ΤΑΞΙΝΟΜΗΣΗ

Δεδομένης μιας ακολουθίας $S=\{s_1, s_2, \ldots, s_n\}$ από η στοιχεία στα οποία έχει οριστεί η μαθηματική διάταξη $<$, να βρεθεί μια νέα ακολουθία $S'=\{s_1', s_2', \ldots, s_n'\}$ τέτοια ώστε $s_i' < s_{i+1}'$ για $i=1,2,\ldots,n-1$.

Ένα κάτω όριο

Η ταξινόμηση με σύγκριση θεωρητικά μπορεί να μελετηθεί με τα δέντρα αποφάσεως (decision trees). Κάθε μια από τις $n!$ μεταθέσεις των n στοιχείων πρέπει να εμφανίζεται σαν ένα από τα φύλλα του δέντρου αποφάσεως προκειμένου ο αλγόριθμος ταξινόμησης να ταξινομεί κανονικά (σωστά). Συνεπώς ο αριθμός των συγκρίσεων στη χειρότερη περίπτωση για μια ταξινόμηση με σύγκριση καθορίζεται από το ύψος του δέντρου αποφάσεών. Ένα κάτω όριο των υψών των δέντρων αποφάσεως είναι συνεπώς ένα κάτω όριο του χρόνου εκτέλεσης οποιουδήποτε αλγόριθμου ταξινόμησης με σύγκριση.

Θεώρημα

Ένα δέντρο αποφάσεως, το οποίο ταξινομεί η στοιχεία έχει ύψος $\Omega(n \log n)$

Απόδειξη

Επειδή υπάρχουν $n!$ μεταθέσεις των n στοιχείων, το δέντρο πρέπει να έχει τουλάχιστον $n!$ φύλλα. Επειδή ένα δυαδικό δέντρο ύψους h έχει το πολύ 2^h φύλλα, έχουμε

$$n! \leq 2^h \quad \text{ή} \quad h \geq \log(n!)$$

επειδή η $\log$ είναι μονότονα αύξουσα. Αλλά από την προσέγγιση Stirling έχουμε

$$n! \geq \left(\frac{n}{e}\right)^n, \quad e = 2.71828\ldots$$

συνεπώς

$$h \geq \log\left(\frac{n}{e}\right)^n = n\log n - n\log e$$

ή

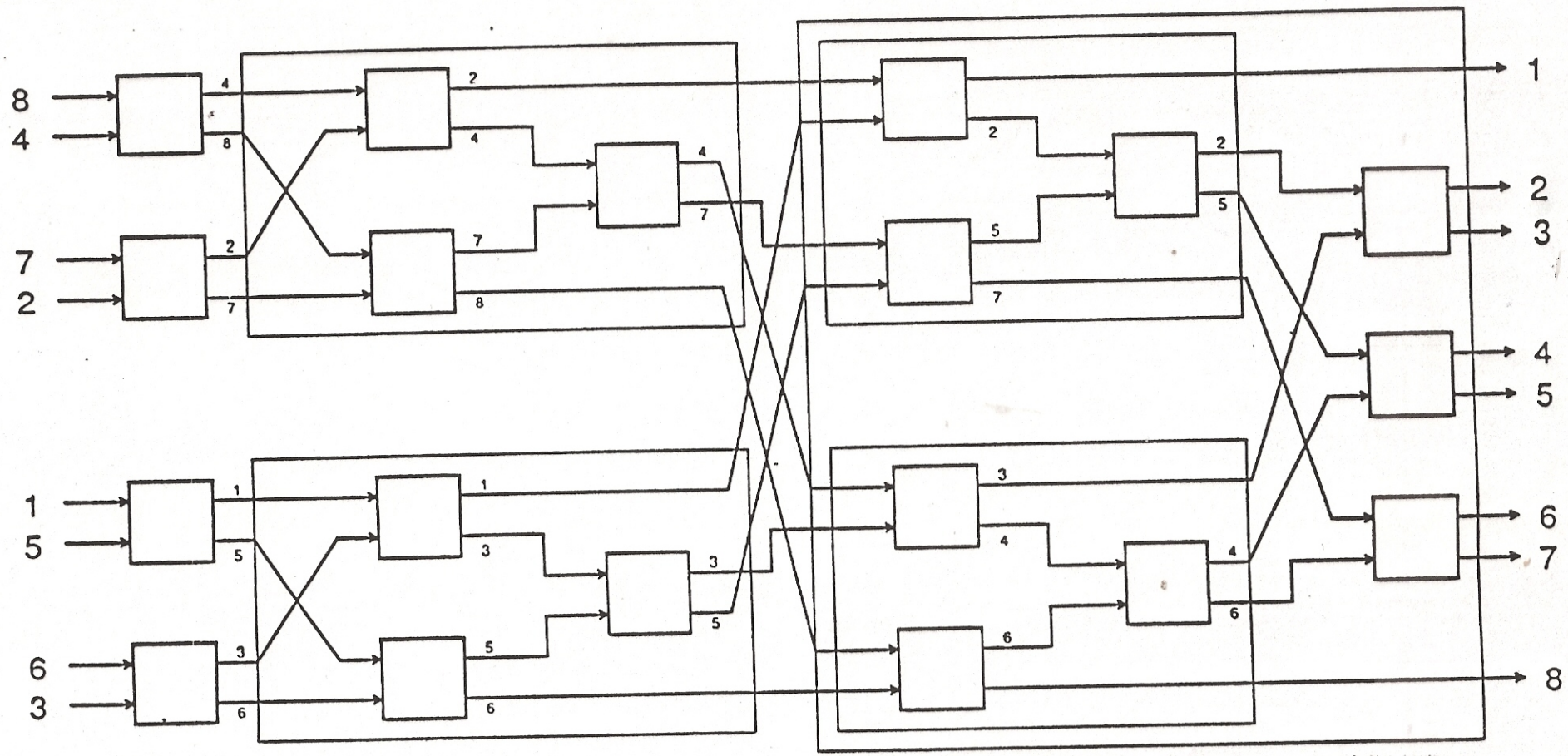
$$h \geq \Omega(n \log n).$$

Άρα ένας αμφίριθμος ταξινόμησης χρειάζεται $\Omega(n \log n)$ πολυπλοκότητα στην χειρότερη περίπτωση.

Συνεπώς για ένα παράλληλο αλγόριθμο ταξινόμησης που χρησιμοποιεί N επεξεργαστές έχουμε $\Omega((n \log n)/N)$, $N \leq n \log n$.

Ένα δίκτυο για ταξινόμηση

Για ταξινόμηση μέσω δικτύου μιας ακολουθίας $S = \{s_1, s_2, \ldots, s_n\}$, όπου n είναι μία δύναμη του 2 ακολουθούμε τα εξής βήματα. Στο πρώτο βήμα χρησιμοποιούνται $n/2$ συγκριτές για τη δημιουργία $n/2$ ταξινομημένων ακολουθιών μήκους 2 η κάθε μία. Στο δεύτερο βήμα ζεύγη αυτών συγχωνεύονται σε ταξινομημένες ακολουθίες μήκους 4 με τη χρήση μιας δομής από (2,2)-δίκτυα συγχώνευσης. Στο τρίτο βήμα, ζεύγη ακολουθιών μήκους 4 συγχωνεύονται με τη χρήση (4,4) δικτύων συγχώνευσης σε ακολουθίες μήκους 8. Η εφαρμογή συνεχίζεται μέχρις ότου δύο ακολουθίες μήκους $n/2$ συγχωνευθούν



(2,2) δίκτυο

(4,4) δίκτυο

Ανάλυση

Επειδή το μέγεθος των ακολουθιών μεγαλώνει διπλασιάζεται σε κάθε βήμα, συνολικά απαιτούνται $\log n$ βήματα.

Χρόνος εκτέλεσης

Αν $s(2^i)$ συμβολίζει το χρόνο εκτέλεσης δύο ακολουθιών στο i-οστό βήμα με 2^{i-1} στοιχεία η κάθε μία, τότε

$$s(2) = 1, \quad i = 1$$

$$s(2^i) = s(2^{i-1}) + 1, \quad i > 1$$

άρα $s(2^i) = i$ και

$$\underline{t(n)} = \sum_{i=1}^{\log n} s(2^i) = \sum_{i=1}^{\log n} i = \underline{O(\log^2 n)}$$

Αριθμός επεξεργασιών

$$q(2) = 1, \quad i = 1$$

$$q(2^i) = 2q(2^{i-1}) + 2^{i-1} - 1, \quad i > 1$$

άρα $q(2^i) = (i-1)2^{i-1} + 1$ και

$$p(n) = \sum_{i=1}^{\log n} 2^{(\log n)-i} \cdot q(2^i) = O(n \log n)$$

Κόστος

$$c(n) = p(n) \cdot t(n) = O(n \log^4 n)$$

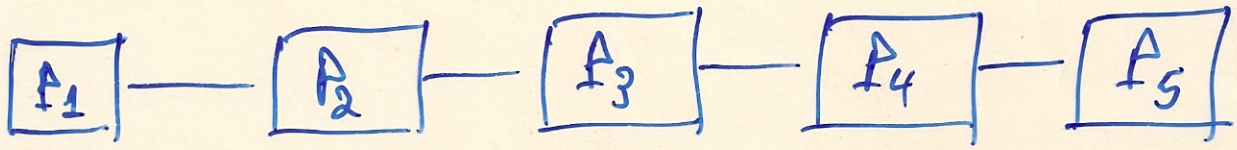
Συνεπώς το δίκτυο ταξινόμησης δεν είναι βέλτιστο.

Παρατηρήσεις

- Το δίκτυο είναι πολύ βαρύφορτο
- Ο αριθμός των συγκριτών είναι πολύ μεγάλος
- Η σύνδεση των συγκριτών παρουσιάζει πρακτικές δυσκολίες.

Ταξινόμηση σε ένα μονοδιάστατο πίνακα επεξεργαστών

Υποθέτουμε ότι έχουμε ένα SIMD υπολογιστή όπου οι επεξεργαστές είναι συνδεδεμένοι ως εξής:



Ο αλγόριθμος που θα αναπτύξουμε για την ταξινόμηση της ακολουθίας $S = \{s_1, s_2, \ldots, s_n\}$ χρησιμοποιεί n επεξεργαστές $P_1, P_2, \ldots, P_n$

Σε κάθε επίπεδο της αναδρομής, η συνάρτηση QUICKSORT βρίσκει το μέσο της ακολουθίας S και έπειτα χωρίζει την S σε δύο υποακολουθίες S1 και S2 στοιχείων <= ή >= του μέσου, αντίστοιχα.

