Bucket και Radix sorting

- Η υπόθεση στους αλγόριθμους που εξετάσαμε: οι αντιπρόσωποι (keys) είναι στοιχεία ενός γραμμικά διαταγμένου συνόλου.
- Υπάρχουν όμως περιπτώσεις που γνωρίζουμε πιο πολλά για τους αντιπροσώπους: π.χ. 5-ψήφιοι αριθμοί.
- Αλγόριθμοι «BUCKET» (κάδος): χωρίζουν πρώτα τους αντιπροσώπους σε σωρούς.
  - Στον ίδιο σωρό πάνε όσα ονόματα αρχίζουν από *A,* σε άλλο σωρό όσα αρχίζουν από *B,* κ.λπ, ή: σε άλλον σωρό όσοι αριθμοί αρχίζουν από 1**,** στον ίδιο όσοι αρχίζουν από 2 κ.λπ.
  - Μετά, διάταξη - εσωτερικά - του κάθε σωρού χωριστά και τέλος, συνδυάζονται οι σωροί για να κατασκευαστεί η διαταγμένη λίστα.
  - Ο κάθε σωρός μπορεί να διαταχθεί με τον ίδιο τρόπο οπότε έχουμε αναδρομικό αλγόριθμο.

- Ο αλγόριθμος έχει τρεις φάσεις και ότι η κάθε φάση απαιτεί το δικό της είδος πράξεων.
- **Φάση Κατανομής** (σε σωρούς):
  - *k* σωροί.
  - κάθε στοιχείο εξετάζεται μία φορά, π.χ. συγκρίνοντάς με *k* τιμές που αντιστοιχούν σε (είναι ενδεικτικές για) κάθε έναν από τους *k* σωρούς, για να αποφασίσει σε ποιο σωρό ανήκει ο αντιπρόσωπος.
  - Μετά, ο αντιπρόσωπος τοποθετείται στον σωρό του: O(n) η πολυπλοκότητα αυτής της φάσης.
- **Φάση διάταξης** (του κάθε σωρού):
  - Έστω ότι για την διάταξη του κάθε σωρού χρησιμοποιείται ένας αλγόριθμος που απαιτεί *S(m)* συγκρίσεις, όταν ο σωρός έχει *m* στοιχεία.
  - n_i στοιχεία στο σωρό i=1,…,k, για όλους τους k σωρούς θα χρειαστούν

$$\sum_{i=1}^{k} S(n_i)$$

- **Φάση συνδυασμού** (των διαταγμένων πλέον σωρών για να κατασκευαστεί η τελική, διαταγμένη, λίστα)
  - Στην χειρότερη περίπτωση, μπορεί να χρειαστεί να αντιγράψει (να μεταφέρει) ο αλγόριθμος όλα τα στοιχεία, ένα προς ένα, από τους σωρούς στην λίστα. Άρα, η προσπάθεια είναι ανάλογη του: O(n) η πολυπλοκότητα της φάσης.

- Έστω *S(m)=cmlogm,* και όλοι οι σωροί έχουν τον ίδιο αριθμό στοιχείων μετά την κατανομή, δηλαδή $n_i=n/k$.
- Τότε από την Φάση Διάταξης θα έχουμε αριθμό συγκρίσεων:

$$\sum_{i=1}^{k} c(n/k)\log(n/k) = \sum_{i=1}^{k} cn\log(n/k)$$

- Όσο μεγαλύτερος είναι ο αριθμός των σωρών *k* τόσο λιγότερες συγκρίσεις χρειάζονται. Όμως
- Δεν είναι πάντα εφικτός ο καθορισμός του *k* (και η επιτυχία να είναι *ni=n/k),*

