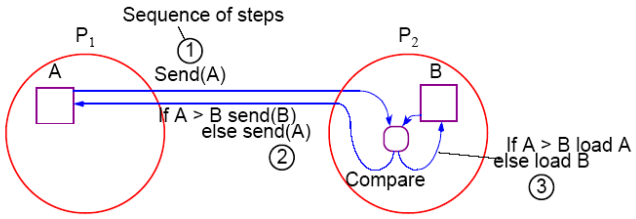
Αλγόριθμοι ταξινόμησης

• Το πρόβλημα της ταξινόμησης είναι η επαναδιάταξη των αριθμών μίας λίστας σε αύξουσα (ή φθίνουσα) διάταξη.
• Οι βέλτιστοι ακολουθιακοί αλγόριθμοι οι οποίοι δεν χρησιμοποιούν ειδικές ιδιότητες των στοιχείων εισόδου έχουν πολυπλοκότητα χειρότερης περίπτωση O(*n*log*n*) όπου n είναι το πλήθος των στοιχείων που ταξινομούνται.
• Επομένως, η βέλτιστη πολυπλοκότητα ενός παράλληλου αλγόριθμου ταξινόμησης με n διεργασίες θα είναι O(n log n)/n = O(log n). Πράγματι υπάρχει παράλληλος αλγόριθμος που επιτυγχάνει αυτή την πολυπλοκότητα αλλά η σταθερά που κρύβεται στον ασυμπτοτικό συμβολισμό είναι πολύ μεγάλη.
• Γενικά, ένας πρακτικός O(log n) αλγόριθμος με n επεξεργαστές είναι ένας στόχος ο οποίος δεν είναι εύκολο να επιτευχθεί με αλγόριθμους ταξινόμησης βασιζόμενοι μόνο σε συγκρίσεις (comparison-based sorting algorithms).
Η λειτουργία compare-and-exchange αποτελεί τη βάση των περισσότερων ακολουθιακών αλγόριθμων ταξινόμησης. Σε αυτή τη λειτουργία, δύο αριθμητικές μεταβλητές, A και B, συγκρίνονται. Αν *A* > *B* τότε οι *A* and *B* εναλλάσσονται, δηλ.:

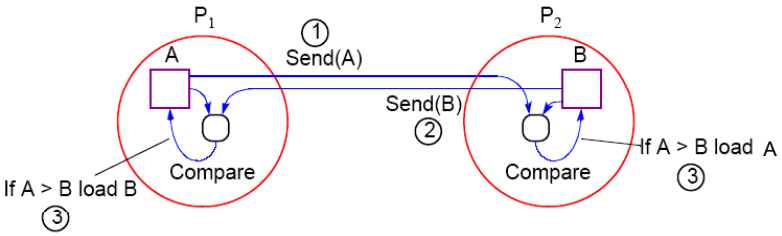
```
if (A > B) {
        temp = A;
        A = B;
        B = temp;
}
```

• Η λειτουργία compare-and-exchange ταιριάζει σε ένα σύστημα κατανεμημένης μνήμης.

• Στο σχήμα που ακολουθεί, απεικονίζεται ο πρώτος τρόπος υλοποίησης της compare-and-exchange με δύο διεργασίες οι οποίες μπορούν να ανταλλάξουν μηνύματα.



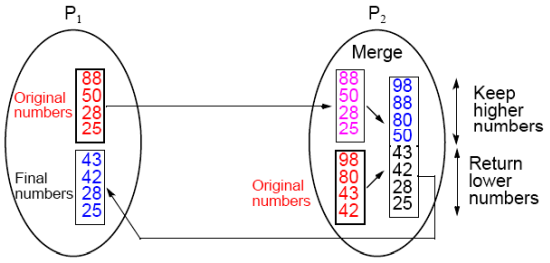
• Η P_1 στέλνει το περιεχόμενο της *A* στη διεργασία P_2. Η P_2 συγκρίνει την A και τη *B* και στέλνει το περιεχόμενο της B στη διεργασία P_1 αν η *A* είναι μεγαλύτερη της *B* (διαφορετικά επιστρέφει ξανά το περιεχόμενο της A στη *P*1).
• Στο επόμενο σχήμα έχουμε μία εναλλακτική υλοποίηση:

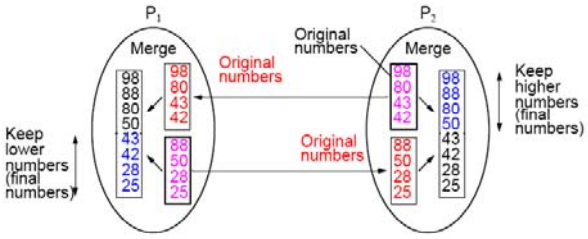


• Η P_1 στέλνει το περιεχόμενο της μεταβλητής *A* στη διεργασία P_2 και η διεργασία P_2 στέλνει το περιεχόμενο της μεταβλητής *B* στη διεργασία P_1. Στη συνέχεια, και οι δύο διεργασίες εκτελούν τις λειτουργίες σύγκρισης . Η P_1 κρατάει το μικρότερο από τους δύο αριθμούς *A* και *B* και η P_2 κρατάει το μεγαλύτερο των αριθμών *A* και *B*.

Χωρισμός Δεδομένων (Data Partitioning)

• Συνήθως το πλήθος *n* των στοιχείων που θα ταξινομηθούν είναι πολύ μεγαλύτερο από το πλήθος *p* των επεξεργαστών/διεργασιών. Σε αυτή την περίπτωση κάθε διεργασία αναλαμβάνει *n*/*p* στοιχεία.
• Η λειτουργία compare-and-exchange γενικεύεται σε αυτή τη περίπτωση όπως φαίνεται στο διπλανό σχήμα
• Η διεργασία P_1 στέλνει τα ταξινομημένα δεδομένα της στη διεργασία P_2. Η P_2 συγχωνεύει τη ταξινομημένη λίστα που λαμβάνει από τη P_1 με την τοπική της ταξινομημένη λίστα και τελικά στέλνει το πρώτο μισό της συγχωνευμένης λίστας στη διεργασία P_1. Υπενθυμίζεται ότι ο ακολουθιακός αλγόριθμος για τη συγχώνευση δύο ταξινομημένων λιστών n και m στοιχείων αντίστοιχα απαιτεί n+m-1 συγκρίσεις στη χειρότερη περίπτωση.
• Εναλλακτικά, σε αντιστοιχία με τον δεύτερο τρόπο υλοποίησης της compare-and-exchange, κάθε διεργασία στέλνει την ταξινομημένη λίστα της στην άλλη διεργασία όπως φαίνεται στο σχήμα. Στη συνέχεια και οι δύο διεργασίες εκτελούν τη λειτουργία της συγχώνευσης και κρατούν το τμήμα της συγχωνευμένης λίστας που τους αντιστοιχεί.

