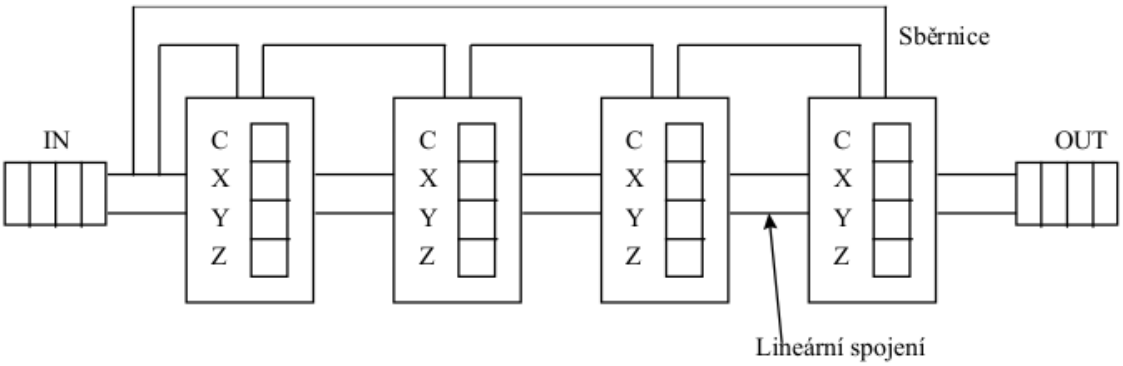
Pipeline Merge sort

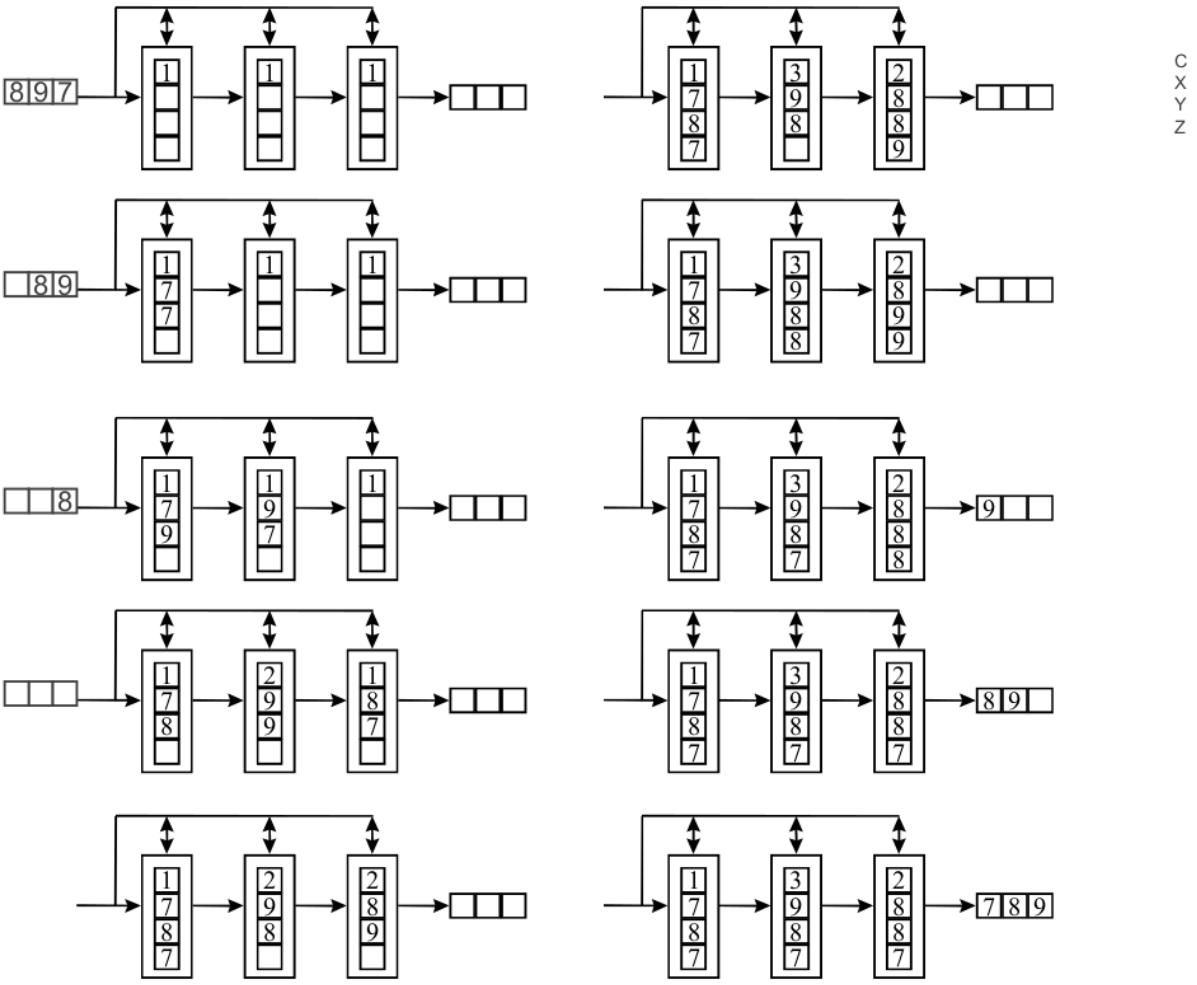


Označení front :



Enumeration sort (lineární pole)