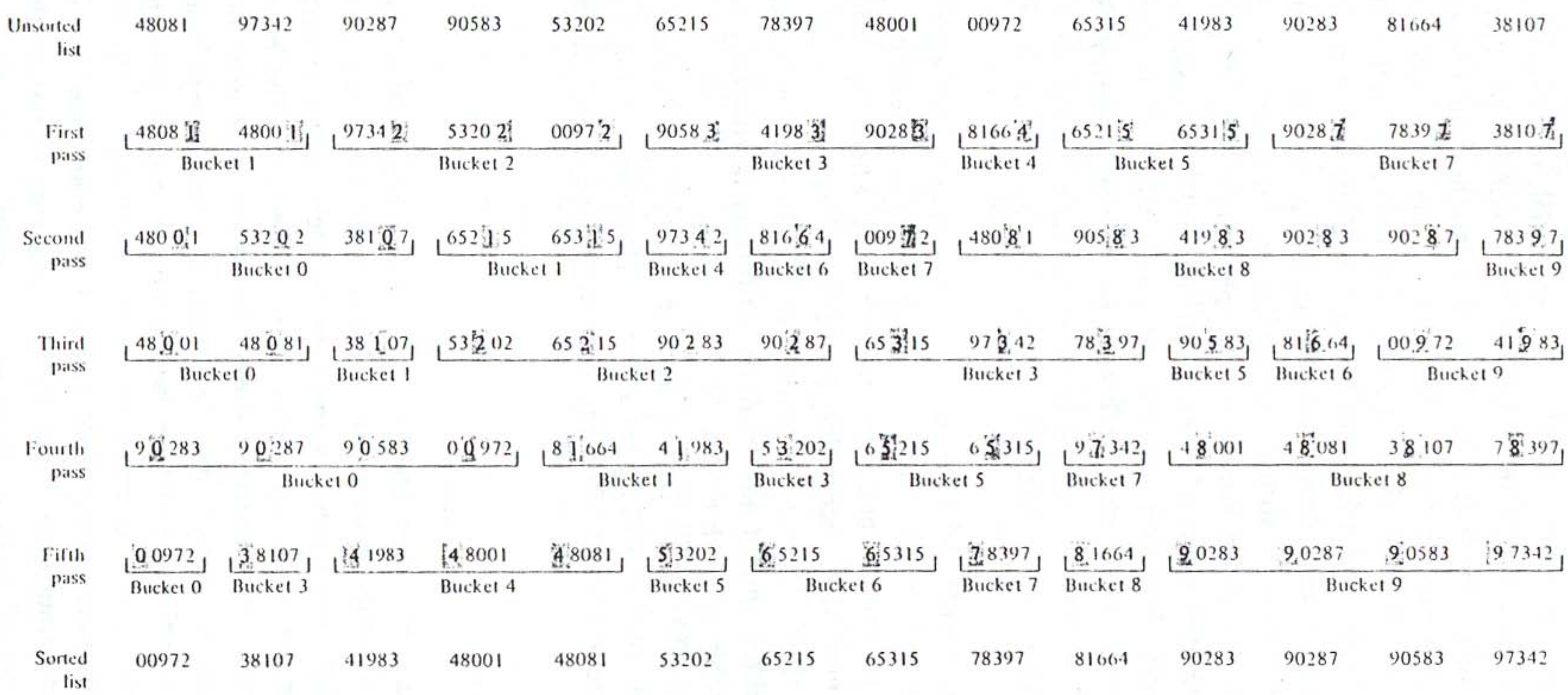
  εκτός αν γνωρίζουμε κι άλλες λεπτομέρειες για το συγκεκριμένο πρόβλημα.
- Μια άλλη λύση: η αναδρομική εφαρμογή του *BUCKETSORT* για τη

  δημιουργία όλο και μικρότερων σωρών.
- Αυτό όμως επιβαρύνει σημαντικά «τα λογιστικά» του αλγορίθμου και, γενικά, δεν συμφέρει.

Radix sort

- Τυπική περίπτωση εφαρμογής της μεθόδου “Radix" είναι όταν οι αντιπρόσωποι είναι ακέραιοι, π.χ. 5-ψήφιοι.
- Ένας αναδρομικός αλγόριθμος θα μπορούσε αρχικά να χωρίσει τους αριθμούς σε 10 σωρούς:
  – ανάλογα με το πρώτο ψηφίο (αν δηλαδή το πρώτο ψηφίο είναι 0,1,...,9), μετά να χωρίσει κάθε σωρό σε 10 σωρούς ανάλογα με το δεύτερο ψηφίο κ.λπ.
  – Το πρόβλημα εξακολουθεί:
    - δημιουργούνται όλο και πιο πολλοί «υπόσωροί» άλλων υποσωρών: τηρείται αυστηρή λογιστική μέχρι να γίνει πλήρης διάταξη.
  – Αν ο χωρισμός γίνεται από το τελευταίο ψηφίο (ή bit, ή γράμμα, ή πεδίο) τότε οι σωροί μπορούν «να συγκολληθούν» σ' ένα μεγάλο σωρό προτού προχωρήσει κανείς σε άλλο διαχωρισμό σύμφωνα με το επόμενο (από δεξιά προς τα αριστερά) στοιχείο
  – Απαλοιφή του προβλήματος της λογιστικής και, με την αναδρομικότητα, δεν χρησιμοποιείται άλλος αλγόριθμος για εσωτερική διάταξη σωρών.