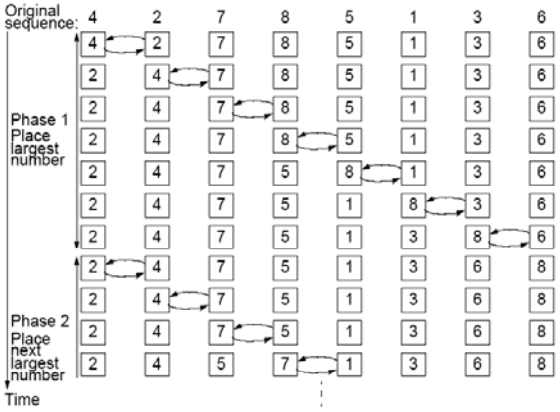




Αλγόριθμος Bubble Sort

Ο αλγόριθμος αυτός εκτελεί n-1 περάσματα σε μία λίστα n στοιχείων. Στο πρώτο πέρασμα, το μεγαλύτερο στοιχείο τοποθετείται στο τέλος της λίστας μετά από μία σειρά συγκρίσεων και εναλλαγών οι οποίες αρχίζουν από την αρχή της λίστας.

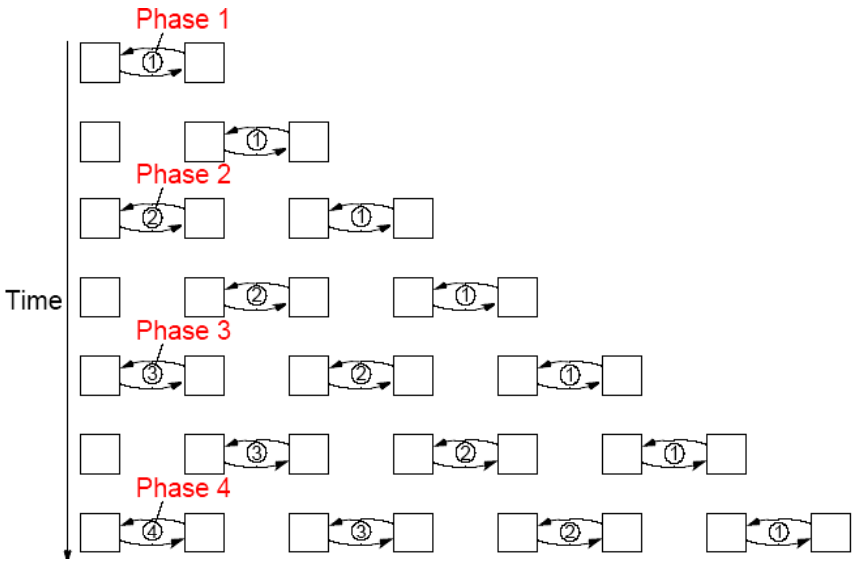
Στο i-στο πέρασμα, οι i-1 μεγαλύτεροι αριθμοί έχουν τοποθετηθεί στις τελευταίες θέσεις του πίνακα και ο αλγόριθμος προχωρεί στην ταξινόμηση των n-i+1 στοιχείων στις πρώτες θέσεις της λίστας.
Στο τέλος του (n-1)-οστού περάσματος, η λίστα έχει προκύψει ταξινομημένη



Είναι εύκολο να δει κανείς ότι το πλήθος των λειτουργιών compare-and-exchange είναι n(n+1)/2 και επομένως αν κάθε τέτοια λειτουργία απαιτεί χρόνο O(1), η συνολική πολυπλοκότητα θα είναι $O(n^2)$.

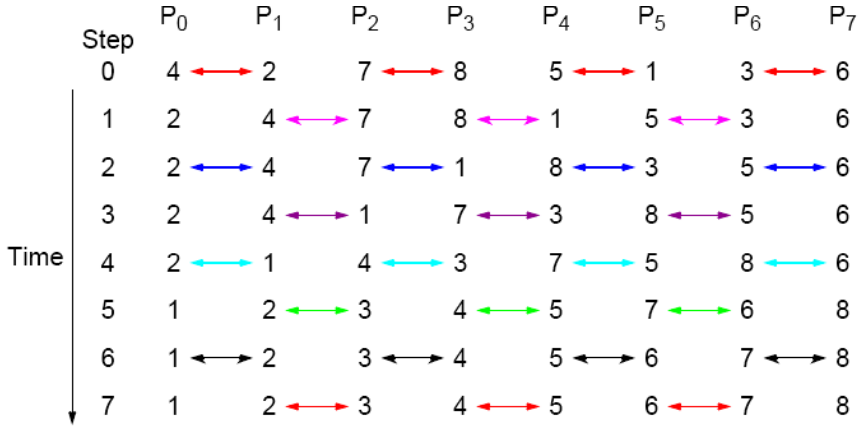
Παράλληλος αλγόριθμος Bubble Sort

• Χρησιμοποιώντας την τεχνική σωλήνωσης, μπορούν να εκτελεστούν, την ίδια χρονική στιγμή, λειτουργίες compare-and-exchange από διαφορετικές φάσεις.
• Η φάση i δεν μπορεί να «προσπεράσει» τη φάση i-1 κατά την επεξεργασία της.
• Επίσης, σε οποιαδήποτε χρονική στιγμή, δεν θα πρέπει το ίδιο στοιχείο να εμπλέκεται σε δύο διαφορετικές λειτουργίες compare-and-exchange

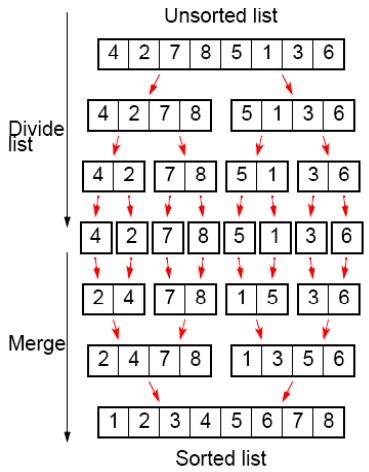


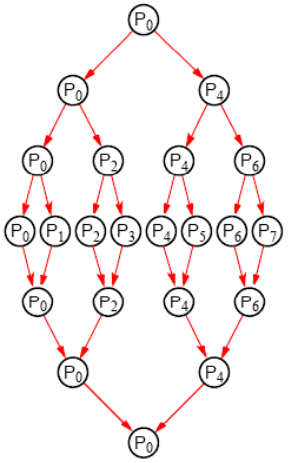
Αλγόριθμος ταξινόμησης Odd-Even (Transposition) Sort

• Ο αλγόριθμος αυτός αποτελεί παραλλαγή του bubble sort.
• Λειτουργεί σε δύο φάσεις που εναλλάσσονται, την ζυγή φάση και την περιττή
• Στη ζυγή φάση, οι διεργασίες με ζυγό αναγνωριστικό συγκρίνουν/ανταλλάσουν τα δεδομένα τους με το δεξιό τους γείτονα.
• Στη μονή φάση, οι διεργασίες με μονό αναγνωριστικό συγκρίνουν/ανταλλάσουν δεδομένα με τον δεξιό τους γείτονα.
• Ο αλγόριθμος αυτός αποτελείται από n φάσεις συνολικά και κάθε φάση απαιτεί το χρόνο μίας σύγκρισης. Άρα, η πολυπλοκότητα του αλγορίθμου είναι Θ(n).



