Α) Σχετικά με την χρήση της ουράς προτεραιότητας στον Αλγόριθμο 1 (κτλ):

Η ουρά προτεραιότητας (εφεξής pq) στο αρχείο GreedyAlgorithm, χρησιμοποιείται προκειμένου να επιλέγεται κάθε φορά ο επεξεργαστής που έχει προς το παρόν τις διεργασίες με τον μικρότερο χρόνο εκτέλεσης - active time (αθροιστικά).

Συγκεκριμένα, η pq δημιουργείται (γραμμή 19), και εισάγουμε σε αυτή αντικείμενα τύπου processor (γραμμή 25). Κάθε ένα από αυτά τα αντικείμενα, έχει εκτελέσει ακριβώς μία διεργασία. Έπειτα, στη γραμμή 28 επιλέγουμε τον επεξεργαστή με το μικρότερο active time και τον αφαιρούμε από την pq, προκειμένου να του δώσουμε μια ακόμα για να επεξεργαστεί και έπειτα ξαναβάζουμε στην pq τον συγκεκριμένο επεξεργαστή με το καινούργιο του active time (βλέπετε γραμμή 33). Η διαδικασία σταματά όταν δεν υπάρχουν άλλες διεργασίες προς επεξεργασία.

Τέλος, βάζουμε όλους τους επεξεργαστές από την pq σε ένα πίνακα ταξινομημένους ανάλογα με το active time του καθενός, από το μικρότερο στο μεγαλύτερο, εκμεταλλευόμενοι το γεγονός ότι με την getMax() λαμβάνουμε κάθε φορά τον επεξεργαστή με το μικρότερο active time.

Για να παίρνουμε κάθε φορά τον επεξεργαστή με το μικρότερο active time, 'πειράξαμε' τον comparator που συγκρίνει δύο επεξεργαστές (αρχείο ProcessorComparator). Συγκεκριμένα, αντιστρέψαμε το πότε επιστρέφεται 1 και πότε -1 στη σύγκριση των δύο αντικειμένων τύπου processor.

Σχετικά με άλλες πτυχές της χρήσης του αλγορίθμου 1 (αρχείο Greedy.java):

Η είσοδος αποθηκεύεται σε ένα πίνακα ακεραίων (βλέπετε γραμμή 7 αρχείου Greedy.java), και προέρχεται από την εκτέλεση της μεθόδου readTxt(name_of_file.txt).

[Η μέθοδος readTxt προέρχεται από την κλάση TxtReader,και δέχεται ως είσοδο το όνομα του αρχείου που θέλουμε να διαβάσουμε, κάνοντας παράλληλα και κατάλληλους ελέγχους προκειμένου να διασφαλιστεί ότι το αρχείο είναι στη σωστή μορφή, ώστε να μην υπάρξουν προβλήματα κατόπιν.]

Έπειτα τρέχουμε τον αλγόριθμο 1, και λαμβάνουμε το makespan του (βλέπετε γραμμή 10 αρχείου Greedy.java), για τα συγκεκριμένα δεδομένα. Παράλληλα, η runAlgorithm1 τυπώνει στην κονσόλα και τα χαρακτηριστικά (id, load) κάθε επεξεργαστή, υπό την προϋπόθεση ότι οι εκτελεσθείσες εργασίες δεν υπερβαίνουν τις 99 (<100).

Το makespan δεν τυπώνεται και αυτό, αλλά επιστρέφεται, γιατί η υλοποίηση αυτή συμφέρει στην χρήση της runAlgorithm1 στο τμήμα Δ της εργασίας.

Β) Σχετικά με τον επιλεχθέντα αλγόριθμο ταξινόμησης:

Ο Αλγόριθμος που επιλέχθηκε ήταν αυτός της ταξινόμησης HeapSort, προκειμένου να αξιοποιηθεί η δομή της pq, όπως αναφέρεται και στην εκφώνηση της εργασίας.