Procedure QUICKSORT (S)

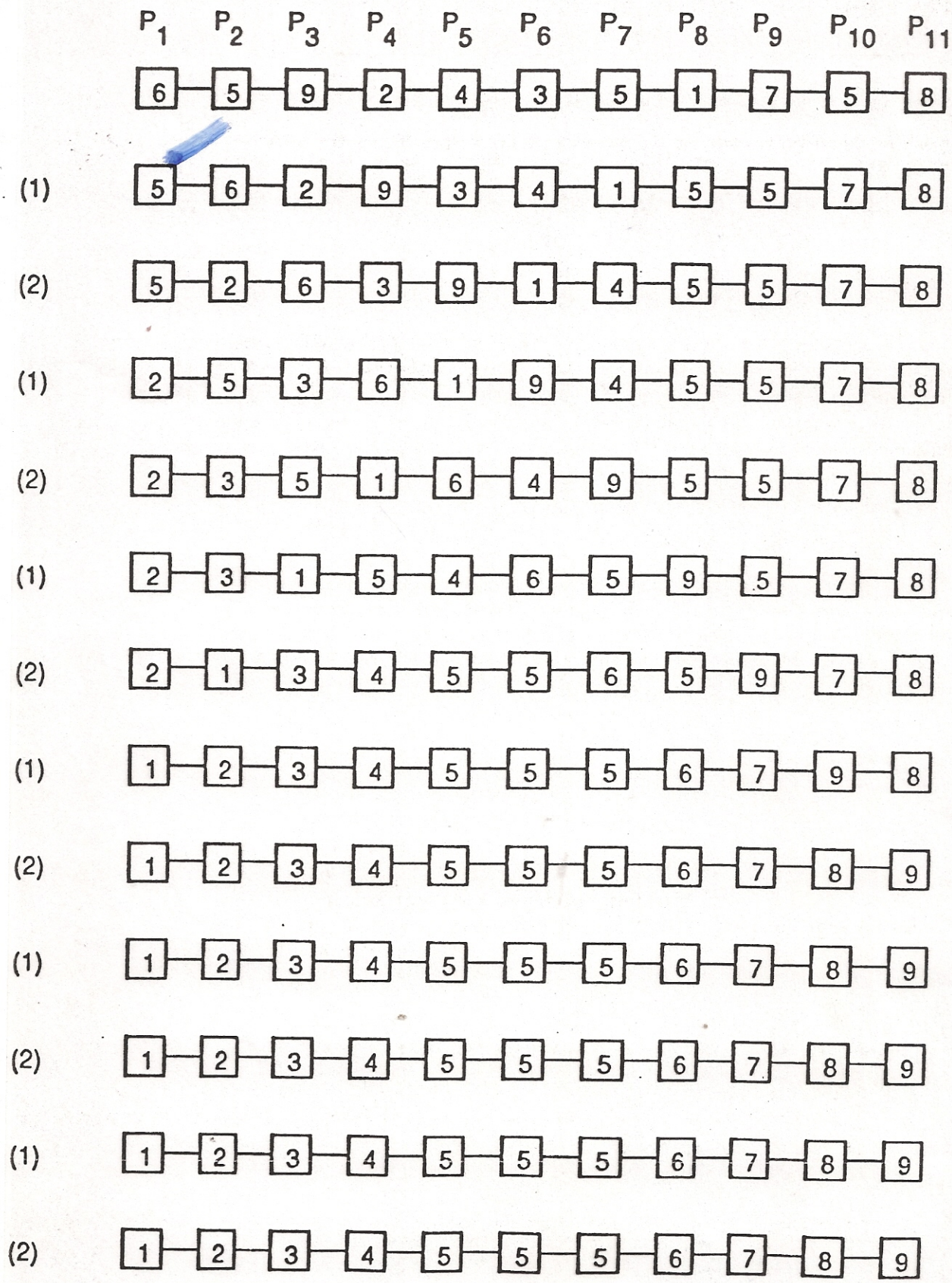
αν $|S|=2$ **και** $S_2 < S_1$
τότε $S_1 \longleftrightarrow S_2$
αλλιώς αν $|S| > 2$ **τότε**
(1) {Προσδιορισμός του m, το μέσο στοιχείο της S}
SEQUENTIAL SELECT $(S, \lceil |S|/2 \rceil)$
(2) {Χωρισμός της S σε δύο υποακολουθίες S_1 και S_2}
(2.1) $S_1 \leftarrow \{s_i : s_i \le m\}$ και $|S_1| = \lceil |S|/2 \rceil$
(2.2) $S_2 \leftarrow \{s_i : s_i \ge m\}$ και $|S_2| = \lceil |S|/2 \rceil$
(3) QUICKSORT(S_1)
(4) QUICKSORT(S_2)
τέλος αν
τέλος αν.

$t(n) = cn + 2t(n/2)$, c = σταθερά
$t(n) = O(n \log n)$ βέλτιστος χρόνος

Procedure ODD-EVEN TRANSPOSITION (S)

για $j=1$ **μέχρι** $\lceil n/2 \rceil$ **κάνε**
(1) **για** $i=1,3,\ldots,2\lfloor n/2 \rfloor -1$ **κάνε παράλληλα**
αν $x_i > x_{i+1}$
τότε $x_i \longleftrightarrow x_{i+1}$
τέλος αν
τέλος για
(2) **για** $i=2,4,\ldots,2\lfloor (n-1)/2 \rfloor -1$ **κάνε παράλληλα**
αν $x_i > x_{i+1}$
τότε $x_i \longleftrightarrow x_{i+1}$
τέλος αν
τέλος για
τέλος για.

ODD-EVEN TRANSPOSITION
μειώνει το χρόνο εκτέλεσης σε σχέση με QUICKSORT



Την 1η φορά χρησιμοποιούνται οι περιττοί επεξεργαστές και τη 2η φορά οι άρτοι.
Συγκρίνουμε αν $a_1 > a_2$ και βάζουμε το a_1 στη θέση του a_2.
Τελειώνει μετά από [n/2] επαναλήψεις των 2 βημάτων και για αυτό το λόγο έχουμε 2 επιπλέον επαναλήψεις παρά το γεγονός ότι έχει γίνει η ταξινόμηση.

Κατά τη διάρκεια εκτέλεσης του αλγορίθμου ο επεξεργαστής P_i έχει ένα στοιχείο x_i, $i = 1, 1, \ldots, n$. Αρχικά $x_i = s_i$. Ο αλγόριθμος επαναλαμβάνει όμως δύο βήματα. Στο πρώτο βήμα όλοι οι περιττοί επεξεργαστές P_i λαμβάνουν το x_{i+1} από τον P_{i+1}. Αν $x_i > x_{i+1}$ τότε οι P_i και P_{i+1} ανταλλάσσουν τα στοιχεία τους. Στο δεύτερο βήμα όλοι οι άρτιοι επεξεργαστές εκτελούν την ίδια εργασία όπως οι περιττοί στο πρώτο βήμα. Μετά από $\lceil n/2 \rceil$ επαναλήψεις αυτών των δύο βημάτων θα έχουν τελειώσει όλες οι ανταλλαγές των στοιχείων.

Ανάλυση

$$t(n) = O(n), \quad p(n) = n \qquad c(n) = O(n^2)$$

c(n)=p(n)xt(n)

άρα δεν είναι βέλτιστος.

Παρατηρήσεις

- Πετυχαίνει σε σχέση με τον Quicksort μία ταχύτητα $O(\log n)$
- Χρησιμοποιεί μεγάλο αριθμό επεξεργαστών
- Δεν είναι βέλτιστου κόστους.

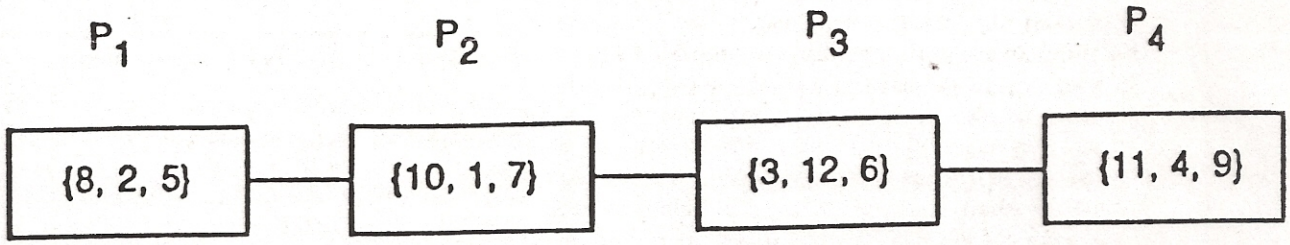
Νέος Αλγόριθμος

Υποθέτουμε ότι κάθε ένας P_i από τους $N \leq M$ επεξεργαστές έχει μία υποακολουθία S_i της S μήκους n/N (προσθέτουμε εικονική στοιχεία, αν το n δεν είναι πολλαπλάσιο του N). Στον νέο αλγόριθμο οι συγκρίσεις-ανταλλαγές γίνονται με συγχώνευση-χωρισμό ανάμεσα σε υποακολουθίες. Στο βήμα 1 κάθε επεξεργαστής P_i ταξινομεί την S_i με τον QUICKSORT. Στο βήμα 2.1 κάθε περιττός επεξεργαστής συγχωνεύει τις δύο υποακολουθίες S_i και S_{i+1} σε μία ταξινομημένη ακολουθία $S_i' = \{S_1', S_2', \ldots, S_{2n/N}'\}$. Διατηρεί το πρώτο μέρος της S_i' και καταχωρεί στον γειτονικό του επεξεργαστή P_{i+1} το δεύτερο τμήμα. Το βήμα 2.2 είναι το ίδιο με το 2.1 εκτός ότι η άνω τομή χρησιμοποιούν οι άρτιοι επεξεργαστές. Μετά από $\lceil N/2 \rceil$ επαναλήψεις έχουν τελειώσει σε όλες οι ανταλλαγές των στοιχείων μεταξύ δύο επεξεργαστών. Στο τέλος η ακολουθία $S = S_1, S_2, \ldots, S_N$ είναι ταξινομημένη.