Mergesort





• Ο αλγόριθμος mergesort είναι ένας κλασσικός ακολουθιακός αλγόριθμος ο οποίος χρησιμοποιεί την τακτική διαίρει και βασίλευε. Η μη ταξινομημένη λίστα πρώτα διαιρείται στα δύο. Κάθε μισό επίσης διαιρείται επίσης στα δύο και αυτό συνεχίζεται μέχρι να προκύψουν λίστες του ενός στοιχείου.
• Στη συνέχεια, ζεύγη στοιχείων συγχωνεύονται σε μία ταξινομημένη λίστα δύο αριθμών. Ζεύγη αυτών των λιστών συγχωνεύονται σε λίστες των τεσσάρων αριθμών και στη συνέχεια αυτές οι λίστες συγχωνεύονται σε λίστες των 8 στοιχείων κοκ., μέχρι να προκύψει ολόκληρη η λίστα ταξινομημένη.
• Για την παράλληλη υλοποίηση αυτού του αλγορίθμου, οι διεργασίες επικοινωνούν σε δενδρική διάταξη. Αρχικά η διεργασία P_0 έχει όλα τα δεδομένα και τα δίνει στη διεργασία P_4. Στη συνέχεια, οι δύο αυτές διεργασίες στέλνουν το δεύτερο μισό της λίστας που διαθέτουν στις διεργασίες P_2 και P_6 αντίστοιχα κοκ. Οι λειτουργίες της συγχώνευσης ακολουθούν την αντίστροφη πορεία.

- Ένα σημαντικό μειονέκτημα αυτής της παράλληλης υλοποίησης είναι ότι ο φόρτος επεξεργασίας δεν κατανέμεται ομοιόμορφα στις διάφορες διεργασίες.
- Αρχικά, μόνο μία διεργασία είναι ενεργή, στη συνέχεια 2, μετά 4 κοκ. Κατά τη φάση της συγχώνευσης, αυτό το μοτίβο επεξεργασίας αντιστρέφεται.
- Η πολυπλοκότητα του ακολουθιακού αλγορίθμου είναι O(nlogn).
- Ο παράλληλος αλγόριθμος αποτελείται από 2logn βήματα. Συγκεκριμένα, στη φάση της διαίρεσης, έχουμε τις εξής πολυπλοκότητες επικοινωνίας

  $t_{startup} + (n/2)\, t_{data}$ P_o -> P_4
  $t_{startup} + (n/4)\, t_{data}$ P_o -> P_2, P_4 -> P_6
  $t_{startup} + (n/8)\, t_{data}$ P_o -> P_1, P_2 -> P_3, P_4 -> P_5, P_6 -> P_7

  .....
- Στη φάση της συγχώνευσης θα έχουμε τις εξής πολυπλοκότητες επικοινωνίας

  ……..
  $t_{startup} + (n/8)\, t_{data}$ P_o <- P_1, P_2 <- P_3, P_4 <- P_5, P_6 <- P_7
  $t_{startup} + (n/4)\, t_{data}$ P_o <- P_2, P_4 <- P_6
  $t_{startup} + (n/2)\, t_{data}$ P_o <- P_4
- Αν αθροίσουμε τις παραπάνω πολυπλοκότητες, η συνολική πολυπλοκότητα επικοινωνίας του αλγορίθμου θα είναι $t_{comm} \sim 2 \log p\, t_{startup} + 2nt_{data}$
- Για τη πολυπλοκότητα υπολογισμού, παρατηρούμε ότι συγκρίσεις γίνονται μόνο στη φάση της συγχώνευσης. Στο βήμα i της φάσης συγχώνευσης, κάθε διεργασία εκτελεί συγχώνευση δύο λιστών 2^{i-1} στοιχείων και εκτελεί 2^i-1 συγκρίσεις για την συγχώνευση αυτή. Αφού η φάση της συγχώνευσης έχει log p φάσεις συνολικά, η συνολική πολυπλοκότητα θα είναι:

$$t_{comp} = \sum_{i=1}^{\log p} (2^i - 1) = O(p)$$

Αν το πλήθος των διεργασιών είναι μικρότερο από το πλήθος των στοιχείων για ταξινόμηση, κάθε διεργασία αναλαμβάνει περισσότερα από ένα στοιχεία

Στο προηγούμενο κώδικα, η συγχώνευση των δύο λιστών εκτελείται από μία μόνο διεργασία. Αυτή η επεξεργασία μπορεί να υλοποιηθεί και αυτή παράλληλα.
Έστω A[1...m] , B[1..,n] οι δύο ταξινομημένες λίστες που πρέπει να συγχωνευθούν, έστω ότι m≥n και C[1,...,m+n] το αποτέλεσμα της συγχώνευσης.
Αναζητούμε στη λίστα B με δυαδική αναζήτηση το μεσαίο στοιχείο της A , δηλ. το στοιχείο A[q] με q=int((1+m)/2)). Έστω ότι το στοιχείο A[q] είναι μεταξύ των στοιχείων B[r] και B[r+1].
Στη συνέχεια το πρόβλημα ανάγεται στην εκτέλεση δύο συγχωνεύσεων ζευγών λιστών. Το πρώτο ζεύγος είναι οι λίστες A[1,..,q-1] και B[1,...,r] ενώ το δεύτερο ζεύγος είναι οι λίστες A[q+1,..,m] και B[r+1,...,n].
Ο ψευδοκώδικας θα έχει ως εξής:

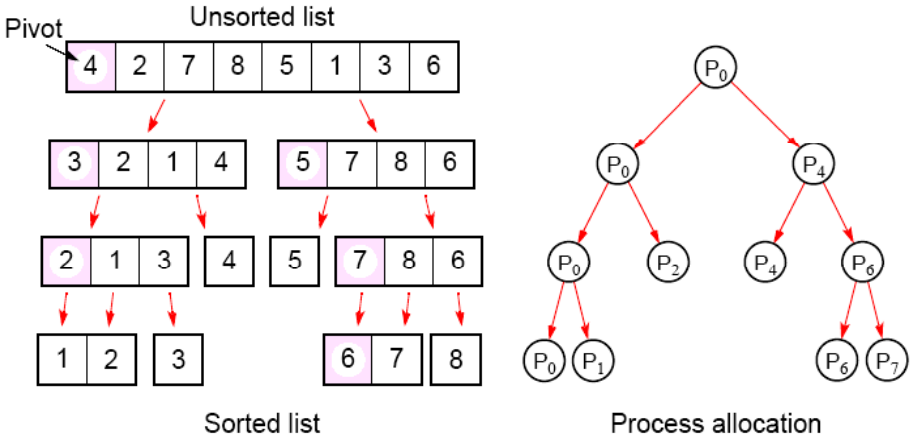
```
Parallel_merge (A[1...m],B[1...n],C[1...n+m])
    q=int((1+m)/2)
    r=Binary_Search(A[q], B[1..n])
    C[q+r]=A[q]
    Parallel do:
      Parallel_merge(A[1...q-1],B[1...r], C[1,...q+r-1])
      Parallel_merge(A[q+1,...,m],B[r+1,...n], C[q+r+1...m+n])
    End do
```