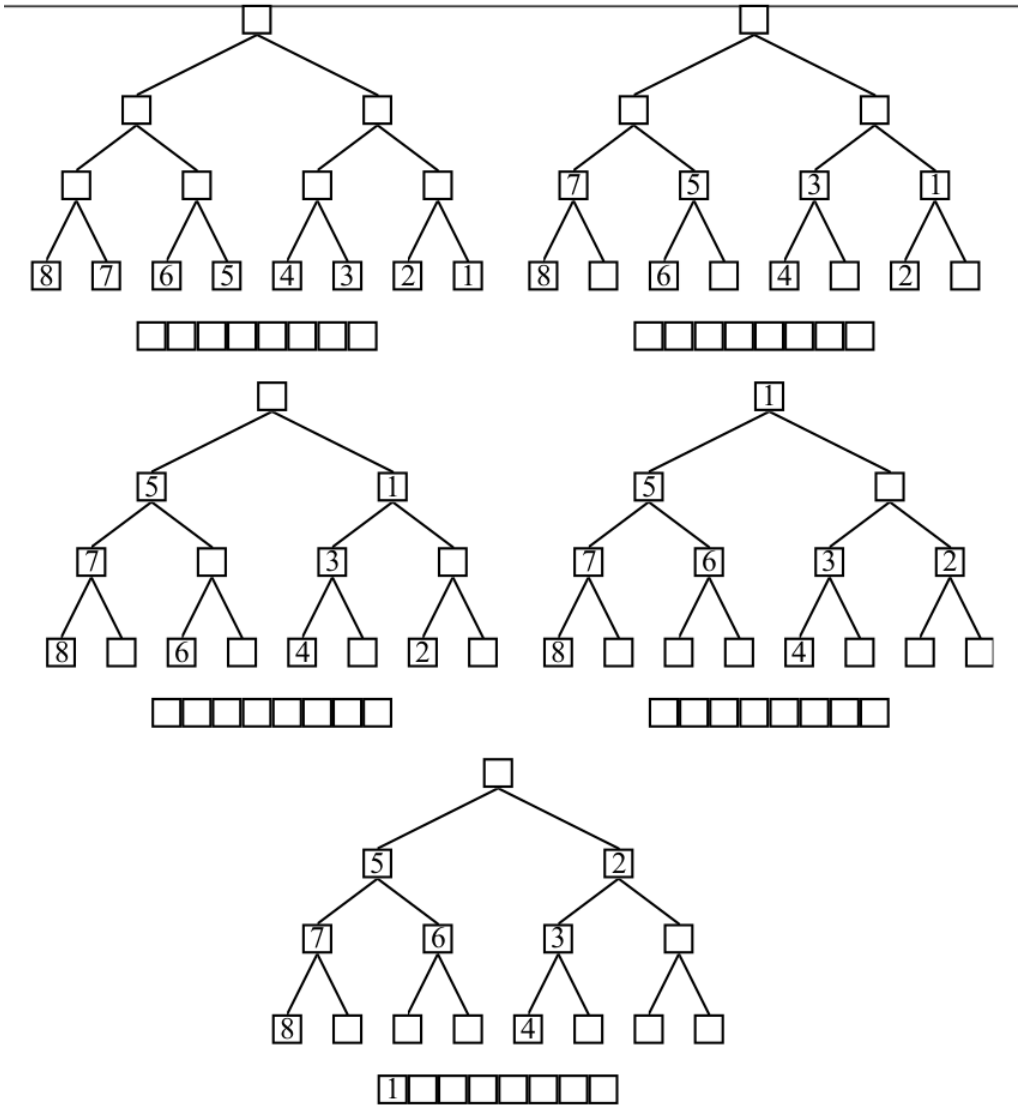


Minimum Extraction Sort

- Stromová architektura
- Každý procesor se stará o jeden prvek

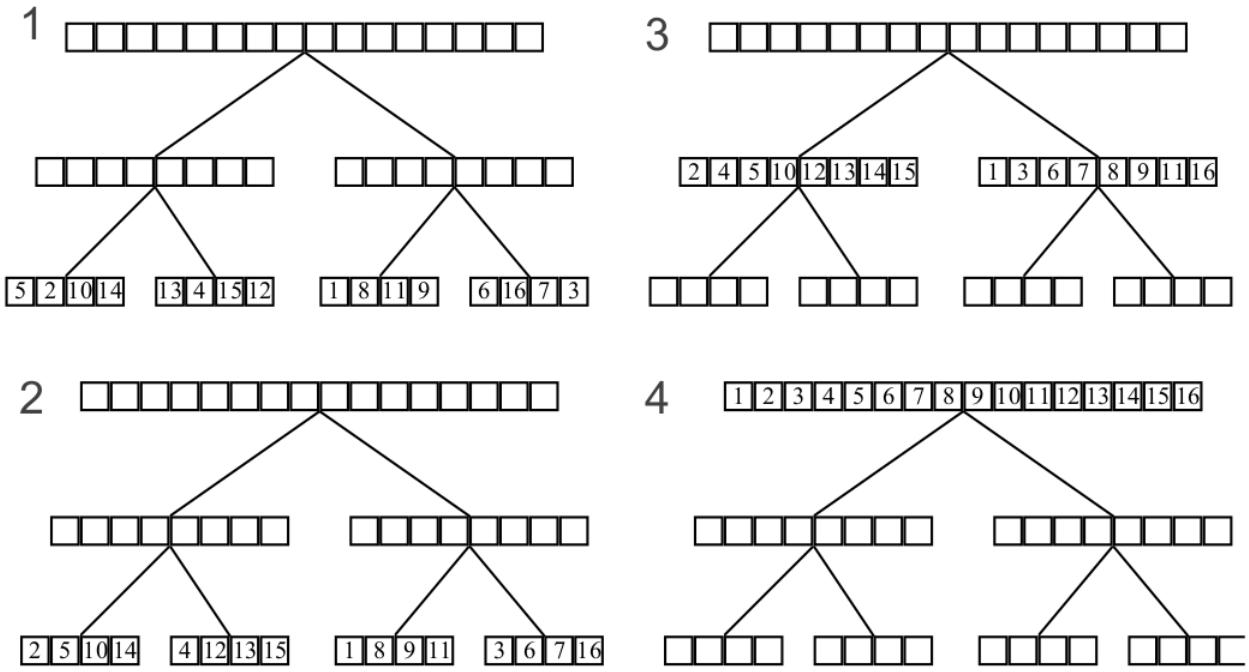
Algoritmus

- 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



Bucket Sort

- Stromová architektura
- Jeden procesor se stará o více než jeden prvek
- Fáze předpřípravy, seřazení optimálním sekvenčním algoritmem
    - viz merge sort