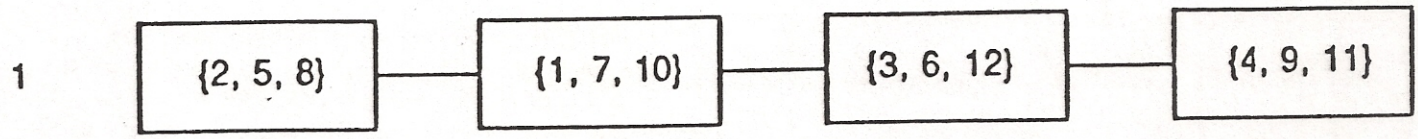
MERGE SPLIT μειώνει τον αριθμό των επεξεργαστών

$S = \{8, 2, 5, 10, 1, 7, 3, 12, 6, 11, 4, 9\}$

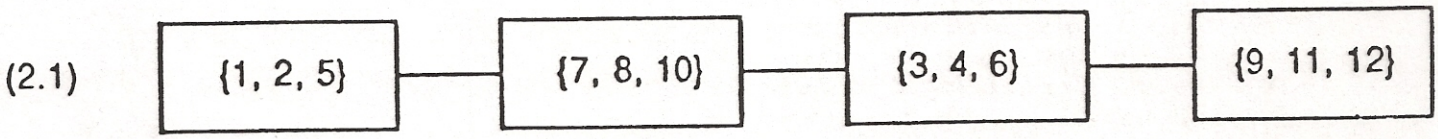
$N = 4, \quad n = 12$



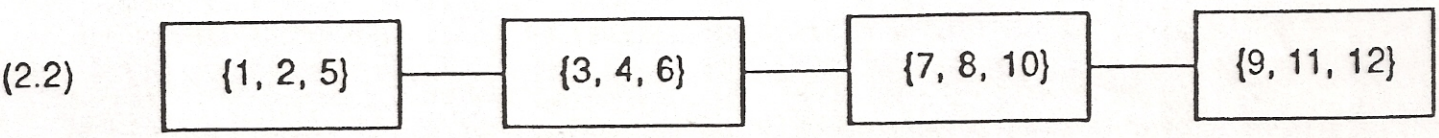
Ταξινόμηση με την QUICKSORT

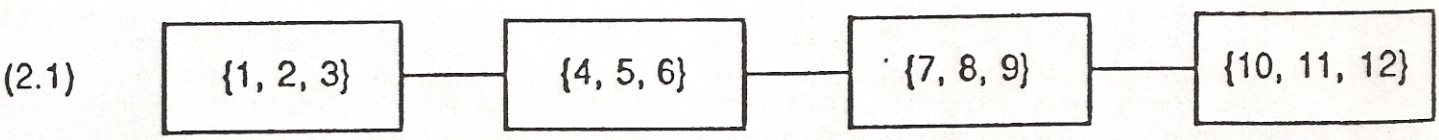


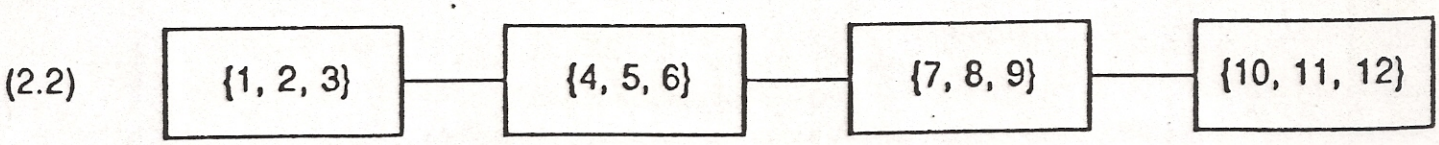
Ξεκινάμε με περιττούς επεξεργαστές και συγχωνεύουμε τις ακιολουθίες. Ο P1 κρατάει τη μικρότερη και στέλνει στον P2 τη μεγαλύτερη.



Όμοια, μόνο που τώρα ξεκινάμε με τους άρτιους επεξεργαστές.







Procedure MERGE SPLIT (S)

Βήμα 1: για i=1 μέχρι N κάνε παράλληλα
QUICKSORT (S_i)
τέλος για.

Βήμα 2: για j=1 μέχρι $\lceil N/2 \rceil$ κάνε
(2.1) για i=1,3,...,$2\lfloor N/2 \rfloor$-1 κάνε παράλληλα
(i) SEQUENTIAL MERGE (S_i, S_{i+1}, S_i')
(ii) $S_i \leftarrow \{ s'_1, s'_2, \ldots, s'_{n/N} \}$
(iii) $S_{i+1} \leftarrow \{ s'_{(n/N)+1}, s'_{(n/N)+2}, \ldots, s'_{2n/N} \}$
τέλος για
(2.2) για i=2,4,...,$2\lfloor (N-1)/2 \rfloor$ κάνε παράλληλα
(i) SEQUENTIAL MERGE (S_i, S_{i+1}, S_i')
(ii) $S_i \leftarrow \{ s'_1, s'_2, \ldots, s'_{n/N} \}$
(iii) $S_{i+1} \leftarrow \{ s'_{(n/N)+1}, s'_{(n/N)+2}, \ldots, s'_{2n/N} \}$
τέλος για
τέλος για.

Ανάλυση: Το Βήμα 1 διαρκεί $O((n/N)\log(n/N))$ βήματα. Μεταφορά της S_{i+1} στον P_i, συγχώνευση με SEQUENTIAL MERGE, και επιστροφή της S_{i+1} στον P_{i+1} διαρκούν $O(n/N)$ χρόνο. Συνεπώς

$t_{comm}=t_s+(n/N)t_w$
$t_{comm}=O(n/N)$

$$t(n) = O((n/N)\log(n/N)) + \lceil N/2 \rceil \cdot O(n/N)$$
$$= O((n\log n)/N) + O(n)$$
$$c(n) = O(n\log n) + O(nN)$$

το οποίο είναι βέλτιστο όταν $N \leq \log n$.

```
1.  procedure BITONIC_SORT(label, d)
2.  begin
3.      for i := 0 to d − 1 do
4.          for j := i downto 0 do
5.              if (i + 1)st bit of label ≠ jth bit of label then
6.                  comp_exchange_max(j);
7.              else
8.                  comp_exchange_min(j);
9.  end BITONIC_SORT
```

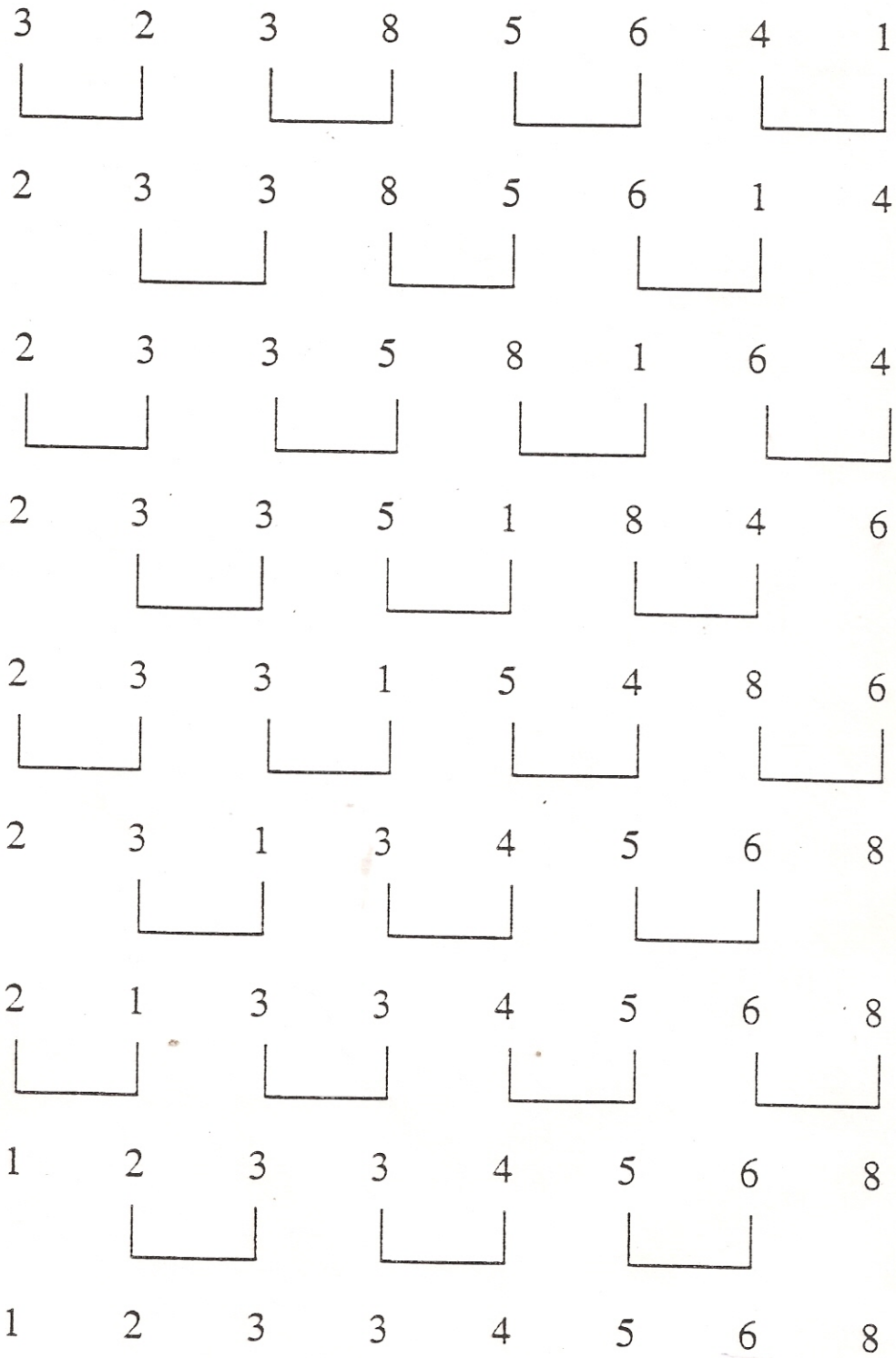
```
1.    procedure BUBBLE_SORT(n)
2.    begin
3.        for i := n − 1 downto 1 do
4.            for j := 1 to i do
5.                compare-exchange(a_j , a_{j+1});
6.    end BUBBLE_SORT
```

```
1.   procedure ODD-EVEN(n)
2.   begin
3.       for i := 1 to n do
4.       begin
5.           if i is odd then
6.               for j := 0 to n/2 − 1 do
7.                   compare-exchange(a_{2j+1}, a_{2j+2});
8.           if i is even then
9.               for j := 1 to n/2 − 1 do
10.                  compare-exchange(a_{2j}, a_{2j+1});
11.      end for
12.  end ODD-EVEN
```

$$T = \Theta\left(\frac{n}{p}\log\frac{n}{p}\right) + \Theta(n) + \Theta(n).$$

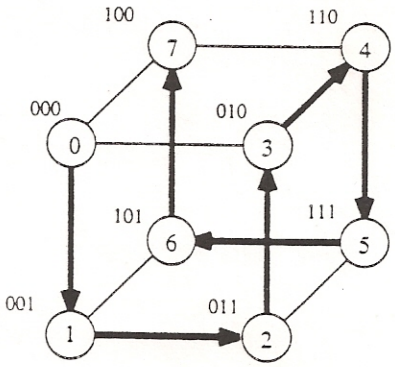
$$S = \frac{\Theta(n\log n)}{\Theta((n/p)\log(n/p)) + \Theta(n)}$$