Ο χρόνος εκτέλεσης της δυαδικής αναζήτησης θα είναι Θ(log n).
Ο χρόνος T(n,m) για τη συγχώνευση των A και B θα είναι $T(n,m)=\max\{T(q-1,r), T(m-q, n-r)\} + \Theta(\log n)$. Αν n=m μπορεί να αποδειχθεί ότι $T(n,n)=O(\log^2 n)$.

Ο αλγόριθμος Quicksort

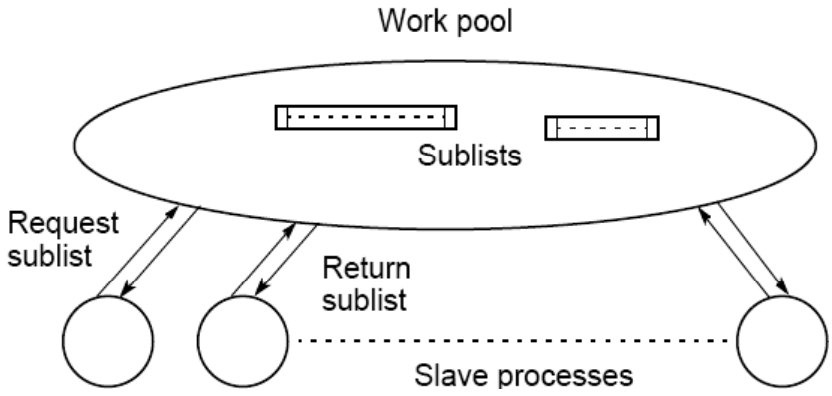
- Ο αλγόριθμος αυτός είναι ένας πολύ γνωστός ακολουθιακός αλγόριθμος ταξινόμησης, ο οποίος έχει μέση πολυπλοκότητα χρόνου O(*n*log*n*).
- Στον αλγόριθμο αυτό η αρχική λίστα διαιρείται σε δύο υπολίστες. Τα στοιχεία της μίας υπολίστας θα πρέπει να είναι μικρότερα από τα στοιχεία της άλλης υπολίστας.
- Αυτό επιτυγχάνεται με την επιλογή ενός αριθμού, του στοιχείου οδηγού. Κάθε ένα από τα υπόλοιπα στοιχεία της λίστας συγκρίνεται με το στοιχείο οδηγό και ανάλογα το αποτέλεσμα της σύγκρισης τοποθετείται στη μία ή την άλλη υπολίστα.
- Το στοιχείο οδηγός μπορεί να είναι οποιοδήποτε στοιχείο στη λίστα αλλά συνήθως επιλέγεται το πρώτο στοιχείο. Στη συνέχεια το στοιχείο οδηγός καταχωρείται σε μία από τις δύο λίστες ή φυλάσσεται και τοποθετείται στη τελική του θέση αφού ολοκληρωθεί η επεξεργασία.
- Η παραπάνω διαδικασία επαναλαμβάνεται αναδρομικά στις δύο υπολίστες μέχρι να προκύψουν λίστες του ενός στοιχείου. Σε αυτό το σημείο, η διαδοχική τοποθέτηση των λιστών αυτών δίνει και το τελικό ταξινομημένο αποτέλεσμα.
- Για την παράλληλη υλοποίηση του αλγορίθμου Quicksort, ανάλογη μέθοδος με αυτή της παράλληλης υλοποίησης του Mergesort ακολουθείται. Συγκεκριμένα, η διεργασία P_0 χωρίζει την αρχική λίστα σε δύο υπολίστες σύμφωνα με το στοιχείο οδηγό. Στη συνέχεια, η P_0 κρατάει τη πρώτη υπολίστα που περιέχει τα μικρότερα στοιχεία, και δίνει την δεύτερη υπολίστα στη διεργασία P_4
- Στη συνέχεια, οι διεργασίες P_0 και P_4 υποδιαιρούν περαιτέρω τις λίστες τους και δίνουν τη δεύτερη υπολίστα στις διεργασίες P_2 και P_6.
- Η διαδικασία συνεχίζεται μέχρι να προκύψουν λίστες ενός στοιχείου.

Παράλληλη Υλοποίηση του Quicksort



• Αν υποθέσουμε ότι σε κάθε βήμα έχουμε ισο-διάσπαση, μπορούμε να δούμε ότι χρόνος υπολογισμού είναι $t_{comp} \approx 2n$ και ότι ο χρόνος επικοινωνίας είναι $t_{comm} \approx \log p\, t_{startup} + nt_{data.}$
• Όμως, η ισοδιάσπαση είναι ιδανική υπόθεση και συνήθως οι λίστες που προκύπτουν μετά τη διάσπαση δεν έχουν το ίδιο μέγεθος. Στη χειρότερη περίπτωση αυτό μπορεί να οδηγήσει σε συνολική πολυπλοκότητα $O(n^2)$.
• Βασικό πρόβλημα τόσο στη παράλληλη υλοποίηση του Mergesort όσο και του Quicksort είναι ότι η αρχική διαίρεση της λίστας σε δύο υπολίστες γίνεται από μία μόνο διεργασία (διεργασία P_0). Αυτό έχει ως αποτέλεσμα τη σημαντική επιβράδυνση ολόκληρου του υπολογισμού.

Η τεχνική της δεξαμενής έργων μπορεί να χρησιμοποιηθεί στην παράλληλη υλοποίηση του quicksort. Σε αυτή την περίπτωση, οι λίστες που προκύπτουν από τις διασπάσεις είναι τα διαθέσιμα έργα προς εκτέλεση. Μετά τη διάσπαση, κάθε διεργασία κρατάει τη μία υπολίστα και επιστρέφει την άλλη στη δεξαμενή έργων. Όταν, μία διεργασία ολοκληρώσει την επεξεργασία στη λίστα της, παίρνει την επόμενη διαθέσιμη υπολίστα από τη δεξαμενή έργων.