Radix sort


Unsorted list
48081 97342 90287 90583 53202 65215 78397 48001 00972 65315 41983 90283 81664 38107
First pass
48081 48001
Bucket 1
97342 53202 00972
Bucket 2
90583 41983 90283
Bucket 3
81664
Bucket 4
65215 65315
Bucket 5
90287 78397 38107
Bucket 7
Second pass
48001 53202 38107
Bucket 0
65215 65315
Bucket 1
97342
Bucket 4
81664
Bucket 6
00972
Bucket 7
48081 90583 41983 90283 90287
Bucket 8
78397
Bucket 9
Third pass
48001 48081
Bucket 0
38107
Bucket 1
53202 65215 90283 90287
Bucket 2
65315 97342 78397
Bucket 3
90583
Bucket 5
81664
Bucket 6
00972 41983
Bucket 9
Fourth pass
90283 90287 90583 00972
Bucket 0
81664 41983
Bucket 1
53202
Bucket 3
65215 65315
Bucket 5
97342
Bucket 7
48001 48081 38107 78397
Bucket 8
Fifth pass
00972
Bucket 0
38107
Bucket 3
41983 48001 48081
Bucket 4
53202
Bucket 5
65215 65315
Bucket 6
78397
Bucket 7
81664
Bucket 8
90283 90287 90583 97342
Bucket 9
Sorted list
00972 38107 41983 48001 48081 53202 65215 65315 78397 81664 90283 90287 90583 97342

Συμβολή και Mergesort

Δίνονται δύο διαταγμένες λίστες (αντιπροσώπων) A και B. Συμβολή είναι η κατασκευή μιας νέας διαταγμένης λίστας C, που περιέχει τα στοιχεία των A και B.

Απλή λύση: Συγκρίνονται το πρώτο από τα εναπομένοντα στοιχεία της A με το πρώτο από τα εναπομένοντα στοιχεία της B. Το μικρότερο τοποθετείται στην C.

ΠΑΡΑΔΕΙΓΜΑ

$A=\{3, 10, 12, 16\}$ $B=\{8, 14, 15\}$

Σύγκριση 3 με 8 $\Rightarrow$

A	B	C
10	8	3
12	14	
16	15	

Σύγκριση 10 με 8 $\Rightarrow$

A	B	C
10	14	3
12	15	8
16		

Σύγκριση 10 με 14 $\Rightarrow$

A	B	C
12	14	3
16	15	8
		10

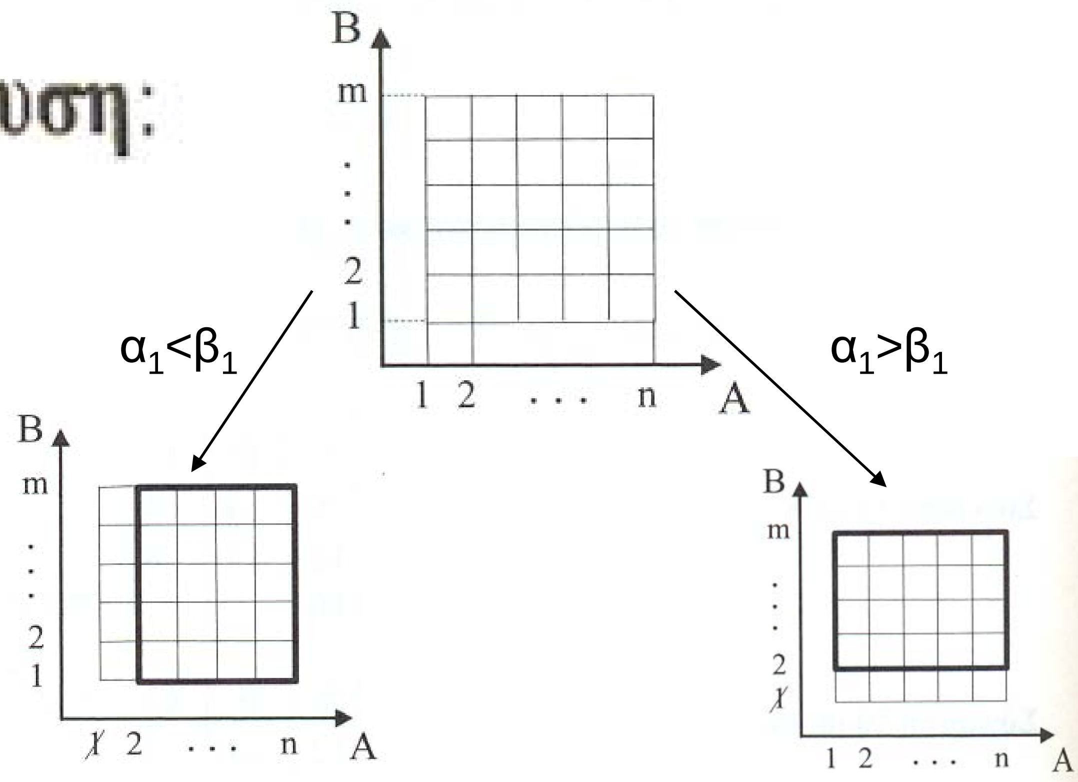
κ.λ.π.