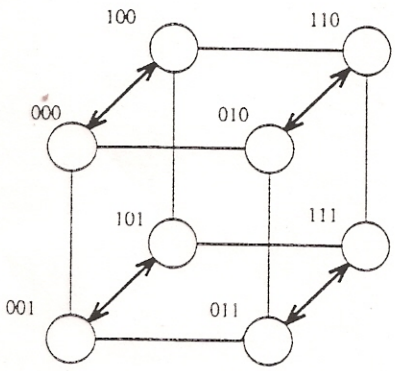
$$E = \frac{1}{1 - \Theta((\log p)/(\log n)) + \Theta(p/\log n)}$$

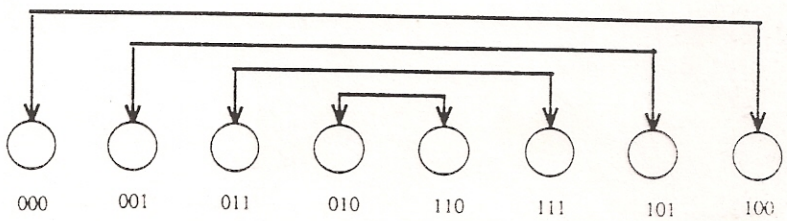

3 2 3 8 5 6 4 1
2 3 3 8 5 6 1 4
2 3 3 5 8 1 6 4
2 3 3 5 1 8 4 6
2 3 3 1 5 4 8 6
2 3 1 3 4 5 6 8
2 1 3 3 4 5 6 8
1 2 3 3 4 5 6 8
1 2 3 3 4 5 6 8

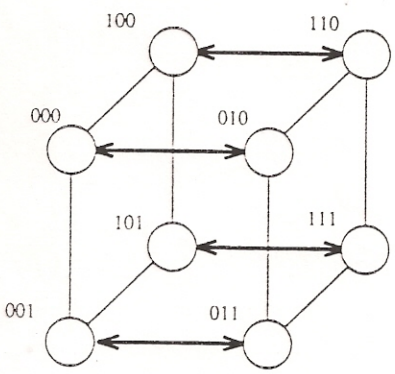
```
1.   procedure ODD-EVEN_PAR(n)
2.   begin
3.       id := processor's label
4.       for i := 1 to n do
5.       begin
6.           if i is odd then
7.               if id is odd then
8.                   compare-exchange_min(id + 1);
9.               else
10.                  compare-exchange_max(id − 1);
11.          if i is even then
12.              if id is even then
13.                  compare-exchange_min(id + 1);
14.              else
15.                  compare-exchange_max(id − 1);
16.      end for
17.  end ODD-EVEN_PAR
```

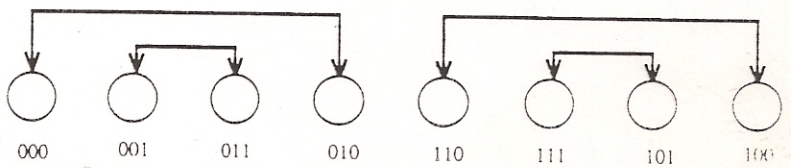
$$T_p = \Theta\left(\frac{n}{p}\log\frac{n}{p}\right) + \Theta\left(\frac{n}{p}\log p\right) + \Theta\left(l\frac{n}{p}\right).$$

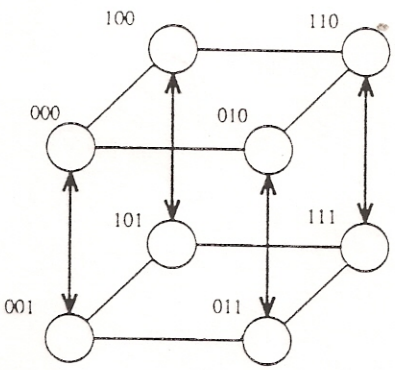

100
110
7
4
000
010
0
3
101
111
6
5
001
011
1
2

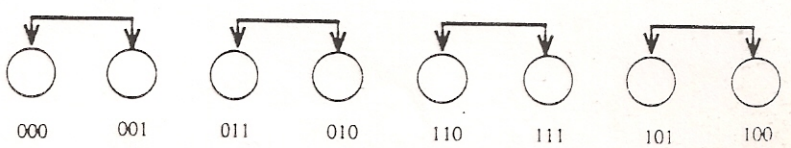

100
110
000
010
101
111
001
011


000
001
011
010
110
111
101
100


100
110
000
010
101
111
001
011


000
001
011
010
110
111
101


100
110
000
010
101
111
001
011


000
001
011
010
110
111
101
100

```
1.   procedure QUICKSORT (A, q, r)
2.   begin
3.       if q < r then
4.       begin
5.           x := A[q];
6.           s := q;
7.           for i := q + 1 to r do
8.               if A[i] ≤ x then
9.               begin
10.                  s := s + 1;
11.                  swap(A[s], A[i]);
12.              end if
13.          swap(A[q], A[s]);
14.          QUICKSORT (A, q, s);
15.          QUICKSORT (A, s + 1, r);
16.      end if
17.  end QUICKSORT
```

(a) | 3 | 2 | 1 | 5 | 8 | 4 | 3 | 7 |

(b) | 1 | 2 | 3 | 5 | 8 | 4 | 3 | 7 |

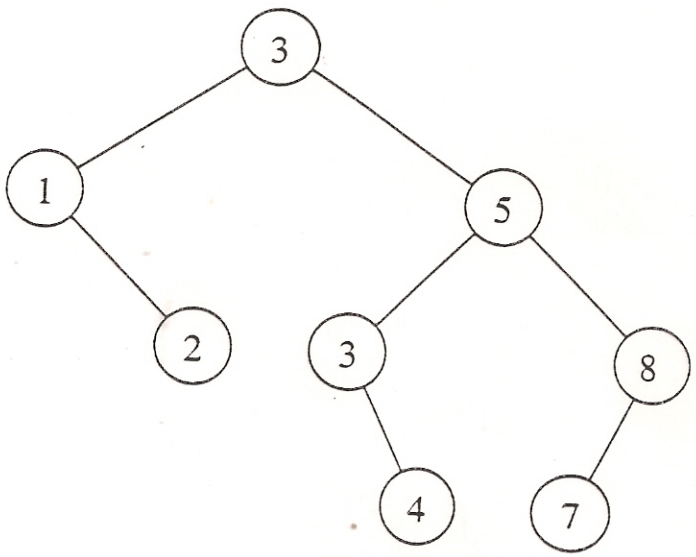
(c) | 1 | 2 | 3 | 3 | 4 | 5 | 8 | 7 |

(d) | 1 | 2 | 3 | 3 | 4 | 5 | 7 | 8 |

(e) | 1 | 2 | 3 | 3 | 4 | 5 | 7 | 8 |

Pivot

Final position



```
1.  procedure BUILD_TREE (A[1 . . . n])
2.  begin
3.      for each processor i do
4.      begin
5.          root := i;
6.          parent_i := root;
7.          leftchild[i] := rightchild[i] := n + 1;
8.      end for
9.      repeat for each processor i ≠ root do
10.     begin
11.         if (A[i] < A[parent_i]) or
                (A[i] = A[parent_i] and i <parent_i) then
12.         begin
13.             leftchild[parent_i] := i;
14.             if i = leftchild[parent_i] then exit
15.             else parent_i := leftchild[parent_i];
16.         end for
17.         else
18.         begin
19.             rightchild[parent_i] := i;
20.             if i = rightchild[parent_i] then exit
21.             else parent_i := rightchild[parent_i];
22.         end else
23.     end repeat
24. end BUILD_TREE
```

(a)

1	2	3	4	5	6	7	8
33	21	13	54	82	33	40	72

(b) = 4

(c)

	1	2	3	4	5	6	7	8
				1				
				5				

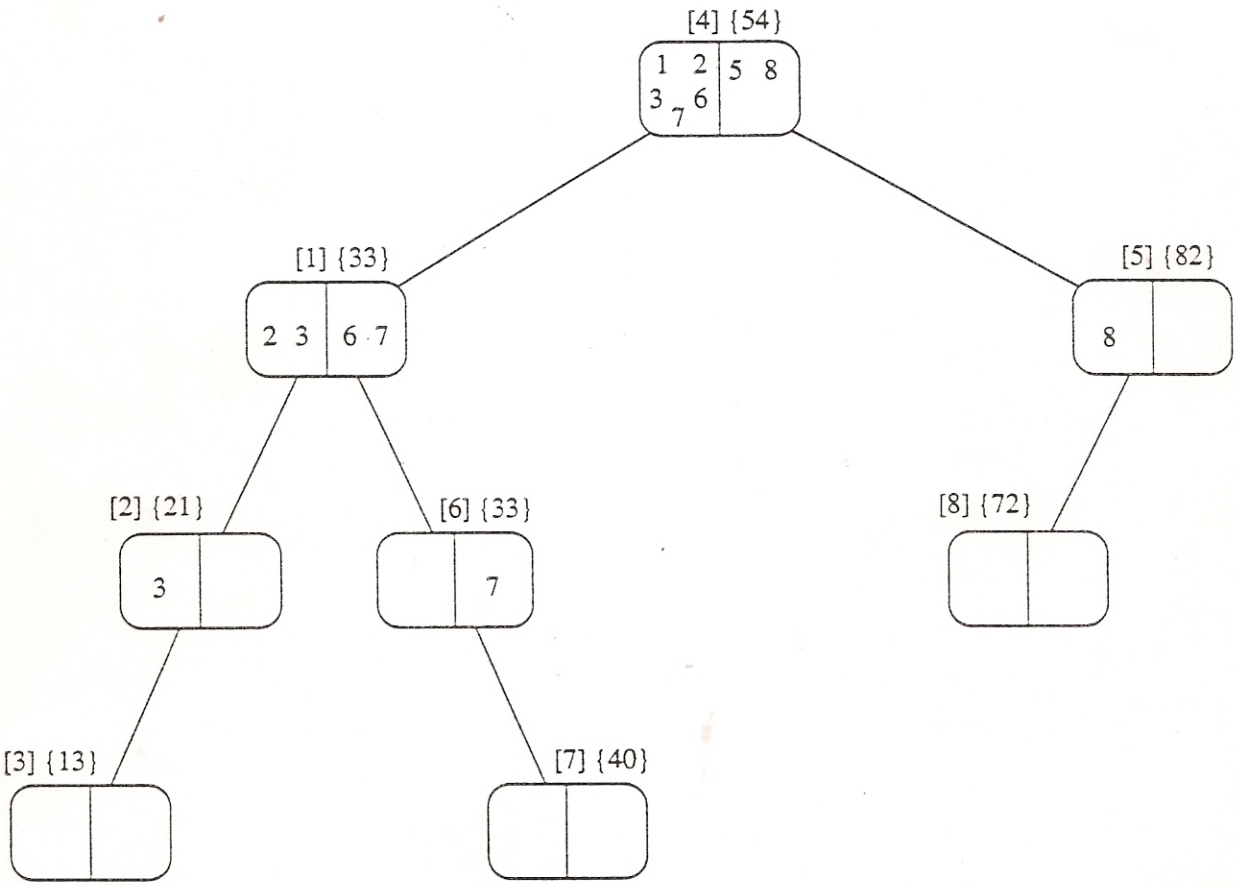
(d)