Εναλλακτικός τρόπος υλοποίησης του Quicksort σε σύστημα διαμοιραζόμενης μνήμης

Θεωρούμε μία λίστα μεγέθους n η οποία κατανέμεται ομοιόμορφα σε p διεργασίες.

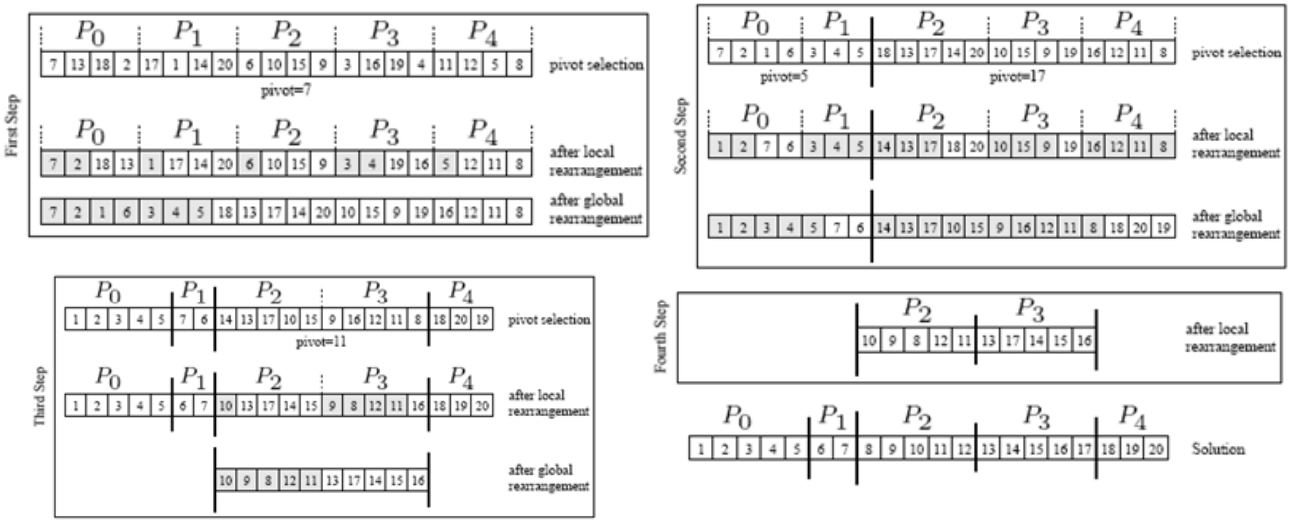
Μία από τις διεργασίες επιλέγει ένα στοιχείο οδηγό και ενημερώνει τις υπόλοιπες διεργασίες για την τιμή του.

[Local rearrangement]. Κάθε διεργασία χωρίζει τη λίστα σε δύο, L_i και U_i, με βάση την τιμή του στοιχείου οδηγού.

[Global rearrangement]. Τα στοιχεία στη διαμοιραζόμενη μνήμη αναδιατάσσονται κατά τέτοιο τρόπο ώστε όλες οι λίστες L_i να τοποθετηθούν σε συνεχόμενες θέσεις μνήμης και ανάλογα πράττουμε για τις λιστες U_i. Έστω *L* η λίστα που δημιουργείται από αυτή την τοποθέτηση των λιστών L_i και *U* η αντίστοιχη λίστα για τις λίστες U_i.

Το σύνολο των επεξεργαστών χωρίζονται σε δύο ομάδες (σε αναλογία με το μέγεθος των λιστών *L* και *U*). Στη συνέχεια, η παραπάνω διαδικασία εφαρμόζεται αναδρομικά στις λίστες *L* και *U*.
Η αναδρομή ολοκληρώνεται όταν οι λίστες *L* και *U* καταχωρηθούν σε μία μόνο διεργασία. Σε αυτή την περίπτωση, η διεργασία ταξινομεί τη λίστα *L* ή την *U* χρησιμοποιώντας τον ακολουθιακό quicksort.

Παράδειγμα εκτέλεσης

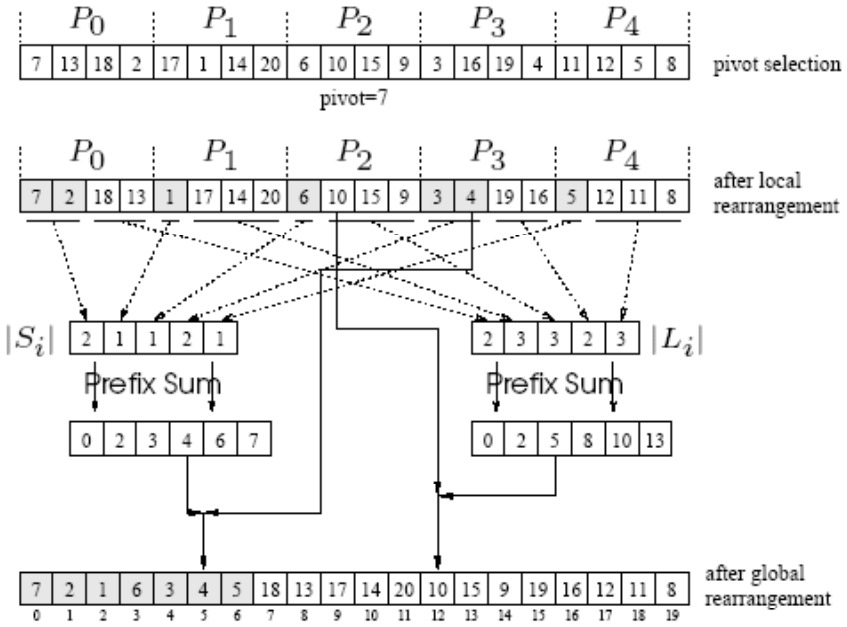


• Δεν έχουμε περιγράψει την συνένωση των λιστών L_i και U_i στις λίστες L και U αντίστοιχα. Το πρόβλημα είναι ο προσδιορισμός της σωστής θέσης κάθε στοιχείου στην τελική συνενωμένη λίστα.

• Κάθε διεργασία υπολογίζει το πλήθος των στοιχείων της τοπικής λίστας που είναι μικρότερα και αντίστοιχα μεγαλύτερα από το στοιχείο οδηγό.

• Στη συνέχεια, εκτελείται ένας υπολογισμός prefix-sum και προσδιορίζεται με αυτό τον τρόπο η θέση κάθε στοιχείου στη λίστες L και U.

• Από τη στιγμή που αυτές οι θέσεις είναι γνωστές, κάθε διεργασία μπορεί να τοποθετήσει τα δεδομένα της στις λίστες L και U.



