Τσώλας Λεωνίδας – 03112422 – ΣΗΜΜΥ ΕΜΠ

Αλγόριθμοι και Πολυπλοκότητα
1η σειρά ασκήσεων

Άσκηση 1

1.α.

$$\sum_{k=1}^{n} k2^{-k} < \log(\binom{n}{\log n}) < (\log n)^2/\log\log n < \log^4 n <$$

$$\log(n!)/(\log n)^3 < \log(\binom{2n}{n}) = n2^{2^{2^{100}}} < n^{3/2} <$$

$$n^3/(\log n)^8 < \binom{n}{6} < (\log_2 n)^{\log_2 n} < 3^{(\log_2 n)^3} <$$

$$\sum_{k=1}^{n} k2^k = n\sum_{k=0}^{n}\binom{n}{k} < (\sqrt{n})! < n^{\log_2(n!)}$$

Η συγκεκριμένη διάταξη προκύπτει ως εξής:

$\sum_{k=1}^{n} k2^{-k} = \Theta(1)$, αφού καθώς $n \to \infty$, το κλάσμα $\frac{k}{2^k} \to 0$.

$\log(\binom{n}{\log n}) = O(loglogn)$

$(\log n)^2/\log\log n = O(\frac{(\log^2)n}{loglogn})$

$\log^4 n = O(\log^4 n)$

$\log(n!)/(\log n)^3 = \Theta(\frac{n}{\log^2 n})$

$\log\left(\binom{2n}{n}\right) = \Theta(\log(2^n)) = \Theta(n\log 2) = \Theta(n)$

$n2^{2^{2^{100}}} = \Theta(n)$

$n^{3/2} = \Theta(n^{\frac{3}{2}})$

$n^3/(\log n)^8 = \Theta(\frac{n^3}{poly\log(n)})$

$\binom{n}{6} = \Theta(n^6)$

$(\log_2 n)^{\log_2 n} = \Theta(logn^{logn})$

$3^{(\log_2 n)^3} = \Theta(3^{poly(\log n)})$

$\sum_{k=1}^{n} k2^k = \Theta(n2^n)$

$n\sum_{k=0}^{n}\binom{n}{k} = \Theta(n2^n)$

$(\sqrt{n})! = \Theta((n^{\frac{1}{2}})!)$

$n^{\log_2(n!)} = \Theta(n^{nlogn})$

1.β.

1. $T(n)=2T(\frac{n}{3})+nlogn$

Είμαστε στην τρίτη περίπτωση του Master Theorem, άρα $T(n)=\Theta(nlogn)$

2. $T(n)=3T(\frac{n}{3})+nlogn$

Είμαστε στη δεύτερη περίπτωση του Master Theorem, άρα $T(n)=\Theta(nlogn)$

3. $T(n)=4T(\frac{n}{3})+nlogn$

Είμαστε στην πρώτη περίπτωση του Master Theorem, έχουμε $\log_3 4 = 1.27 > 1$, άρα $T(n)=\Theta(n\log_3 4)$.

4. $T(n)=T(\frac{n}{2})+T(\frac{n}{3})+n$

Επειδή $\frac{n}{2}+\frac{n}{3}<n$, σχηματίζεται ένα δέντρο αναδρομής το οποίο – αν κοιτάξουμε τα πλήρη επίπεδά του - έχει άθροισμα σε κάθε επίπεδο ίσο με της φθίνουσας γεωμετρικής προόδου

$$T(n)=n+\frac{5}{6}n+\frac{25}{36}n+\ldots$$

Μπορούμε να πάρουμε άνω όριο αναπτύσσοντας το δέντρο προς το άπειρο και κάτω όριο μετρώντας τους κόμβους στα πλήρη επίπεδα. Με τον ένα ή τον άλλο τρόπο, η γεωμετρική σειρά καθορίζεται από το μεγαλύτερο όρο, άρα $T(n)=\Theta(n)$.