	1	2	3	4	5	6	7	8
	2			1	8			
	6			5				

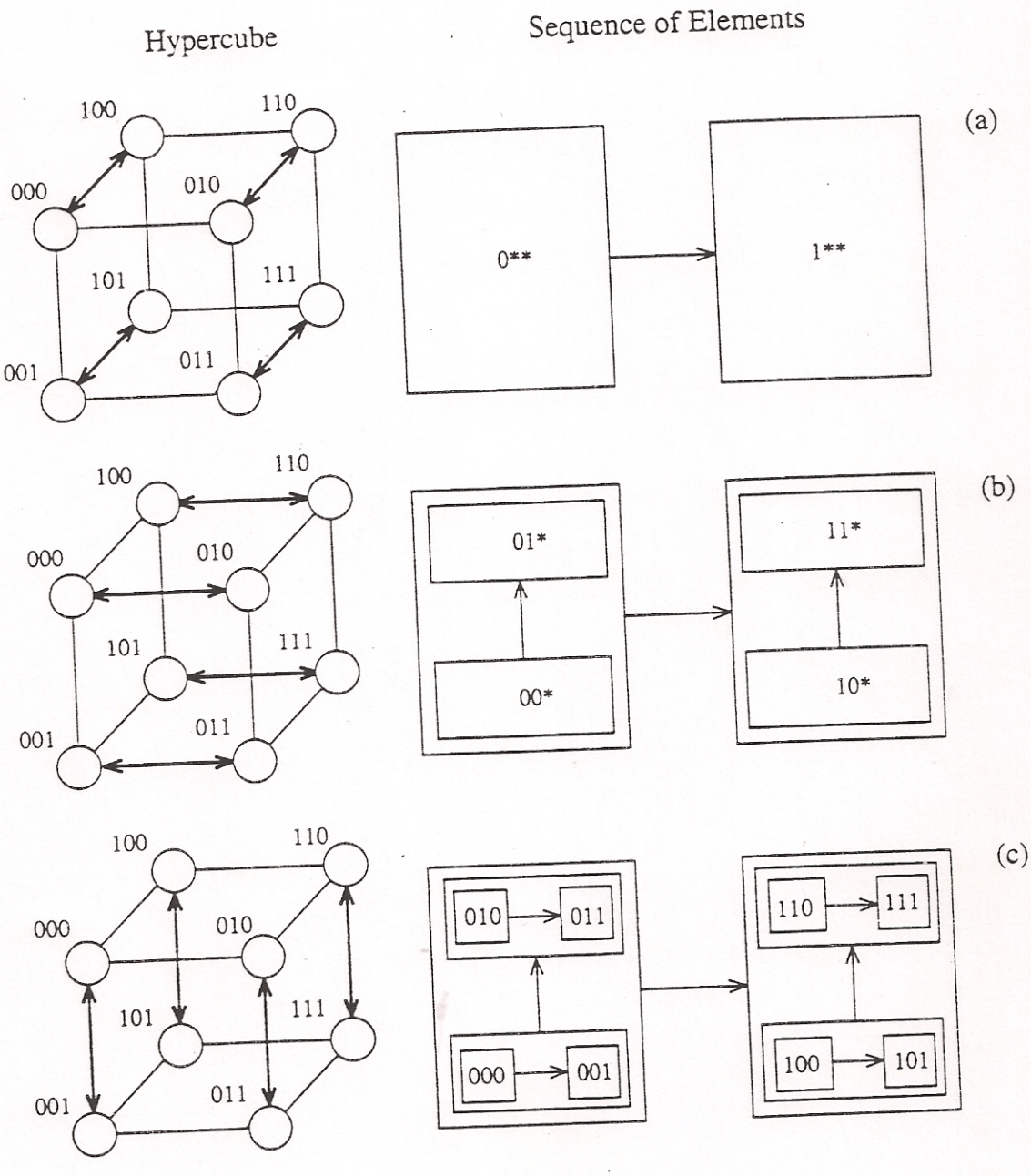
(e)

	1	2	3	4	5	6	7	8
	2	3		1	8			
	6			5		7		

(f)



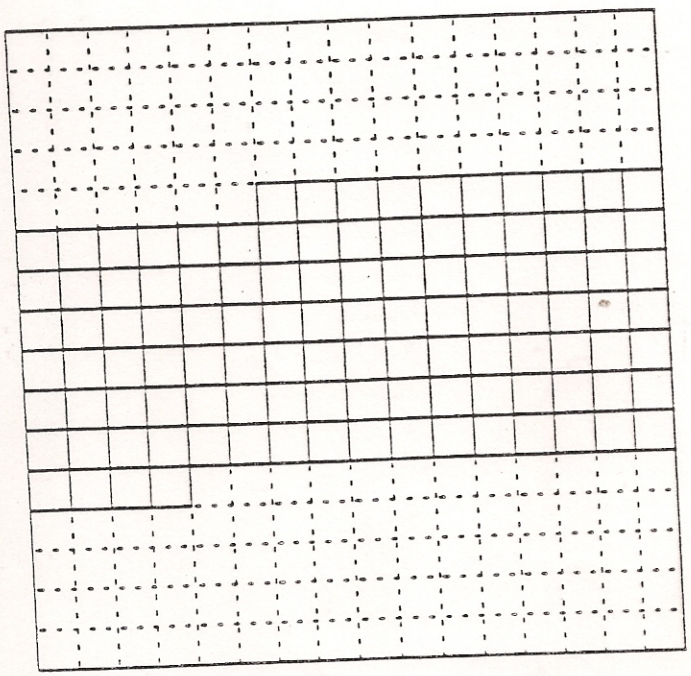
```
1.  procedure HYPERCUBE_QUICKSORT (B, n)
2.  begin
3.      id := processor's label;
4.      for i := 1 to d do
5.      begin
6.          x := pivot;
7.          partition B into B1 and B2 such that B1 ≤ x < B2;
8.          if i^th bit is 0 then
9.          begin
10.             send B2 to the processor along the i^th communication link;
11.             C := subsequence received along the i^th communication link;
12.             B := B1 ∪ C;
13.         endif
14.         else
15.             send B1 to the processor along the i^th communication link;
16.             C := subsequence received along the i^th communication link;
17.             B := B2 ∪ C;
18.         endelse
19.     endfor
20.     sort B using sequential quicksort;
21. end HYPERCUBE_QUICKSORT
```


Hypercube
Sequence of Elements
100
110
000
010
101
111
001
011
0**
1**
(a)
01*
00*
11*
10*
(b)
010
011
000
001
110
111
100
101
(c)

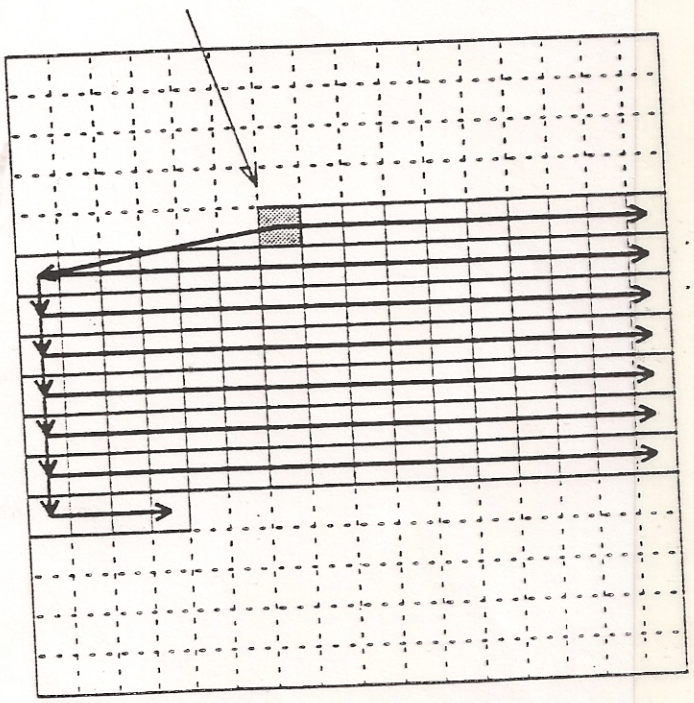
$$T_p = \Theta\left(\frac{n}{p}\log\frac{n}{p}\right) + \Theta\left(\frac{n}{p}\log p\right) + \Theta(\log^2 p).$$

$$S = \frac{\Theta(n\log n)}{\Theta((n/p)\log(n/p)) + \Theta((n/p)\log p) + \Theta(\log^2 p)}$$

$$E = \frac{1}{1 + \Theta((\log p)/(\log n)) + \Theta((p\log^2 p)/(n\log n))}$$



(a)



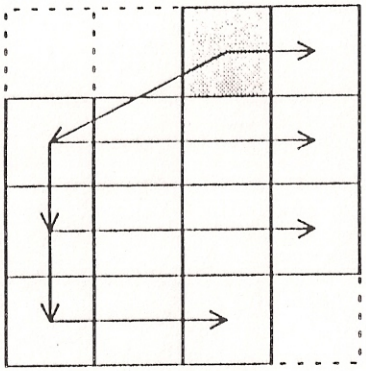
(b)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(a)

		50	11
60	15	91	49
24	77	83	17
41	94	55	

(b)



(c)

6/6
5/6
3/4
1/2

		50	11
60	15	91	49
24	77	83	17
41	94	55	

(d)

