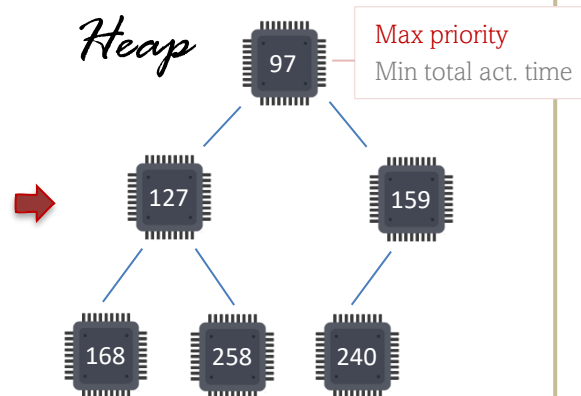
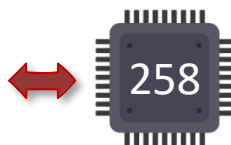
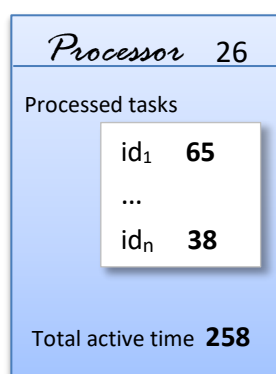




# DATA Structures

## ΜΕΡΟΣ Α-Β

Η υλοποίηση της ουράς προτεραιότητας (με χρήση **Generics**) βρίσκεται στο αρχείο MaxPQ.java και έχει τη μορφή **μεγιστοστρεφούς σωρού** με χρήση μονοδιάστατου πίνακα. Ο σωρός περιέχει επεξεργαστές (αντικείμενα της κλάσης Processor) και η έννοια της μεγιστοστρέφειας αναφέρεται στην υψηλότερη προτεραιότητα ανάληψης διεργασίας. Την μεγαλύτερη προτεραιότητα έχει ο επεξεργαστής με το μικρότερο συνολικό active time (totalActiveTime). Η σύγκριση για τον καθορισμό της προτεραιότητας βασίζεται στην υλοποίηση της μεθόδου compareTo (Processor.java), η οποία αναλαμβάνει τη σύγκριση σύμφωνα με τα συνολικά active time. Ο σωρός σε κάθε στιγμιότυπο του προγράμματος έχει στην κορυφή του τον επεξεργαστή με το μικρότερο συνολικό active time. Επομένως, η μέθοδος getMax() επιστρέφει τον επεξεργαστή της κορυφής του σωρού, δηλαδή αυτόν με τη μέγιστη κάθε φορά προτεραιότητα.



Η μέθοδος **greedy()** (Greedy.java) δέχεται ως όρισμα το path του αρχείου των διεργασιών προς ανάγνωση (από τη main\*) και καλεί τις μεθόδους: input(), greedyCore() και output() με τη σειρά.

Η `input()` αναλαμβάνει την ανάγνωση του εκάστοτε αρχείου και τον έλεγχο ότι ο αριθμός διεργασιών που περιλαμβάνονται είναι σύμφωνος με τον πληθικό αριθμό που είναι δηλωμένος στη 2<sup>η</sup> γραμμή του αρχείου. Αν υπάρξει ασυμφωνία επιστρέφεται null και ενημερώνεται κατάλληλα ο χρήστης. Αλλιώς, επιστρέφει έναν πίνακα διεργασιών (`Task[]`) και ενημερώνει τον αριθμό των επεξεργαστών, σύμφωνα με τον αριθμό στην 1<sup>η</sup> γραμμή του αρχείου.

Στη συνέχεια η μέθοδος `greedy()` κατασκευάζει την ουρά προτεραιότητας με βάση το πλήθος των επεξεργαστών του αρχείου και τους εισάγει σε αυτή.

Η `greedyCore()` δέχεται ως ορίσματα την ουρά των επεξεργαστών και τον πίνακα των διεργασιών και υλοποιεί τον βασικό αλγόριθμο:

➡ Επαναληπτικά για κάθε διεργασία:



1. Με χρήση της `getMax()` επιστρέφεται ο επεξεργαστής με τη μέγιστη προτεραιότητα,
2. Ανατίθεται σε αυτόν η τρέχουσα διεργασία, μέσω εισαγωγής της στη λίστα των διεργασιών του συγκεκριμένου επεξεργαστή,
3. Ο επεξεργαστής επανεισάγεται στο σωρό για μελλοντική χρήση και μέσω της διαδικασίας ανάδυσης καταλαμβάνει την κατάλληλη θέση σε αυτόν, με βάση τη νέα του προτεραιότητα (νέο συνολικό active time).