Αν υποθέσουμε ότι σε κάθε βήμα οι λίστες ισοδιασπώνται γύρω από το στοιχείο οδηγό, μπορούμε να αποδείξουμε ότι συνολική πολυπλοκότητα του αλγορίθμου θα είναι:

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

Ο Αλγόριθμος Odd-Even Mergesort

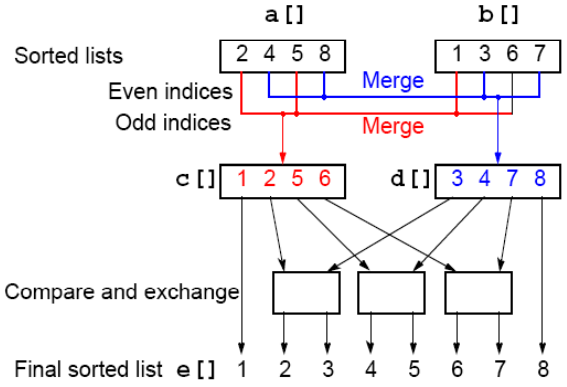
Ο αλγόριθμος Odd-Even Mergesort βασίζεται στη ρουτίνα odd-even merge αλγόριθμο ο οποίος συγχωνεύει δύο λίστες n στοιχείων η κάθε μία.

Συγκεκριμένα, αν δύο λίστες εισόδου είναι οι $a_1, a_2, a_3, \ldots, a_n$ και $b_1, b_2, b_3, \ldots, b_n$ (όπου n είναι δύναμη του 2), ο odd-even merge αλγόριθμος εκτελεί τις ακόλουθες ενέργειες:

1. Τα στοιχεία με μονούς δείκτες από κάθε λίστα, δηλ. $a_1, a_3, a_5, \ldots, a_{n-1}$ και $b_1, b_3, b_5\ldots, b_{n-1}$ συγχωνεύονται σε μία ταξινομημένη λίστα $c_1, c_2, c_3,\ldots, c_n$.
2. Τα στοιχεία με ζυγούς δείκτες από κάθε λίστα, δηλ. $a_2, a_4, a_6, \ldots, a_n$ και $b_2, b_4, b_6\ldots, b_n$ συγχωνεύονται σε μία ταξινομημένη λίστα $d_1, d_2, d_3,\ldots, d_n$.
3. Η τελική ταξινομημένη λίστα $e_1, e_2, \ldots, e_{2n}$ προκύπτει ως εξής:

$$e_{2i}=min\{c_{i+1}, d_i\}$$
$$e_{2i+1}=max\{c_{i+1}, d_i\}$$

Η συγχώνευση στα βήματα 1 και 2 πραγματοποιείται με αναδρομική εκτέλεση του odd-even merge αλγόριθμου

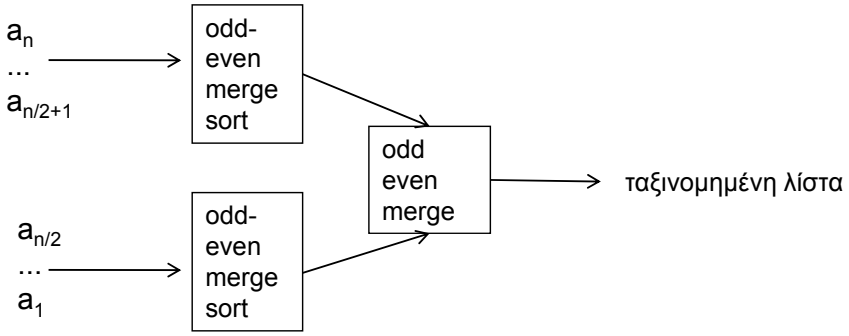


Με n επεξεργαστές, η συνολική πολυπλοκότητα του odd-even merge αλγόριθμου θα είναι O(log n).

Ο αλγόριθμος Odd-Even Mergesort

ο αλγόριθμος αυτός μπορεί να ορισθεί αναδρομικά ως εξής:

```
odd-even-mergesort(a1,a2,.....,an) {
        if (n>1) then
                return odd-even-merge(odd-even-mergesort(a1,a2,...,an/2),
                odd-even-merge-sort(an/2+1,...,an));
        else
                return an
}
```



Η πολυπλοκότητα του Odd-Even Mergesort δίνεται από την ακόλουθη αναδρομική σχέση: $T(n)=T(n/2)+\Theta(\log n)$ που έχει συνολική πολυπλοκότητα $\Theta(\log^2 n)$

Ο Αλγόριθμος Bitonic Mergesort

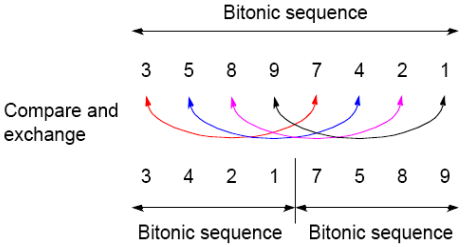
• Ο αλγόριθμος αυτός βασίζεται στις ιδιότητες των διτονικών (bitonic) ακολουθιών.
• Καταρχήν, διτονική χαρακτηρίζεται μία ακολουθία αριθμών $a_0, a_1, ..., a_{n-1}$ όταν αποτελείται από δύο υπακολουθίες $a_0,a_1,...,a_i$ και $a_{i+1},a_{i+2}...,a_{n-1}$ όπου η πρώτη υπακολουθία είναι αύξουσα και η δεύτερη φθίνουσα δηλ.

$$a_0 < a_1 < a_2 < a_3, \ldots, a_{i-1} < a_i > a_{i+1}, \ldots, a_{n-2} > a_{n-1}$$

για κάποια τιμή i ($0 <= i < n$).
• Μία ακολουθία χαρακτηρίζεται επίσης διτονική όταν μπορεί να προέλθει από τη παραπάνω ακολουθία με μία κυκλική ολίσθηση (αριστερή ή δεξιά). Π.χ. η ακολουθία (2,4,6,5,3,1) είναι διτονική ακολουθία καθώς και η ακολουθία (3,1,2,4,6,5) αφού μπορεί να προκύψει από τη πρώτη με κυκλική ολισθηση.
• Κάθε διτονική ακολουθία $A=\{a_0,a_1,..., a_{n-1}\}$ έχει την εξής σημαντική ιδιότητα:

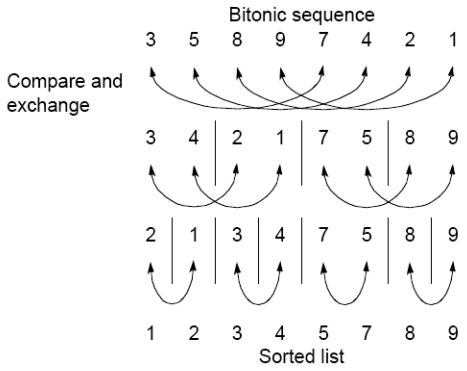
Αν $c_i = \min \{a_i, a_{i+n/2}\}$ και $d_i = \max\{a_i,a_{i+n/2}\}$ όπου i=0,..,n/2-1 , τότε οι ακολουθίες $C=\{c_0,c_1,...,c_{n/2-1}\}$ και $D=\{d_0,d_1,...,d_{n/2-1}\}$ είναι διτονικές και όλα τα στοιχεία της C είναι μικρότερα από τα στοιχεία της D.