2/9
3/9
5/11
7/13

		50	11
60	15	91	49
24	77	83	17
41	94	55	

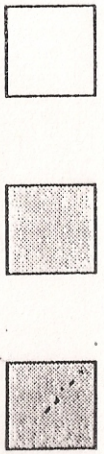
(e)

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

(f)

		11	15
49	24	17	41
50	60	91	77
83	94	55	

(g)



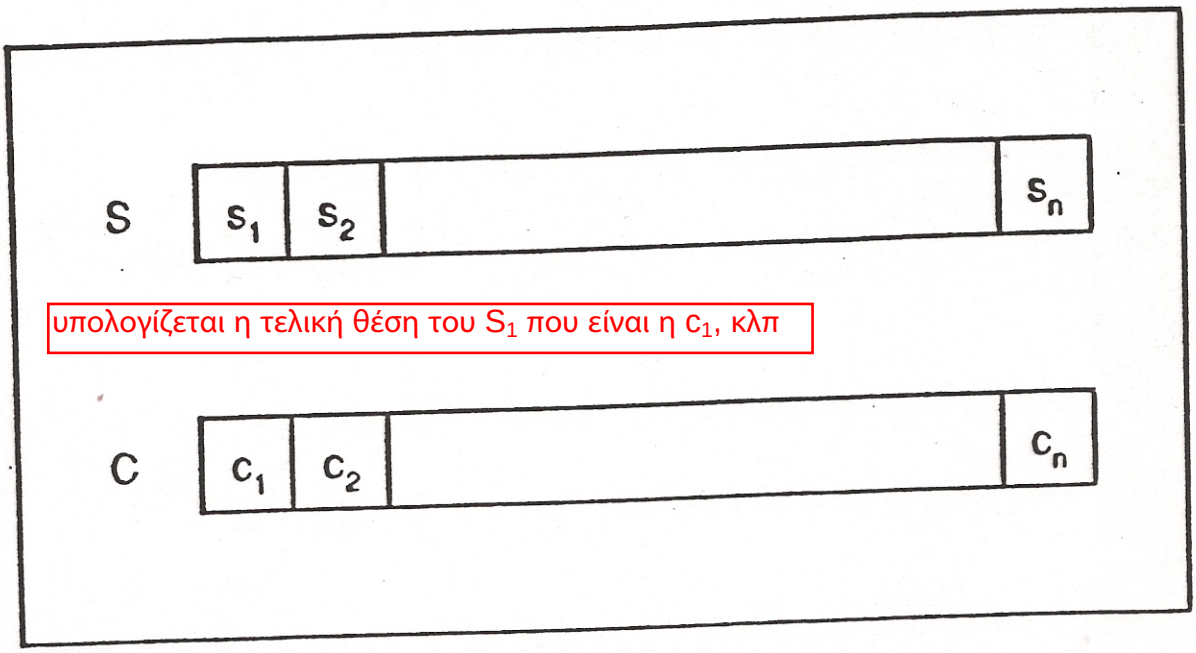
```
1.   procedure ENUM_SORT (n)
2.   begin
3.       for each processor P_{1,j} do
4.           C[j] := 0;
5.       for each processor P_{i,j} do
6.           if (A[i] < A[j]) or (A[i] = A[j] and i < j) then
7.               C[j] := 1;
8.           else
9.               C[j] := 0;
10.      for each processor P_{1,j} do
11.          A[C[j]] := A[j];
12.  end ENUM_SORT
```

Ταξινόμηση στο CRCW Μοντέλο

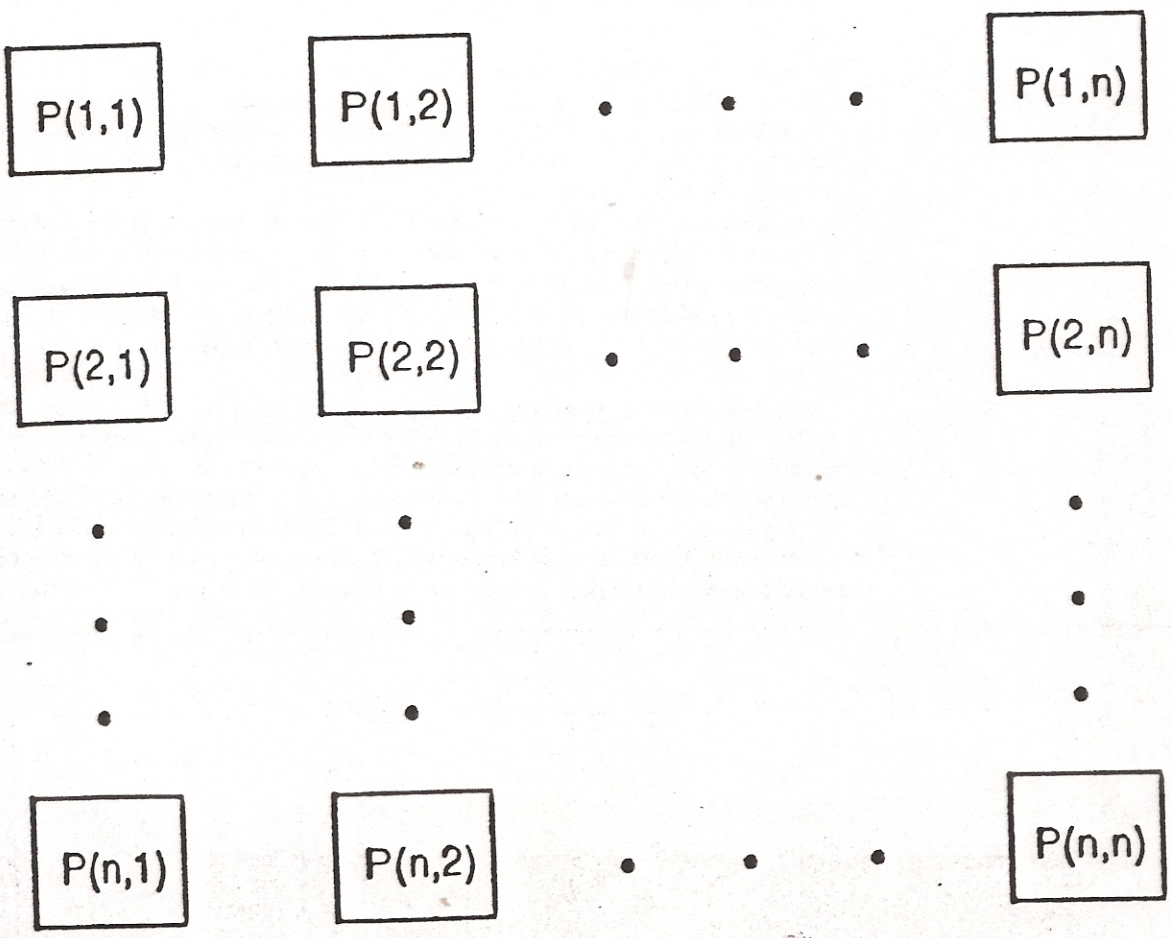
Υποθέτουμε ότι έχουμε το SM SIMD CRCW μοντέλο, όπου υποθέτουμε ότι οι συγκρούσεις γραφής (προσπάθεια γραφής διαφορετικών δεδομένων στην ίδια θέση μνήμης) επιλύονται με την αποθήκευση του αθροίσματος των ακεραίων στη διεύθυνση αυτή.

Υποθέτουμε επίσης ότι n^2 επεξεργαστές είναι διαθέσιμοι στο CRCW μοντέλο για την ταξινόμηση της ακολουθίας $S = \{s_1, s_2, \ldots, s_n\}$. Ο αλγόριθμος της ταξινόμησης βασίζεται στην ιδέα της ταξινόμησης με αναρίθμηση. Η θέση του κάθε στοιχείου s_i της S στην ταξινομημένη ακολουθία προσδιορίζεται με τον υπολογισμό του c_i που συμβολίζει το πλήθος των στοιχείων μικρότερα από αυτό. Αν $s_i = s_j$ τότε όσον μεγαλύτερο λαμβάνεται το s_i αν $i > j$, διαφορετικά λαμβάνεται το s_j. Από τη στιγμή που έχει υπολογιστεί όλα τα c_i, το s_i τοποθετείται στη θέση $1 + c_i$ της ταξινομημένης ακολουθίας. Για ευκολία υποθέτουμε ότι οι

ΔΙΑΜΟΙΡΑΖΟΜΕΝΗ ΜΝΗΜΗ



Διάταξη Επεξεργαστών



Ταξινόμηση δτο CRCW μοντέλο.

$S = \{5, 1, 4, 5\}$

	S	P(1,1)	P(1,2)	P(1,3)	P(1,4)	C	S
S_1	5	5, 5	5, 2	5, 4	5, 5	2	2
S_2	2	2, 5	2, 2	2, 4	2, 5	0	4
S_3	4	4, 5	4, 2	4, 4	4, 5	1	5
S_4	5	5, 5	5, 2	5, 4	5, 5	3	5

Όλοι της 1ης γραμμής διαβάζουν S_1, της 2ης διαβάζουν S_2 κλπ

όλοι της 1ης στήλης διαβάζουν το 5 όλοι της 2ης στήλης το 2, της 3ης το 4 και της 4ης το 5

2=0+1+1+0

0=0+0+0+0

Το 5 το βάζουμε στη θέση c1+1=2+1=3
Το 2 το βάζουμε στη θέση c2+1=0+1=1, κλπ

1=0+1+0+0

3=1+1+1+0

- Κάθε στοιχείο s_i διαβάζεται ταυτόχρονα από όλους τους επεξεργαστές στην i γραμμή και i στήλη
- Όλοι οι επεξεργαστές σε μία δεδομένη γραμμή βρίσκουν στην ίδια θέση μενίσμς.

$t(n) = O(1)$!
$p(n) = n^2$
$c(n) = \Omega(n^2)$ όχι βέλτιστο κόστος

Procedure CRCW SORT (S)

Βήμα 1: **για** i=1 **μέχρι** n **κάνε παράλληλα**
για j=1 **μέχρι** n **κάνε παράλληλα**
αν ($s_i > s_j$) **ή** ($s_i = s_j$ **και** $i > j$)
τότε ο P(i,j) γράφει 1 στην c_i
αλλιώς ο P(i,j) γράφει 0 στην c_i
τέλος αν
τέλος για
τέλος για.