Η `output()` δέχεται ως ορίσματα την ουρά και τον αριθμό των διεργασιών του αρχείου και αν ο αριθμός αυτός είναι μικρότερος του 50 εμφανίζει τους επεξεργαστές σε αύξουσα σειρά σύμφωνα με το συνολικό active time τους και τις διεργασίες που διεκπεραίωσε ο κάθε ένας. Σε κάθε περίπτωση στο τέλος εμφανίζεται και επιστρέφεται το `makespan` (μέγιστο συνολικό active time).

## ΜΕΡΟΣ Γ

Αλγόριθμος: **HeapSort**

Task 1393	Task 4251	Task 1002	Task 3980	Task 7352	Task 3001
time: 78	time: 71	time: 58	time: 53	time: 44	time: 30

Υλοποιήθηκε στο αρχείο `Sort.java` με χρήση πίνακα. Η μέθοδος `heapsort()` δέχεται ως όρισμα έναν πίνακα διεργασιών (`Task[]`). Για τον αλγόριθμο οι διεργασίες του πίνακα πρέπει να ταξινομούνται σε φθίνουσα σειρά και για να επιτευχθεί αυτό, ο πίνακας παίρνει τη μορφή ελαχιστοστρεφούς σωρού. Στο ίδιο αρχείο υπάρχουν οι διαδικασίες `sink()`, `greater()` και `exch()` που βοηθούν στη φθίνουσα ταξινόμηση, ώστε στο αριστερό άκρο του πίνακα (κορυφή του σωρού) να βρίσκεται η διεργασία με τη μεγαλύτερη διάρκεια.

## ΜΕΡΟΣ Δ

Στο μέρος αυτό παράγονται αρχεία με τυχαία δεδομένα διεργασιών προς επεξεργασία από τον αλγόριθμο Greedy και υπολογίζονται οι μέσοι όροι των makespan, για κάθε περίπτωση αριθμού διεργασιών και χρήσης ή μη της Heapsort.

Τα αποτελέσματα που εξάγονται αφορούν 3 διαφορετικές τιμές για το πλήθος των διεργασιών. Οι τιμές αυτές καθορίζονται στον πίνακα N\_values. Πχ αρχικά το πρόγραμμα τρέχει για:

```
int[ ] N_values = {100, 250, 500};
```

Για κάθε τιμή του πίνακα N\_values παράγονται από τη μέθοδο writer() 10 αρχεία με τόσες διεργασίες όσες και η κάθε τιμή. Τα ονόματα των αρχείων έχουν τη μορφή:

“N\_100\_i.txt”, “N\_250\_i.txt”, “N\_500\_i.txt” , i = 1,2,..10

Στη συνέχεια καλείται 2 φορές η μέθοδος calcAverage() για κάθε τιμή του πίνακα N\_values, η οποία αναλαμβάνει τον υπολογισμό του μέσου όρου των makespan που προκύπτει για κάθε τιμή του N\_values από τα 10 αρχεία.

Π.χ. 1 μέσος όρος για τα αρχεία “N\_100\_i.txt”

1 μέσος όρος για τα αρχεία “N\_250\_i.txt”

1 μέσος όρος για τα αρχεία “N\_500\_i.txt”

i = 1,2,..10

Αυτό επιτυγχάνεται με επαναληπτικές κλήσεις της static μεθόδου greedy() του αρχείου Greedy.java, μέσα στο σώμα της calcAverage().

Για κάθε N (100, 250, 500) η 1<sup>η</sup> κλήση της calcAverage() αφορά την επεξεργασία των διεργασιών με τη σειρά που καταφθάνουν (χωρίς ταξινόμηση) και η 2<sup>η</sup> κλήση αφορά την επεξεργασία των ταξινομημένων από τη heapsort() μέθοδο διεργασιών. Για να επιτευχθεί αυτός ο διαχωρισμός της χρήσης ή μη της HeapSort, χρησιμοποιείται η static μέθοδος setUseHeapSort() μέσα στη greedy() με όρισμα false και true αντίστοιχα.