Φυσικά, όταν εξαντληθούν τα στοιχεία μιας από τις A ή B, τα εναπομείναντα στοιχεία της άλλης προσαρτωνται στο τέλος της C χωρίς παραπέρα συγκρίσεις.

Αλγόριθμος MERGE {όπου η A έχει n και η B έχει m στοιχεία}.

$i \leftarrow j \leftarrow k \leftarrow 1$
while $i \le n$ and $j \le m$ do
if $a_i < b_j$ then do $c_k \leftarrow a_i$; $i \leftarrow i+1$ end
else do $c_k \leftarrow b_i$; $j \leftarrow j+1$ end
end
if $i > n$ then move $b_j, \ldots, b_m$ to $c_k, \ldots, c_{n+m}$
else move $a_i, \ldots, a_n$ to $c_k, \ldots, c_{n+m}$

Ανάλυση:



Μετά από κάθε σύγκριση το υπάρχον ορθογώνιο θα περιορίζεται σε κάποιο μικρότερο από το οποίο θα λείπει η αριστερή κάθετη ή η κάτω οριζόντια γραμμή. Η χειρότερη περίπτωση θα είναι όταν φτάσουμε στο ορθογώνιο-σημείο της πάνω δεξιά γωνίας, όπου δηλαδή θα έχουν απομείνει τα στοιχεία α_n και b_m και όπου με μια ακόμα σύγκριση τελειώνουμε.

- Η πορεία από το αρχικό στο τελικό σημείο μπορεί να γίνει με

$$(n-1)+(m-1) = n+m-2$$

μεταβάσεις (συγκρίσεις), στις οποίες προστίθεται και η μια τελική σύγκριση α_n με b_m, δίνοντας συνολικά

$$\text{ΠΧΠ}(n,m)= n+m-1 \text{ συγκρίσεις}$$

Για m=n, ΠΧΠ(n,m)= 2n-1, και ο αλγόριθμος είναι βέλτιστος.

ΘΕΩΡΗΜΑ. Για οποιοδήποτε σωστό αλγόριθμο για συμβολή, όταν οι A και B έχουν από n στοιχεία η κάθε μία, χρειάζονται στην χειρότερη περίπτωση τουλάχιστον 2n- 1 συγκρίσεις.

Ας θεωρήσουμε οποιονδήποτε σωστό αλγόριθμο και ας εξετάσουμε τις εισόδους για τρεις περιπτώσεις:

I : $b1<\alpha1<...<bj<\alpha j<...<bn<\alpha n$

Ij: ίδιο με το I μόνο που τα aj και bj έχουν εναλλαχθεί, δηλαδή aj<bj.

I’j: ίδιο με το I μόνο που τα aj, bj+1 έχουν εναλλαχθεί, δηλαδή

bj < bj+1 < aj...

Ισχυριζόμαστε ότι ο αλγόριθμος (όποιος κι αν είναι) είναι υποχρεωμένος, για το /:

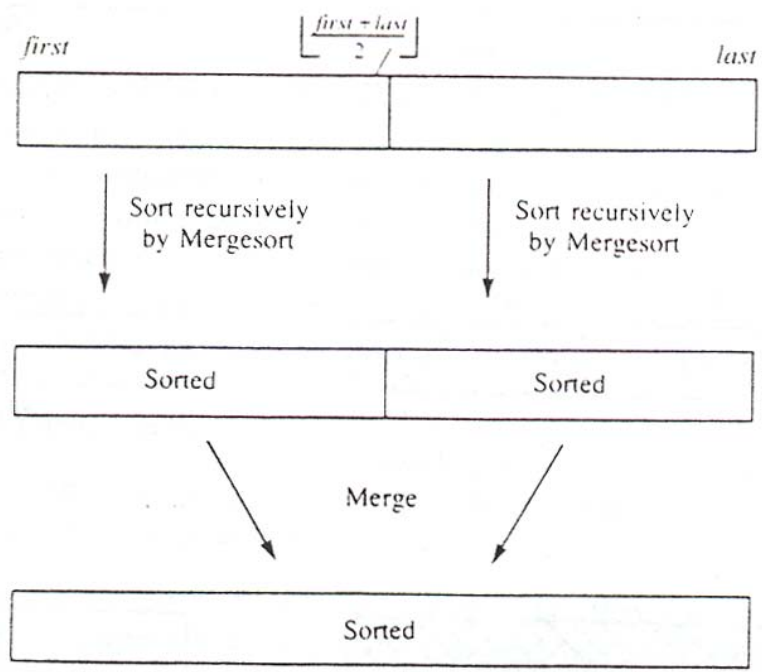
(i) να συγκρίνει a_i με b_i, για όλα τα $i=1,\ldots,n$, Αφού $b_i<a_i$ θα τοποθετηθεί στην C το b_i. Αμέσως μετά

(ii) για όλα τα $i=1,\ldots,(n-1)$ να συγκρίνει a_i με b_{i+1}. Αφού $a_i < b_{i+1}$ θα τοποθετηθεί στην C το a_i.

Αν δεν εφάρμοζε το *(i)* τότε για κάποιο j, $1<j<n$, ο αλγόριθμος δεν θα έκανε την σύγκριση a_j με b_j. Άρα, θα έδινε το ίδιο αποτέλεσμα για το I και για το I_j. Αυτό είναι άτοπο αφού ο αλγόριθμος είναι σωστός. Το ίδιο ισχύει για το *(ii)* και τα I και I'_j.

από *(i)* & *(ii)* έχουμε αντίστοιχα ένα ελάχιστο από n και $n-1$ συγκρίσεις, δηλαδή $2n-1$ συγκρίσεις.

```
procedure Mergesort (first, last : Index);
begin
    if first < last then
        Mergesort (first, ⌊(first+last)/2⌋);
        Mergesort (⌊(first+last)/2⌋+1, last);
        Merge (first, ⌊(first+last)/2⌋, ⌊(first+last)/2⌋+1, last);
    end { if }
end { Mergesort }
```



Ανάλυση

Ας είναι $n=2^p$. Πρώτα έχουμε n λίστες του ενός στοιχείου και τις συμβάλλουμε κατά ζεύγη πληρώνοντας έτσι

$$(1 \text{ σύγκριση/ζεύγος})\ (n/2 \text{ ζεύγη}) = n/2 \text{ συγκρίσεις}$$

Τώρα έχουμε $n/2$ λίστες των δύο στοιχείων και τις συμβάλλουμε κατά ζεύγη. Από το *Θ2.8-1* κάθε ζεύγος απαιτεί $2*2-1=4-1$ συγκρίσεις, έτσι συνολικά απαιτούνται

$$(4-1)*n/4=n(1-1/4) \text{ συγκρίσεις.}$$

Γενικά, θα υπάρχουν $n/2^j$ λίστες των 2^j στοιχείων και θα απαιτούνται

$$(2(2^j)-1)n/2^{j+1} = n(1-1/2^{j+1}) \text{συγκρίσεις,}$$

για $j=0,\ldots,(p-1)$. Αθροίζοντας ως προς j ΠΧΠ(n)=nlogn-n+1 συγκρίσεις

Εξωτερική διάταξη

- Ο αριθμός n των στοιχείων είναι πολύ μεγάλος: δεν χωρούν στην μνήμη RAM του υπολογιστή
- m: η χωρητικότητα της RAM (n>>m)
- Μεγαλύτερη βαρύτητα δίνεται στο χρόνο μεταφοράς δεδομένων μεταξύ RAM και δίσκου (ή ταινίας)
- Ο χρόνος αυτός αρκετά μεγαλύτερος από το χρόνο συγκρίσεων στη RAM
- Προτιμότερες οι ακολουθιακές διαδικασίες
- π.χ. η δυαδική αναζήτηση απαιτεί παλινδρομήσεις στην συσκευή δευτερεύουσας αποθήκευσης

Algorithm 2.13 External Sort with Four Files (Outline)

{Phase I: Run construction and distribution}
while there are more records in T_0 **do**
  1. Read in m records (perhaps fewer the last time).
  2. Sort them using an internal sorting algorithm.
  3. If the previous run went in T_2, add this one to T_3; else to T_2.
end;

Rewind tapes, or reset disk files.