Η υλοποίηση της HeapSort έγινε με πίνακα, προκειμένου να συμβαδίζει με την δομή της pq. Η υλοποίηση της pq στηρίχθηκε στον αντίστοιχο αλγόριθμο που δίνεται στις διαφάνειες του μαθήματος.

C) Συμπεράσματα από την πειραματική αξιολόγηση των δύο αλγορίθμων:

Η εκτέλεση των δύο αλγορίθμων με είσοδο τα 30 αρχεία που περιέχουν τυχαίους χρόνους για την επεξεργασία κάθε διεργασίας, κατηγοριοποιημένα σε τρεις ομάδες ανάλογα με το πλήθος των διεργασιών (00-09 για N=100, 10-19 για N = 500 και 20-29 για N = 1000) μας ώθησε στο να εξάγουμε το εξής συμπέρασμα:

Ο Αλγόριθμος Greedy decreasing είναι κατά μέσο όρο περίπου 5% ταχύτερος από τον αλγόριθμο Greedy. [Αναλυτικότερα, η διαφορά είναι μεγαλύτερη όσο πιο μικρό είναι το πλήθος των διεργασιών (κυμαίνεται από 2 μέχρι 8%), αλλά ο μέσος όρος της διαφοράς είναι μεταξύ του 4 και 6%.]

Το συμπέρασμα αυτό είναι αναμενόμενο, καθόσον με τον Greedy decreasing πριν αρχίσει η κατανομή των διαφόρων διεργασιών σε κάθε επεξεργαστή, οι διεργασίες ταξινομούνται από την μεγαλύτερη στη μικρότερη. Αυτό έχει ως αποτέλεσμα κατά την κατανομή των διεργασιών στους επεξεργαστές, να επιλέγονται πρώτα οι πιο «βαριές» επεξεργασίες. Έτσι όταν έχουν ήδη επεξεργαστεί επεξεργασίες ίσες ή περισσότερες από τον αριθμό των επεξεργαστών (και άρα οι επεξεργαστές έχουν διαφορετικά active time), δεν υπάρχει περίπτωση να ανατεθεί μια πιο «βαριά» διεργασία από τις προηγούμενες, και επομένως να υπάρξει κάποια σημαντική αύξηση του active time κάποιου επεξεργαστή έναντι των υπολοίπων.

```
C:\Users\a\Desktop\aft\OPA\Domes Dedomenon\Exercise 2>javac *.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\a\Desktop\aft\OPA\Domes Dedomenon\Exercise 2>java Comparisons
The average makespan for 100 procedures using greedy algorithm is: 11081
The average makespan for 100 procedures using greedy decreasing algorithm is: 10220
Greedy decreasing algorithm is faster by 7.770056854074542 %, for 100 procedures

The average makespan for 500 procedures using greedy algorithm is: 24250
The average makespan for 500 procedures using greedy decreasing algorithm is: 23213
Greedy decreasing algorithm is faster by 4.276288659793814 %, for 500 procedures

The average makespan for 1000 procedures using greedy algorithm is: 33641
The average makespan for 1000 procedures using greedy decreasing algorithm is: 32613
Greedy decreasing algorithm is faster by 3.055795012038881 %, for 1000 procedures

CONCLUSION: greedy decreasing algorithm is faster by 5.034046841969079 %.
```

Βέβαια, δεν αναμενόταν κάποια μεγαλύτερη διαφορά, γιατί ούτως ή άλλως ο αλγόριθμος Greedy κάνει μια έξυπνη επιλογή επεξεργαστή, αφού επιλέγεται κάθε φορά αυτός που έχει το μικρότερο active time. Οπότε μπορούμε να σχολιάσουμε πως μια επιτάχυνση της τάξης του 5% στην εκτέλεση όλων των διεργασιών είναι αναμενόμενη, αφού δεν λαμβάνει χώρα κάποια σημαντική αλλαγή στον τρόπο που διαμοιράζονται οι εργασίες (πχ αύξηση αριθμού επεξεργαστών, αναβάθμισή τους κλπ.)