Οπότε δημιουργούνται 2 πίνακες που περιλαμβάνουν τους μέσους όρους για κάθε N που έχει χρησιμοποιηθεί. Ο 1<sup>ος</sup> πίνακας έχει τα αποτελέσματα της αταξινομήτης κλήσης και ο 2<sup>ος</sup> της ταξινομημένης.

N\_values = { 100, 250, 500 }

429.2	704.0	940.2
-------	-------	-------

Unsorted

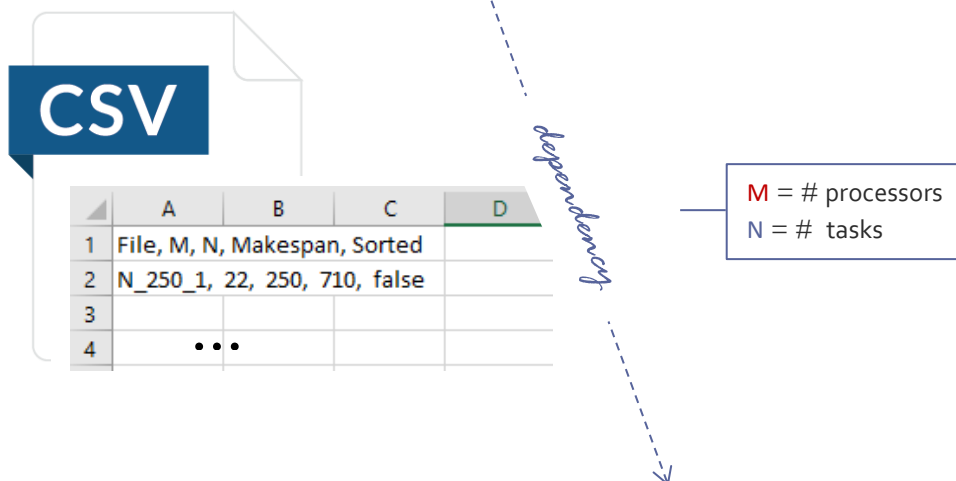
{ 100, 250, 500 }

392.2	667.0	901.1
-------	-------	-------

Sorted

Τυπώνεται για κάθε ξεχωριστό αρχείο το makespan (από την output() στο Greedy.java) και στη συνέχεια τα δεδομένα των πινάκων για ενημέρωση του χρήστη.

Τέλος, δημιουργείται ένα αρχείο **data.csv** με τα αναλυτικά δεδομένα που έχει επεξεργαστεί το πρόγραμμα για κάθε αρχείο σε μορφή:

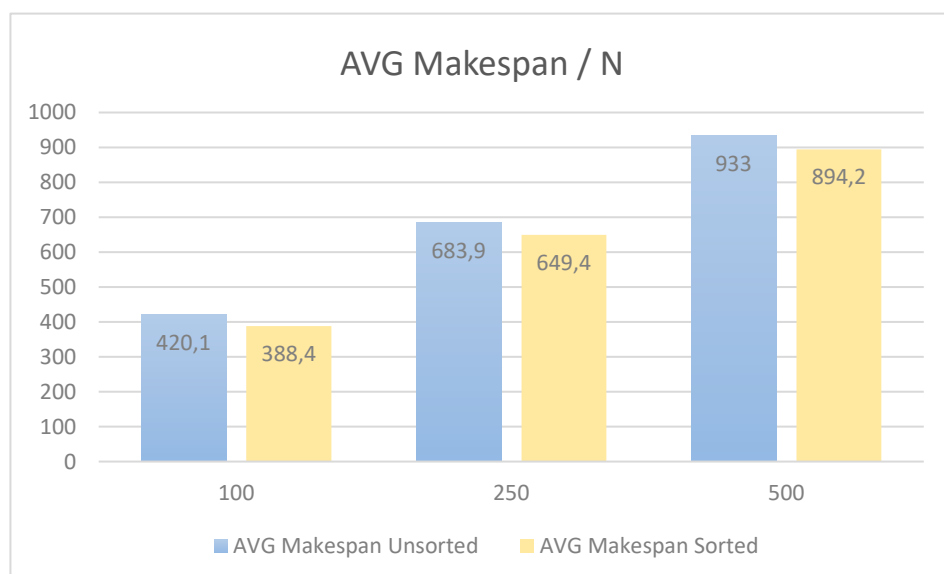


	A	B	C	D
1	File, M, N, Makespan, Sorted			
2	N_250_1, 22, 250, 710, false			
3				
4	...			