{Phase II: Merging the runs}
$i_1 := 2$; $i_2 := 3$ {indexes of input files}
$j_1 := 0$; $j_2 := 1$ {indexes of output files}
while there is more than one run **do**
  1. Merge the first run in T_{i_1} with the first run in T_{i_2} and put the resulting run in T_{j_1}.
  2. Merge the next run from T_{i_1} and T_{i_2} and put the result in T_{j_2}.
  3. Repeat steps 1 and 2 (putting the output alternately in T_{j_1} and T_{j_2}) until the end of T_{i_1} and T_{i_2}.
  4. Rewind tapes, or reset files. Add 2 (modulo 4) to i_1, i_2, j_1, and j_2 to reverse the roles of input and output files.
end

T_0 49 12 23 51 64 71 15 1 33 61 92 2 8 45 6 81 11 5 19 3 13 ■■

Μετά από την 1η Φάση:

T_2 12 23 49 51 ■ 2 33 61 92 ■ 3 5 11 19 ■■

T_3 1 15 64 71 ■ 6 8 45 81 ■ 13 ■■

2η Φάση, 1ο πέρασμα:

T_0 1 12 15 23 49 51 64 71 ■ 3 5 11 13 19 ■■

T_1 2 6 8 33 45 61 81 92 ■■

2η Φάση, 2ο πέρασμα:

T_2 1 2 6 8 12 15 23 33 45 49 51 61 64 71 81 92 ■■

T_3 3 5 11 13 19 ■■

2η Φάση, 3ο πέρασμα:

T_0 1 2 3 5 6 8 11 12 13 15 19 23 33 45 49 51 61 64 71 81 92 ■■

Ανάλυση

- Βασικές πράξεις: συγκρίσεις, επανατυλίξεις, πλήρη περάσματα από τα δεδομένα
- Με τη λήξη της πρώτης φάσης έχουμε r=n/m διατεταγμένες ομάδες στοιχείων
- Για τη πρώτη φάση το κόστος έχει ως εξής:
  - Ένα πλήρες πέρασμα από τα δεδομένα
  - Μία επανατύλιξη
  - σ συγκρίσεις όπου σ=n/m* S(m) όπου S(k) ο χρόνος για τη διάταξη κ στοιχείων π.χ. S(k)=2klogk αν χρησιμοποιούμε Heapsort
  - Στη δεύτερη φάση έχουμε $\lceil \log r \rceil$ περάσματα/επανατυλίξεις:
    - Αρχικά έχουμε r ομάδες των m στοιχείων
    - Μετά από το πρώτο πέρασμα: r/2 ομάδες των 2m στοιχείων
    - Μετά από το δεύτερο πέρασμα: r/4 ομάδες των 4m στοιχείων κοκ
  - Συνολικά $\lceil \log r \rceil + 1$ επανατυλίξεις/περάσματα

- Συγκρίσεις
- 1η φάση: 2nlogm συγκρίσεις
- 2η φάση:
  - κατά το πέρασμα i: $r/2^i$ συμβολές λιστών μεγέθους $2^{i-1}m$. Άρα συνολικά $r/2^i\,(2^i m-1) = n - n/m\;2^{-i}$
  - Άρα για τα $\lceil \log r \rceil$ περάσματα έχουμε

$$\sum_{i=1}^{\lceil \log r \rceil}(n - n/m \cdot 2^{-i}) = n\lceil \log(n/m)\rceil - (n/m)\sum_{i=1}^{\lceil \log n/m \rceil} 2^{-i}$$

συγκρίσεις. Ο αριθμός αυτός γράφεται και στη μορφή

$$n\lceil \log(n/m)\rceil - (n/m) + \varepsilon\,,$$

όπου $0<\varepsilon<1$ (για ποιές τιμές του n/m είναι $(1/2)<\varepsilon$;). Έτσι, τελικά ο συνολικός αριθμός των συγκρίσεων, όταν π.χ. χρησιμοποιείται ο *Heapsort* στην *1η* φάση, είναι:

$$2n\log m + n\lceil \log(n/m)\rceil - n/m + \varepsilon = O(n\log n).$$