Βήμα 2: **για** i=1 **μέχρι** n **κάνε παράλληλα**
Ο P(i,1) αποθηκεύει το s_i στη θέση $1+c_i$ του S
τέλος για

επεξεργαστές αντιστοιχούν ένα διδιάστατο πίνακα όπως φαίνεται στο σχήμα. Η διαμοιραζόμενη μνήμη περιέχει δύο πίνακες: Η αρχική ακολουθία έχει καταχωρηθεί στον πίνακα S, ενώ το πλήθος των στοιχείων που είναι μικρότερα του s_i, c_i έχουν καταχωρηθεί στον πίνακα C. Η τελική ταξινομημένη ακολουθία καταχωρείται στον πίνακα S. Η i-οστή σειρά των επεξεργαστών "είναι υπεύθυνη για το στοιχείο s_i: Οι επεξεργαστές P(i,1), P(i,2), ..., P(i,n) υπολογίζουν το c_i και αποθηκεύουν το s_i στη θέση c_i+1.

ΤΑΞΙΝΟΜΗΣΗ ΣΤΟ CREW ΜΟΝΤΕΛΟ

Σκοπός μας είναι να σχεδιάσουμε έναν αλγόριθμο, ο οποίος δεν απαιτεί ταυτόχρονη χρήση κοινόχρηστης μνήμης από ένα μικρό αριθμό επεξεργαστών. Στη συνέχεια θα χρησιμοποιηθεί παίζει ένας αλγόριθμος συγχώνευσης για την ταξινόμηση (CREW MERGE).

Υποθέτουμε ότι έχουμε ένα CREW SM SIMD υπολογιστή με N επεξεργαστές $P_1, P_2, \ldots, P_N$ για την ταξινόμηση της ακολουθίας $S=\{s_1, s_2, \ldots, s_n\}$, όπου $N \leq n$. Τα στοιχεία της S κατανέμονται ομοιόμορφα μεταξύ των N επεξεργαστών (n/N στοιχεία ανά επεξεργαστή). Κάθε επεξεργαστής ταξινομεί την ακολουθία του με τη χρήση της QUICKSORT. Οι N ταξινομημένες υποακολουθίες συγχωνεύονται ανά δύο, ταυτόχρονα, με τη χρήση της CREW MERGE για κάθε ζευγάρι. Οι προκύπτουσες ακολουθίες συγχωνεύονται ξανά κατά ζεύγη μέχρις ότου μείνει μία ακολουθία με η στοιχεία. Στη συνέχεια δίνεται ο αλγόριθμος, όπου S_j^k συμβολίζει τη συγχωνευμένη ακολουθία και P_j^{Dkj} τον επεξεργαστή που εκτελεί τη συγχώνευση.

Ανάλυση

Η QUICKSORT απαιτεί $O((n/N)\log(n/N))$ χρόνο. Σε κάθε επανάληψη του βήματος 2.3, συγχωνεύονται $\lfloor v/2 \rfloor$ ζεύγη υποακολουθιών με $n/\lfloor v/2 \rfloor$ στοιχεία για κάθε ζεύγος με τη χρήση $N/\lfloor v/2 \rfloor$ επεξεργαστών για κάθε ζεύγος. Η CREW MERGE απαιτεί $(O(n/N + \log n))$ $O([(n/\lfloor v/2 \rfloor)/(N/\lfloor v/2 \rfloor)] + \log(n/\lfloor v/2 \rfloor))$, δηλαδή $\boxed{O(n/N + \log n)}$ χρόνο. Επειδή το βήμα 2.3 επαναλαμβάνεται $\lfloor \log N \rfloor$ φορές, ο συνολικός χρόνος της CREW SORT είναι

$$t(n) = \underbrace{O((n/N)\log(n/N))}_{\text{QUICKSORT}} + O((n/N)\log N + \log n \log N)$$

$$= O((n/N)\log n + \log^2 n).$$

Επειδή, $p(n) = N$, το κόστος είναι

$$c(n) = O(n \log n + N \log^2 n)$$

το οποίο είναι βέλτιστο για $N \leq n/\log n$

Procedure CREW SORT (S)

Βήμα 1: **για** i=1 **μέχρι** N **κάνε παράλληλα**

Ο επεξεργαστής P_i

(1.1) διαβάζει μία διακεκριμένη υποακολουθία S_i του S μεγέθους n/N

(1.2) QUICKSORT (S_i)

(1.3) $S_i^1 \leftarrow S_i$

(1.4) $P_i^1 \leftarrow \{P_i\}$

τέλος για.

Βήμα 2: (2.1) $u \leftarrow 1$

(2.2) $v \leftarrow N$

(2.3) **Οσο** $v > 1$ **κάνε**

(2.3.1) **για** m=1 **μέχρι** $\lfloor v/2 \rfloor$ **κάνε παράλληλα**

(i) $P^{u+1}{}_m \leftarrow P^u{}_{2m-1} \cup P^u{}_{2m}$

(ii) Οι επεξεργαστές του συνόλου $P^{u+1}{}_m$ εκτελούν την CREW MERGE ($S^u{}_{2m-1}, S^u{}_{2m}, S_m{}^{u+1}$)

τέλος για

(2.3.2) **αν** το v είναι περιττό **τότε** (i) $P^{u+1}{}_{\lceil v/2 \rceil} \leftarrow P^u{}_v$

(ii) $S^{u+1}{}_{\lceil v/2 \rceil} \leftarrow S^u{}_v$

τέλος για

(2.3.3) $u \leftarrow u+1$

(2.3.4) $v \leftarrow \lceil v/2 \rceil$

τέλος όσο.

ΠΑΡΑΔΕΙΓΜΑ

Έστω $\Sigma = \{2, 8, 5, 10, 15, 1, 12, 6, 14, 3, 11, 7, 9, 4, 13, 16\}$ και $N = 4$. Έχουμε $n = 16$ και $n/N = 4$ άρα στο βήμα 1 οι ελεξεφαντές P_1, P_2, P_3 και P_4 παραβάνουν τις υποακολουθίες

$S_1 = \{2, 8, 5, 10\}$, $S_2 = \{15, 1, 12, 6\}$

$S_3 = \{14, 3, 11, 7\}$ και $S_4 = \{9, 4, 13, 16\}$

αντίστοιχα, τις οποίες ταξινομούν τοπικά. Στο τέλος του βήματος 1, θα έχουμε

$S_1^1 = \{2, 8, 5, 10\}$, $S_2^1 = \{1, 6, 1, 15\}$

$S_3^1 = \{3, 7, 11, 14\}$, $S_4^1 = \{4, 9, 13, 16\}$

$P_1^1 = \{P_1\}$, $P_2^1 = \{P_2\}$, $P_3^1 = \{P_3\}$, $P_4^1 = \{P_4\}$

Στην πρώτη επανάληψη του βήματος 2.3 οι ελεξεφαντές στο $P_1^2 = P_1^1 \cup P_2^1 = \{P_1, P_2\}$ συγχωνεύουν τα στοιχεία των S_1^1 και S_2^1 για το εκπαιδευτικό της

$S_1^2 = \{1, 2, 5, 6, 8, 10, 12, 15\}$.

Ταυτόχρονα οι επεξεργαστές του $P_2^2 = P_3^1 \cup P_4^1 = \{P_3, P_4\}$ διαχωρίζουν τις S_3^1 και S_4^1 στην $S_2^2 = \{3, 4, 7, 9, 11, 13, 14, 16\}$.
Στη δεύτερη επανάληψη οι επεξεργαστές του $P_1^3 = P_1^2 \cup P_2^2 = \{P_1, P_2, P_3, P_4\}$ διαχωρίζουν τις S_1^2 και S_2^2 στην $S_1^3 = \{1, 2, \ldots, 16\}$.

ΤΑΞΙΝΟΜΗΣΗ ΣΤΟ EREW ΜΟΝΤΕΛΟ

Υποθέτουμε ότι έχουμε στη διάθεσή μας N επεξεργαστές $P_1, P_2, \ldots, P_N$ σε ένα EREW SM SIMD υπολογιστή για την ταξινόμηση της ακολουθίας $S = \{s_1, s_2, \ldots, s_n\}$, όπου $N \leq n$.

Προσομοίωση της CREW SORT

Ο ευκολότερος τρόπος προκειμένου να αποφευχθεί το ταυτόχρονο διάβασμα στην CREW SORT είναι να χρησιμοποιηθεί η MULTIPLE BROADCAST. Η προσομοίωση αυτή της CREW SORT στο EREW μοντέλο απαιτεί

$$t(n) = O((n/N)\log n + \log n \log N) \times \underbrace{\log N}_{\text{BROADCAST}}$$
$$= O([n/N] + \log N] \log n \log N)$$

με κόστος

$$C(n) = O((n + N \log N) \log n \log N)$$

το οποίο δεν είναι βέλτιστο.

Ταξινόμηση χωρίς συγκρούσεις διαμέρισμα.

Η αντικατάσταση της CREW MERGE με την EREW MERGE εξαλείφει το πρόβλημα της σύγκρουσης του διαμερίσματος. Το βήμα αυτό απαιτεί $O(n/N + \log n \log N)$ και επειδή υπάρχουν $\log N$ επαναλήψεις, ο συνολικός χρόνος θα είναι

$$t(n) = O((n/N) \log(n/N)) + O((n/N) \log N + \log n \log^2 N)$$

$$= O([(n/N) + \log^2 n] \log n)$$

με κόστος

$$C(n) = O((n + N \log^2 n) \log n)$$

το οποίο είναι βέλτιστο όταν $N \le n / \log^2 n$

Ταξινόμηση με επιτροπή

Στον αλγόριθμο αυτό χρησιμοποιείται πάλι η QUICKSORT. Σημειώνουμε ότι NLM και μοιράζουμε να χρόνους $N = n^{1-x}$, όπου $0 < x < 1$.

Έστω ότι συμβολίζει το $\lceil i(n/2^{1/x}) \rceil$ μικρότερο στοιχείο της S για $1 \le i \le 2^{1/x} - 1$. Τα διαμοιράζουν την S σε $2^{1/x}$ υποακολουθίες μήκους $n/2^{1/x}$ η καθε μία. Αυτές οι υποακολουθίες συμβολίζονται με

$$S_1, S_2, \ldots, S_j, S_{j+1}, S_{j+2}, \ldots, S_{2j}$$

όπου $j = 2^{(1/x)-1}$ και ικανοποιούν την ιδιότητα: Κάθε στοιχείο της S_i είναι μικρότερο ή ίσο με κάθε στοιχείο της S_{i+1} για $1 \le i \le 2j-1$ (βλ. σχήμα). Η εφαρμογή της υποδιαίρεσης μπορεί τώρα να επεκταθεί αναδρομικά σε κάθε μία από τις υποακολουθίες S_i μέχρις ότου όλη η ακολουθία S ταξινομηθεί!