**M** = # processors  
**N** = # tasks

Το αρχείο αυτό τροφοδοτεί ένα λογιστικό φύλλο **book.xlsx**, το οποίο κατά το άνοιγμα ενημερώνεται με τα τελευταία δεδομένα που έχουν παραχθεί από το πρόγραμμα. Εσωτερικά υπολογίζεται ο μέσος όρος κάθε τιμής του N για ταξινομημένες και μη διεργασίες ( $2 \times 3 = 6$  συνολικά μέσοι όροι) και ενημερώνεται το διάγραμμα.

## Ανάλυση Δεδομένων

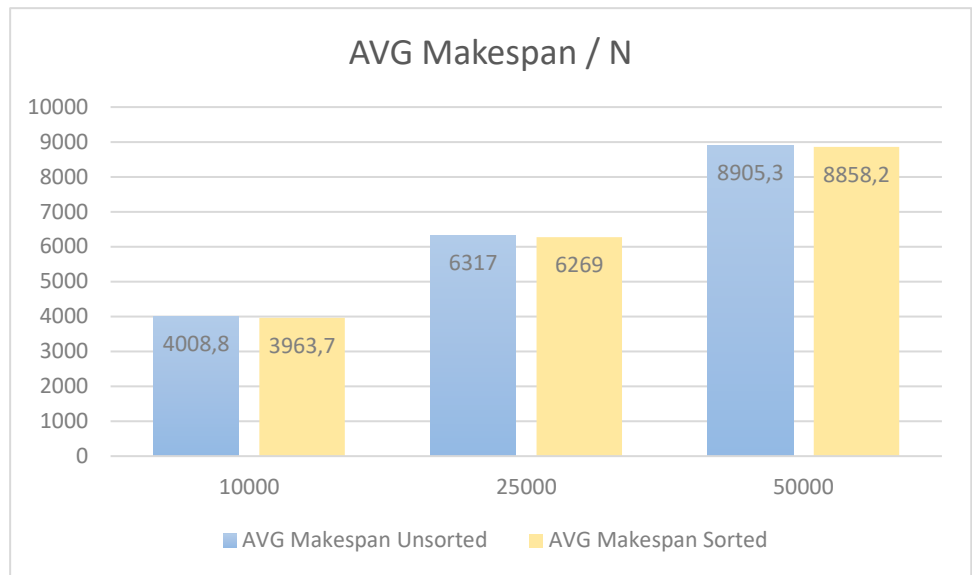


1. Αρχεία 100 – 250 – 500 διεργασιών

Σύμφωνα με το διάγραμμα, παρατηρούμε ότι και για τις 3 τιμές διεργασιών, η sorted κλήση του προγράμματος οδηγεί τελικά σε μικρότερο makespan κατά 40 περίπου μονάδες χρόνου.

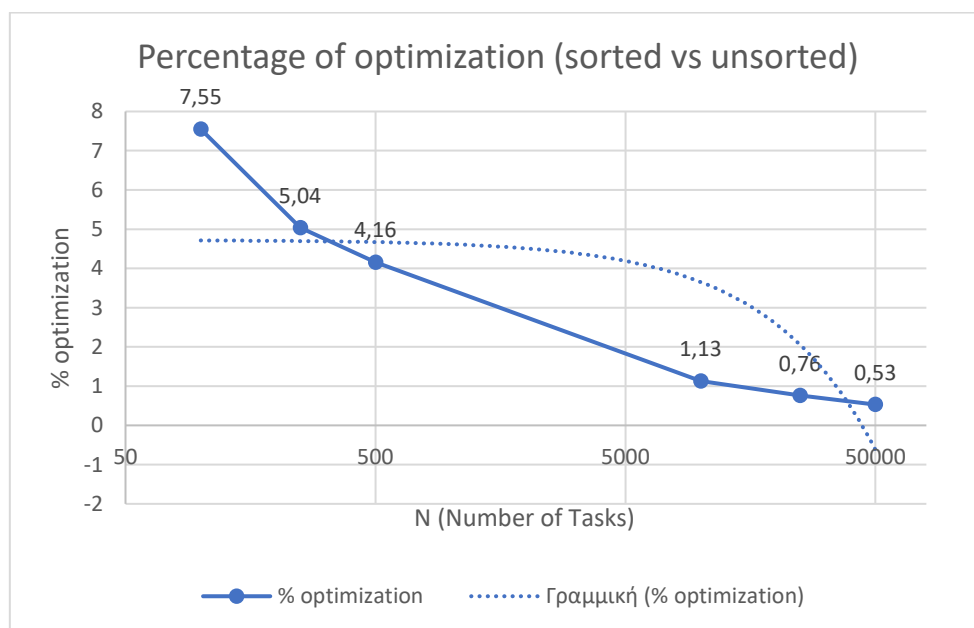
## 2. Αρχεία 10000-25000-50000 διεργασιών