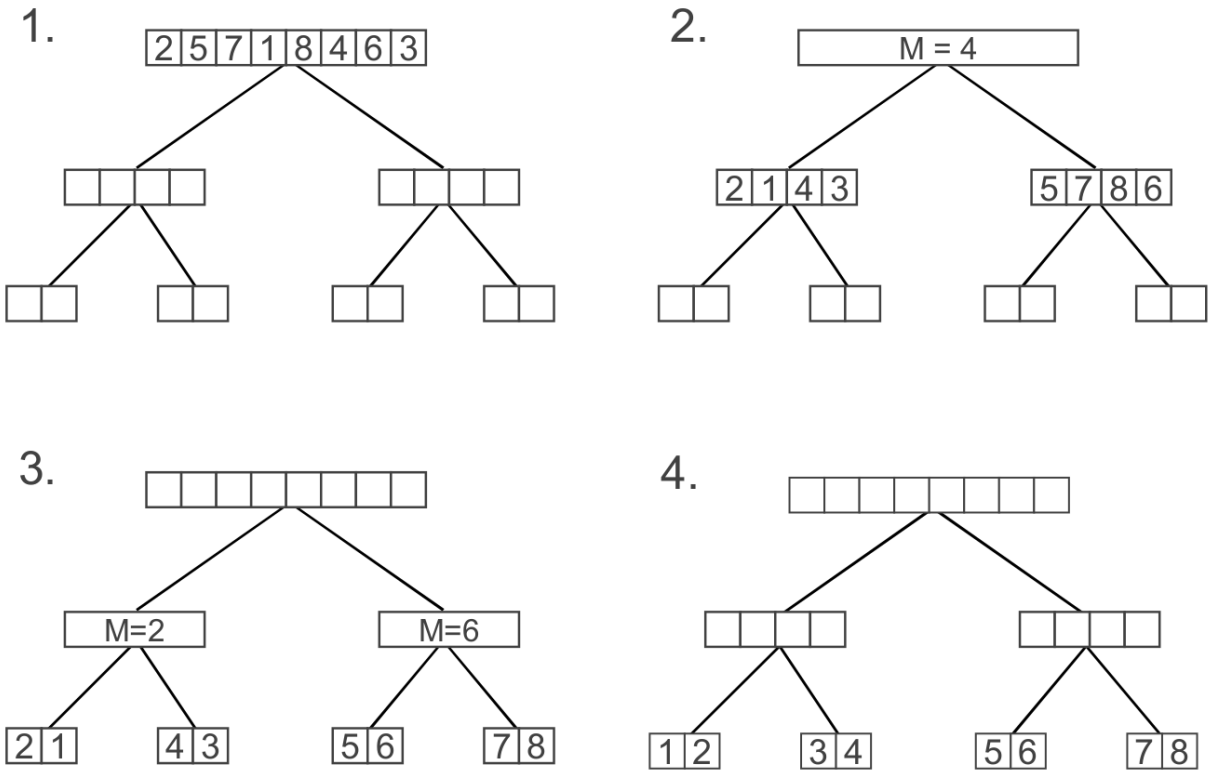


Median Finding and Splitting

- Stromová architektura
- Jeden procesor se stará o více než jeden prvek
- Jdeme opačným směrem, od kořene k listům

Algoritmus

- Uzel najde medián a všechny menší prvky pošle levému synovi a všechny větší pravému.
- Pokračuju do nějakého počtu, poté všichni seřadí zbytek optimálním sekvenčním algoritmem



Algoritmus nalezení mediánu není úplně triviální.

Shrnutí

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 agregovaná verze předchozího		A
Median Finding and Splitting	n	A
Hyperkostka		
Cube Sort $t(n) = O(\log n) .. O(\log^2 n)$ $p(n) = n^2 .. 2n$		

5) Algoritmy vyhledávání

- Zjišťujeme zda prvek je v posloupnosti, případně na jakém indexu.
- Varianty
  - Posloupnost je seřazená
  - Posloupnost není seřazená