Ο αλγόριθμος εκτελείται παράλληλα καλώντας αρχικά την PARALLEL SELECT για τον προσδιορισμό των στοιχείων μι- και στη συνέχεια της δημιουργίας των υποακολουθιών S_i. Ο αλγόριθμος εφαρμόζεται παράλληλα στις υποακολουθίες

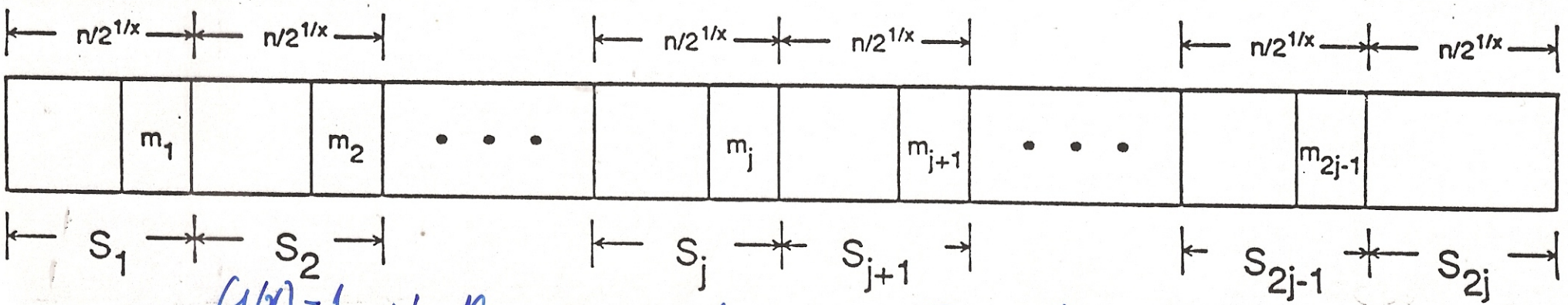
$$S_1, S_2, \ldots, S_{\lfloor \sqrt{n} \rfloor}$$

χρησιμοποιώντας $N/\lfloor \sqrt{n} \rfloor$ επεξεργαστές για κάθε υποακολουθία. Στη συνέχεια βρίσκει το ίδιο για κάθε υποακολουθία από τις

$$S_{\lfloor \sqrt{n}+1 \rfloor}, S_{\lfloor \sqrt{n}+2 \rfloor}, \ldots, S_{2\lfloor \sqrt{n} \rfloor}.$$

Ας υποθέσει ότι ο αριθμός των επεξεργαστών για την εξεταζόμενη κάθε υποακολουθίας μεγέθους $n/2^{1/k}$ είναι $n^{1-x}/2^{(1/k)-1}$ δηλαδή ακριβώς ίσος με όσο απαιτεί μια κανονική διαδρομή $(n/2^{1-x})^{1-x}$. Θα πρέπει να τονιστεί ότι το $2^{1/k}$ δεν είναι ένας ακέραιος πεπερασμένου μεγέθους, πράγμα που εξασφαλίζει την ύπαρξη ενός ορίου στον εκτελέσιμο χρόνο καθώς και ότι

$N \leq n$, $N = n^{1-x}$, $0 < x < 1$ και m_i $\lceil i(n/2^{1/x}) \rceil$-οστό μικρότερο στοιχείο της S, $1 \leq i \leq 2^{1/x} - 1$. Τα m_i διαιρούν την S σε $2^{1/x}$ υποακολουθίες μεγέθους $n/2^{1/x}$ η κάθε μία. $2j = 2^{1/x}$



$j = 2^{(1/x)-1}$. Κάθε στοιχείο της S_i είναι μικρότερο ή ίσο από κάθε στοιχείο της S_{i+1} για $1 \leq i \leq 2j-1$.

Σημειώνεται ότι η ακολουθία S_i (βλ. βήματα 2-4) δημιουργείται με τη μέθοδο που εκτίθεται με την PARALLEL SELECT. Επίσης στο βήμα 3 τα στοιχεία της S' μικρότερα από το m_i και μεγαλύτερα ή ίσα του m_{i-1} τοποθετούνται πρώτα στην S_i. Αν $|S_i| \leq \lceil |S|/k \rceil$, τότε στοιχεία ίσα με το m_i προστίθενται στο S_i έτσι ώστε είτε $|S_i| = \lceil |S|/k \rceil$ ή δεν υπάρχουν (έχουν απορριφθεί) για να προστεθούν στην S_i. Τα βήματα 2 και 4 εκτελούνται αναδρομικά.

Παράδειγμα

Έστω $S = \{5, 9, 12, 16, 18, 2, 10, 13, 17, 4, 7, 18, 18, 11, 3, 17, 20, 19, 14, 8, 5, 17, 1, 11, 15, 10, 6\}$, δηλ. $n = 27$ και έχουμε 5 επεξεργαστές P_1, P_2, P_3, P_4 και P_5 δηλ. $N = 5$, δηλαδή $5 = 27^{1-x} \Rightarrow x \simeq 0.5$, $k = 2^{\lceil 1/x \rceil} = 4$. Άρα $m_1 = 6$, $m_2 = 11$ και $m_3 = 17$. Στο βήμα 5

εφαρμόζεται αναδρομικά ο αλγόριθμος και ταυτόχρονα στις S_1 και S_2. Επειδή $|S_1| = |S_2| = 7$ και $\lceil 7^{1-x} \rceil = 2$ άρα 2 ελεύθερες χρησιμοποιούνται για την κατασκευή κάθε υποακολουθίας S_1 και S_2 ο πίνακας παραμένει αναλλοίωτος. Για την S_1 οι ελεύθερες P_1 και P_2 υπολογίζουν τα $m_1 = 2$, $m_2 = 4$ και $m_3 = 5$ και δημιουργούνται οι ζεύγαρες υποακολουθίες $\{1,2\}$, $\{3,4\}$, $\{5,5\}$ και $\{6\}$ κάθε μία από τις οποίες είναι ήδη ταξινομημένη. Για την S_2, οι ελεύθερες P_3 και P_4 υπολογίζουν τα $m_1 = 8$, $m_2 = 10$ και $m_3 = 11$ και τις υποακολουθίες $\{7,8\}$, $\{9,10\}$, $\{10,11\}$, και $\{11\}$ που ήδη είναι ταξινομημένες. Στο βήμα 6 ο αλγόριθμος εφαρμόζεται αναδρομικά και ταυτόχρονα στις S_3 και S_4. Επειδή $|S_3| = 7$ και $|S_4| = 6$, τα 7^{1-x} και 6^{1-x}

Procedure EREW SORT (S)

$k = 2^{\lceil 1/x \rceil}$

αν $|S| \leq k$
τότε QUICKSORT (S)
αλλιώς (1) **για** i=1 **μέχρι** k-1 **κάνε**
PARALLEL SELECT (S, $\lceil i|S|/k \rceil$) {βρίσκει το m_i}
τέλος για
(2) $S_1 \leftarrow \{s \in S : s \leq m_1\}$
(3) **για** i=2 **μέχρι** k-1 **κάνε**
$S_i \leftarrow \{s \in S : m_{i-1} \leq s \leq m_i\}$
τέλος για
(4) $S_k \leftarrow \{s \in S : s \geq m_{k-1}\}$
(5) **για** i=1 **μέχρι** k/2 **κάνε παράλληλα**
EREW SORT (S_i)
τέλος για
(6) **για** i=(k/2)+1 **μέχρι** k **κάνε παράλληλα**
EREW SORT (S_i)
τέλος για
τέλος για.

Ν/ι επεξεργαστές $(n^{1-x}/2^{(1/x)-1}) = (n/2^{1/x})^{1-x}$.

Ν/ι επεξεργαστές/S_i

όλα τα m_i υπάρχουν. Αρχικά οι Ν διαθέσιμοι επεξεργαστές υπολογίζουν το x από την $N = n^{1-x}$. Αν το x δεν ικανοποιεί τις συνθήκες (i) $\lceil 1/x \rceil \leq 10$[1] (π.χ) και (ii) $n \geq 2^{\lceil 1/x \rceil}$[2] τότε ο μικρότερος πραγματικός αριθμός μεταξύ 0 και 1 που ικανοποιεί την (i) και (ii) λαμβάνεται ο x. Έστω $k = 2^{\lceil 1/x \rceil}$ τότε ο αλγόριθμος δίνεται από τον EREW SORT.

1 Εξασφαλίζει ότι ο $2^{1/x}$ είναι ακέραιος περιορισμένα μεγέθους. 2 Εξασφαλίζει ότι τα m_i θα βρεθούν.

$k = 2^{\lceil 1/x \rceil}$ $\Rightarrow$ $k=4, j=2$ $j = 2^{1/x-1}$ $1 \le i \le 2j-1$,

S

5	19	12	16	18	2	10	13	17	4	7	18	18	11	3	17	20	19	14	8	5	17	1	11	15	10	6

Αρχικά

m_1 m_2 m_3

5	2	4	3	5	1	6	9	10	7	8	10	11	11	12	16	13	14	5	17	17	18	18	18	20	19	17

S_1 S_2 S_3 S_4

Μετά το βήμα (4)

1	2	3	4	5	5	6	7	8	9	10	10	11	11	12	16	13	14	15	17	17	18	18	18	20	19	17

S_3 S_4

Μετά το βήμα (5)

1	2	3	4	5	5	6	7	8	9	10	10	11	11	12	13	14	15	16	17	17	17	18	18	18	19	20

Μετά το βήμα (6).

στρογγυλεύονται στο 2 και έτσι δυο επεξεργαστές χρησιμοποιούνται για την ταξινόμηση κάθε μιας από τις δύο υποακολουθίες S_3 και S_4. Για την S_3, $m_1 = 13$, $m_2 = 15$ και $m_3 = 17$ και δημιουργούνται οι τέσσερις υποακολουθίες $\{12, 13\}$, $\{14, 15\}$, $\{16, 17\}$ και $\{17\}$ κάθε μία από τις οποίες είναι ήδη ταξινομημένη. Για την S_4, έχουμε $m_1 = 18$, $m_2 = 18$ και $m_3 = 20$, οπότε προκύπτουν οι τέσσερις υποακολουθίες $\{17, 18\}$, $\{18, 18\}$, $\{19, 20\}$ και μία κενή. Η ακολουθία S μετά το βήμα 5 παρουσιάζεται στο Σχήμα

Ανάλυση

Ο χρόνος για τον EREW SORT είναι

$$t(n) = \underbrace{cn^x}_{\text{PARALLEL SELECT}} + 2t(n/k)$$

$$= O(n^x \log n)$$

Επειδή $p(n) = n^{1-x}$

$$c(n) = p(n) \cdot t(n) = O(n \log n)$$

το οποίο είναι βέλτιστο.