Και με αισθητά μεγαλύτερες τιμές διεργασιών παρατηρούμε ότι επιβεβαιώνεται το συμπέρασμα που αναφέρθηκε και παραπάνω

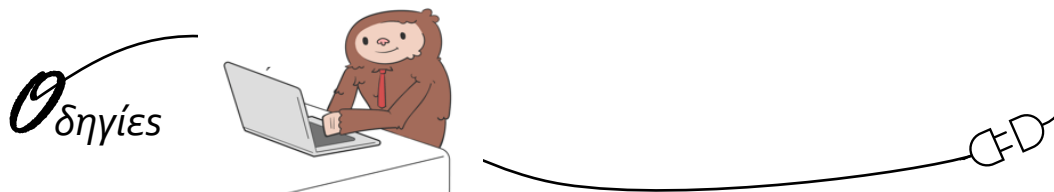


## Overall

Στο κάτω διάγραμμα φαίνεται η πτωτική τάση (διάγραμμα τάσης) της βελτίωσης που επιφέρει η ταξινόμηση των διεργασιών στο makespan. Όσο αυξάνεται ο αριθμός των διεργασιών μειώνεται το ποσοστό της βελτίωσης, επομένως η ταξινόμηση δείχνει πιο αποδοτική για σχετικά μικρό αριθμό διεργασιών. Έτσι στους αρκετά μεγαλύτερους αριθμούς διεργασιών (10-25-50 χιλιάδες) το ποσοστό βελτίωσης που επιφέρει η ταξινόμηση είναι αισθητά μικρότερο.



Τέλος παρατηρούμε τον ρυθμό της βελτίωσης να φθίνει και να τείνει ασυμπτωτικά στο 0 όσο αυξάνεται ο αριθμός των διεργασιών.



Σημ.: Κατά το compilation παράγεται warning από τον compiler λόγω **generic** υλοποίησης της *MaxPQ.java*.

### Για το μέρος Β (Greedy.java)

\*Η main του Greedy.java παίρνει ως όρισμα το path ενός αρχείου με διεργασίες (args[0]) μέσω command line ως εξής:

```
Terminal — -zsh — 80x24

... > java Greedy "C:\Users\[USER] \Desktop\tasks.txt"
```

ή και μέσα από τις ρυθμίσεις εκκίνησης ενός IDE (run configurations).

### Για το μέρος Δ (Comparisons.java)

Η main του Comparisons.java, χρησιμοποιεί τον πίνακα N\_values για να δημιουργήσει τα τυχαία αρχεία διεργασιών για οποιοσδήποτε 3 τιμές επιθυμεί ο χρήστης, να τα επεξεργαστεί και να υπολογίσει τους μέσους όρους, εξάγοντας τα δεδομένα στο data.csv αρχείο. Το book.xlsx κατά το άνοιγμά του ενημερώνεται με τα νέα δεδομένα.

Για να γίνει αυτό σε νέο υπολογιστή, πρέπει να ενημερωθεί χειροκίνητα το absolute path του data.csv στο υπολογιστικό φύλλο book.xlsx ώστε να δημιουργηθεί η σύνδεση και τα διαγράμματα θα ανανεωθούν αυτόματα, χωρίς κάποια άλλη ενέργεια από τον χρήστη.

Στο directory "data/..." υπάρχουν και τα αρχεία .txt, .csv, .xlsx για τα δεδομένα που αναλύθηκαν παραπάνω. (Τα συγκεκριμένα αρχεία κρατάνε σταθερές τις τιμές τους.)