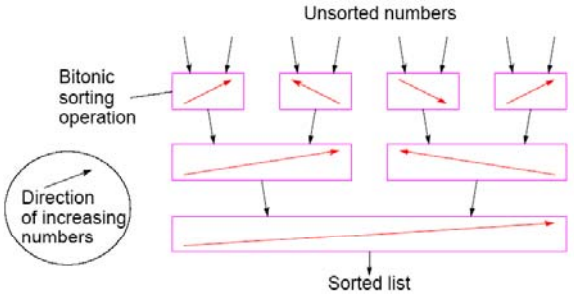
Αφού η παραπάνω διαδικασία τοποθετεί τα μικρότερα στοιχεία στη C και τα μεγαλύτερα στοιχεία στη D, αν επαναλάβουμε τη διαδικασία αυτή αναδρομικά στις λίστες C και D, τελικά θα λάβουμε τα στοιχεία της A ταξινομημένα.



Παράδειγμα

Παράδειγμα ταξινόμησης διτονικής ακολουθίας 8 στοιχείων





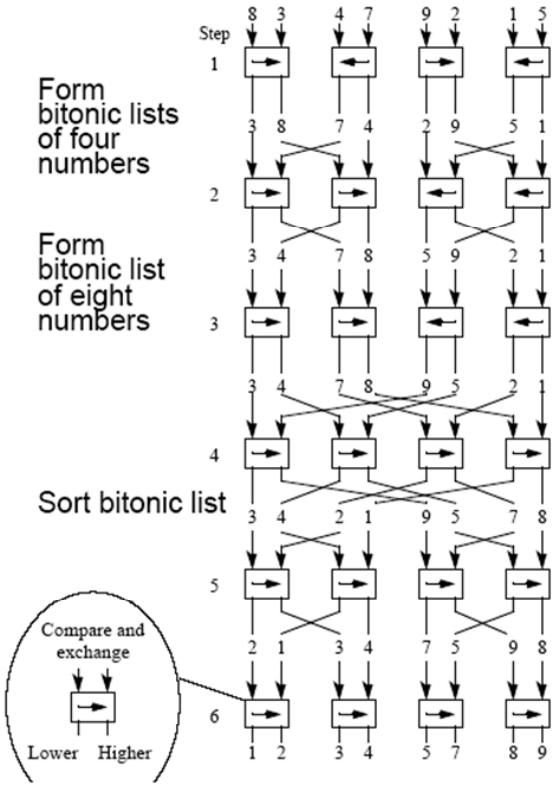
Γενικός αλγόριθμος ταξινόμησης με βάση τον αλγόριθμο ταξινόμησης διτονικής ακολουθίας.

• Η ταξινόμηση μίας διτονικής ακολουθίας n στοιχείων με n διεργασίες / επεξεργαστές απαιτεί χρόνο O(log n) .
• Ο αλγόριθμος ταξινόμησης μίας διτονικής ακολουθίας μπορεί να χρησιμοποιηθεί για την ταξινόμηση οποιαδήποτε ακολουθίας αριθμών. Η βασική ιδέα είναι να συγχωνεύουμε τα στοιχεία σε όλο και μεγαλύτερες διτονικές ακολουθίες, αρχίζοντας από τα γειτονικά στοιχεία της ακολουθίας.
• Αρχικά, τα γειτονικά στοιχεία σχηματίζουν αύξουσες και φθίνουσες ακολουθίες στοιχείων
• Έτσι, με τη χρήση του αλγόριθμου ταξινόμησης διτονικών ακολουθιών, ζεύγη γειτονικών ακολουθιών μπορούν να δώσουν εναλλασσόμενες αύξουσες και φθίνουσες ακολουθίες τεσσάρων στοιχείων.
• Επαναλαμβάνοντας την παραπάνω διαδικασία, προκύπτουν διτονικές ακολουθίες αυξανόμενου μήκους.
• Στο τελικό βήμα, η μοναδική διτονική ακολουθία ταξινομείται με τον αλγόριθμο ταξινόμησης διτονικών ακολουθιών

Συνολικά έχουμε κ εφαρμογές του αλγόριθμου ταξινόμησης διτονικών ακολουθιών ($n=2^κ$).

Στην i-οστη εφαρμογή του , ο αλγόριθμος εκτελείται σε i βήματα.

Το συνολικό πλήθος βημάτων θα είναι $1+2+..+κ=κ(κ+1)/2=Θ(\log^2 n)$.



Παράδειγμα ταξινόμησης 8 στοιχείων

Αλγόριθμοι Bucket sort και Sample sort

Βασική υπόθεση στον αλγόριθμο Bucket sort είναι ότι τα στοιχεία του πίνακα εισόδου παίρνουν τιμές σε ένα συγκεκριμένο διάστημα τιμών $[a,b]$ ($l=b-a+1$)

Το διάστημα αυτό υποδιαιρείται σε m διαστήματα (buckets) ίσου μήκους . Έτσι ο i-στος κάδος περιέχει στοιχεία στο διάστημα $[(i-1)*l/m...i*l/m-1]$ (i=1...m) .

Κάθε στοιχείο ανάλογα την τιμή του τοποθετείται στο κατάλληλο κάδο (bucket).

Αν τα στοιχεία κατανέμονται ομοιόμορφα στο διάστημα $[a,b]$, οι κάδοι αναμένονται να έχουν περίπου το ίδιο πλήθος στοιχείων.

Τα στοιχεία κάθε κάδου ταξινομούνται τοπικά και στη συνέχεια τα ταξινομημένο περιεχόμενο κάθε κάδου τοποθετούνται στο τελικό πίνακα. Πρώτα τοποθετούνται τα περιεχόμενα του1ου κάδου, στη συνέχεια του 2ου κοκ.