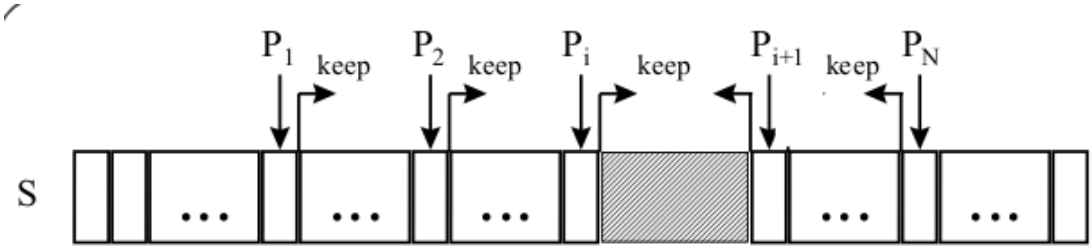
N-ární search

- Vyhledává se seřazené posloupnosti

Princip

- Při binárním vyhledávání zjistíme, ve které polovině se prvek nachází

- Při n-árním vyhledávání zjistíme, ve které z N+1 částí, se prvek nachází, kde N je počet procesorů



- Každý procesor se podívá na svoji “vlaječku” a na “vlaječku” svého pravého souseda
- Ten který detekuje změnu, zapíše svoji pozici a pozici svého pravého souseda – získáme blok je kterém je prvek
- Celý proces opakuju, dokud vyhledávací část není “akorát” malá, pak sekvenčně najdu

Analýza

- 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

Unsorted search

- Vyžaduje architekturu se sdílenou pamětí (PRAM)
- Vyhledává prvek v neseřazené posloupnosti

Princip

- Mám N procesorů, n prvků a hledaný prvek x
- Každý procesor si přečte hledaný prvek
- n prvků je rozděleno mezi N procesorů a provede se sekvenční search
- Do sdílené proměnné pak ten kdo našel zapíše index prvku, případně nedefinovaná hodnota

Analýza

Architektura EREW – Exclusive read exclusive write

- 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)

Architektura CREW – Concurrent read exclusive write

- 1. krok = O(1)
- 2. krok = O(n/N)
- 3.krok = O(log N)
- t(n) = O(log N + n/N)
- c(n) = O(N.log N + n)

Architektura CRCW – Concurrent read concurrent write

- 1. krok = O(1)
- 2. krok = O(n/N)
- 3.krok = O(1)
- t(n) = O(n/N)
- c(n) = O(n) což je optimální

Tree search

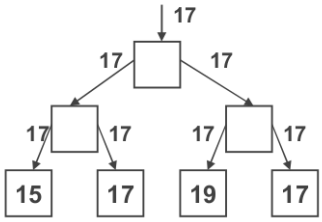
- Vyhledává v neseřazené posloupnosti
- Stromová architektura s 2n-1 procesorů

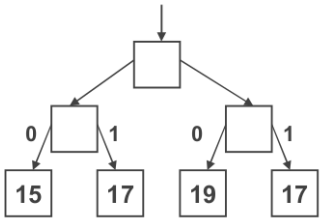
Algoritmus

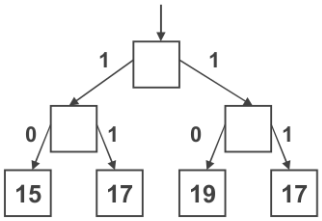
1. Kořen načte hledanou hodnotu x a předá ji synům … až se dostane ke všem listům
2. Listy obsahují seznam prvků, ve kterých se vyhledává (každý list jeden). Všechny listy paralelně porovnávají x a x i , výsledek je 0 nebo 1.
3. Hodnoty všech listů se předají kořenu každý ne list spočte logické or svých synů a výsledek zašle otci. Kořen dostane 0 - nenalezeno, 1- nalezeno

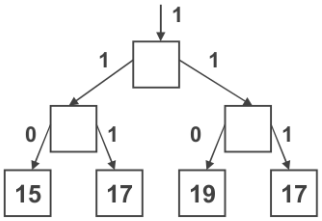
Analýza

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


17
17
17
17
17
17
17
15
17
19
17


0
1
0
1
15
17
19
17


1
1
0
1
0
1
15
17
19
17


1
1
1
0
1
0
1
15
17
19
17