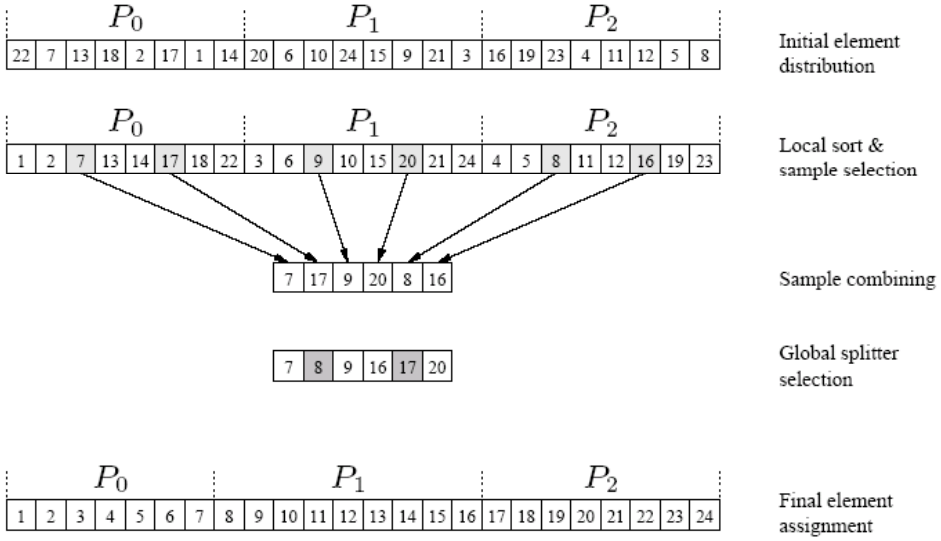
Ο συνολικός χρόνος του ακολουθιακού αλγορίθμου θα είναι $\Theta(n\log(n/m))$ εφόσον ισχύει η υπόθεση της ισοκατανομής των στοιχείων στους m κάδους.

Η παράλληλη υλοποίηση του bucket sort είναι σχετικά απλή. Αν έχουμε p διεργασίες, επιλέγουμε m=p δηλ. ο κάδος i αντιστοιχεί στη διεργασία/επεξεργαστή i . Επομένως κάθε διεργασία είναι υπεύθυνη για ένα συγκεκριμένο εύρος τιμών.

Κάθε διεργασία χωρίζει τα στοιχεία της σύμφωνα με τους m κάδους. Τα στοιχεία του i-οστου κάδου στέλνονται στην διαδικασία i. Επομένως, το βήμα αυτό απαιτεί επικοινωνία κάθε διεργασίας με όλες τις υπόλοιπες και αποστολή διαφορετικών δεδομένων σε κάθε διεργασία.

Μετά το βήμα επικοινωνίας, κάθε διεργασία i θα έχει στη διάθεση της όλα τα στοιχεία του αρχικού πίνακα που ανήκουν στο κάδο i. Στη συνέχεια , κάθε διεργασία ταξινομεί τα στοιχεία αυτά και ο τελικός ταξινομημένος πίνακας προκύπτει κατανεμημένος στις διεργασίες.

- Ή διαίρεση του εύρους τιμών [a,..,b] σε m ίσου μήκους διαστήματα, δεν οδηγεί συνήθως σε ισοκατανομή των στοιχείων του πίνακα στους διάφορους κάδους.
- Ο αλγόριθμος sample sort αντιμετωπίζει αυτό το πρόβλημα ορίζοντας διαφορετικά το εύρος τιμών κάθε κάδου. Συγκεκριμένα, θα πρέπει να επιλεγούν κατά τέτοιο τρόπο τα όρια των κάδων (στοιχεία-splitters) έτσι ώστε σε κάθε κάδο να καταλήξουν το ίδιο περίπου πλήθος στοιχείων με μεγάλη πιθανότητα.
- Η επιλογή των splitters γίνεται με τον εξής τρόπο:
  - Αρχικά ο πίνακας των n στοιχείων διαιρείται σε p blocks των n/p(=l) στοιχείων το καθένα όπου p είναι το πλήθος διαθέσιμων διεργασιών. Η διεργασία i αναλαμβάνει το i-οστο block (ι=1,..,p).
  - Κάθε διεργασία ταξινομεί το δικό της block [b1,b2,...,bl] με εφαρμογή του αλγόριθμου quicksort και στη συνέχεια επιλέγει p-1 στοιχεία, συγκεκριμένα τα στοιχεία i*(l/p) (i=1,..,p-1).
  - Συνολικά, υπάρχουν p(p-1) στοιχεία που έχουν επιλεχθεί με αυτή τη διαδικασία. Όλα αυτά τα στοιχεία συλλέγονται σε μία διεργασία η οποία και τα ταξινομεί.
  - Στη συνέχεια, η παραπάνω διεργασία επιλέγει πάλι ομοιόμορφα p-1 στοιχεία από τα p(p-1) ταξινομημένα στοιχεία και τα στέλνει στις υπόλοιπες διεργασίες
  - Τα στοιχεία αυτά αποτελούν τα στοιχεία-splitters που ορίζουν τους p κάδους.
  - Μπορεί να αποδειχθεί ότι κάθε ένα από τα p buckets θα έχει το πολύ $2n/\mathrm{p}$ στοιχεία.
  - Ο υπόλοιπος αλγόριθμος ακολουθεί τη λογική του bucket sort, δηλ.
    - συγκέντρωση από κάθε διεργασία των στοιχείων που ανήκουν στον κάδο που έχει αναλάβει.
    - Ταξινόμηση των στοιχείων των κάδων.



Στο παραπάνω παράδειγμα, εφαρμόζεται ο sample-sort για την ταξινόμηση ενός πίνακα 24 στοιχείων.
Σε πρώτη φάση, κάθε διεργασία αφού ταξινομήσει τα στοιχεία της, επιλέγει δύο splitters.
Οι έξι συνολικά splitters συλλέγονται σε μία διεργασία η οποία τα ταξινομεί και στη συνέχεια επιλέγει τους δύο τελικούς splitters (στοιχεία 8 και 17)
Κάθε διεργασία συλλέγει από τις υπόλοιπες τα στοιχεία του bucket που τις αντιστοιχεί και στη συνέχεια τα ταξινομεί.

- Η πολυπλοκότητα του παράλληλου sample sort μπορεί να προσδιορισθεί ως εξής:
- Η εσωτερική ταξινόμηση των n/p στοιχείων απαιτεί χρόνο $\Theta((n/p)\log(n/p))$, και η επιλογή των p-1 στοιχείων απαιτεί χρόνο $\Theta(p)$.
- Ο χρόνος για την συλλογή των p(p-1) στοιχείων θέλει χρόνο $\Theta(p^2)$, ο χρόνος για την τοπική ταξινόμηση των στοιχείων αυτών είναι $\Theta(p^2 \log p)$, και η επιλογή πάλι $p-1$ στοιχείων απαιτεί χρόνο $\Theta(p)$.
- Κάθε διεργασία μπορεί να χωρίσει τα στοιχεία της σε p κάδους αναζητώντας τους p-1 splitters στον τοπικό του ταξινομημένο πίνακα με p-1 δυαδικές αναζητήσεις και σε συνολικό χρόνο $\Theta(p\log(n/p))$.
- Ο χρόνος για την αποστολή των στοιχείων κάθε κάδου από κάθε διεργασία στη διεργασία που έχει αναλάβει τον συγκεκριμένο κάδο απαιτεί συνολικά χρόνο $O(n/p)$.
- Συνολικά, η πολυπλοκότητα του αλγορίθμου θα είναι:
  $T=\Theta(n/p \log n/p + p^2 \log p + p \log n/p + n/p)$