5. $T(n)=T(\frac{n}{2})+T(\frac{n}{3})+T(\frac{n}{6})+n$

Σε αυτή την περίπτωση. Αν σχηματίσουμε το δέντρο αναδρομής, αυτό θα έχει σε κάθε επίπεδο που είναι πλήρες, άθροισμα σταθερό και ίσο με n. Επίσης, κάθε φύλλο στο αναδρομικό δέντρο έχει βάθος d,όπου:

$$\log(_6 n)<d<\log(_2 n)$$

Για να υπολογίσουμε το άνω όριο, υπερεκτιμούμε το T(n) και αναπτύσσουμε το δέντρο προς τα κάτω ως το βαθύτερό του φύλλο. Ομοίως, για να υπολογίσουμε το κάτω όριο, υπερεκτιμούμε το T(n) και υπολογίζουμε μόνο τους κόμβους του δέντρου μέχρι το επίπεδο του πιο "ρηχού" φύλλου. Οι παρατηρήσεις αυτές δίνουν τα άνω και κάτω όρια για το T(n) και ισχύει

$$\log(_6 n) < T(n) < \log(_2 n)$$

Επειδή αυτά τα όρια διαφέρουν κατά μία σταθερά έχουμε ότι $T(n)=\Theta(nlogn)$

6. $T(n)=T(n-1)+logn$

$$\Leftrightarrow T(n)=T(n-2)+\log(n-1)+logn$$

$$\Leftrightarrow T(n)=T(n-3)+\log(n-2)+\log(n-1)+logn$$

$$\Leftrightarrow T(n)=T(1)+\log 2+...+\log(n-1)+logn$$

$$\Leftrightarrow T(n)=T(1)+\log(n!)$$

$$\Leftrightarrow T(n)=\Theta(nlogn)$$

7. $T(n)=T(n^{\frac{5}{6}})+\Theta(logn)$

Επειδή $\frac{5}{6}\leqslant 1$ έχουμε μία παράσταση που εκθετικά τείνει στο μηδέν.

$$T(n)=T(n^{\frac{5}{6}})+\Theta(logn)$$

$$\Leftrightarrow T(n)=T((n^{\frac{5}{6}})^2)+\Theta(logn^{\frac{5}{6}})+\Theta(logn)$$

$$\Leftrightarrow T(n)=T((n^{\frac{5}{6}})^n)+(\frac{5}{6})^n*\Theta(logn)+(\frac{5}{6})^{(n-1)}*\Theta(logn)+...+\Theta(logn)$$

$\Leftrightarrow T(n)=\Theta(1)+6*\Theta(logn)$ (άθροισμα φθίνουσας γεωμετρικής προόδου με λόγο $\frac{5}{6}$)

$\Leftrightarrow T(n)=\Theta(logn)$

8. $T(n)=T(\frac{n}{4})+\sqrt{n}$

Από Master Theorem έχουμε να επιλέξουμε το μεγαλύτερο (πολυωνυμικά) όρο μεταξύ των $n^{(\log 1)}=n^0=1$ και $\sqrt{n}$.

Προφανώς μεγαλύτερος είναι ο δεύτερος, άρα είμαστε στην τρίτη περίπτωση του Master Theorem και ισχύει $T(n)=\Theta(\sqrt{n})$

Άσκηση 2

2.α.1.

Σύμφωνα με τις απαιτήσεις θέλουμε – με είσοδο πίνακα n στοιχείων – να λάβουμε $\frac{n}{k}$ "εξωτερικά" ταξινομημένους πίνακες k στοιχείων έκαστος. Ο όρος "εξωτερικά" ορίζεται πλήρως από τις ακόλουθες 2 (δύο) συνθήκες:

i) Το μικρότερο στοιχείο κάθε υποπίνακα να είναι μεγαλύτερο από τα μεγαλύτερα στοιχεία των προηγούμενων υποπινάκων.
ii) Το μεγαλύτερο στοιχείο κάθε υποπίνακα να είναι μικρότερο από τα μικρότερα στοιχεία των επόμενων υποπίνακα.

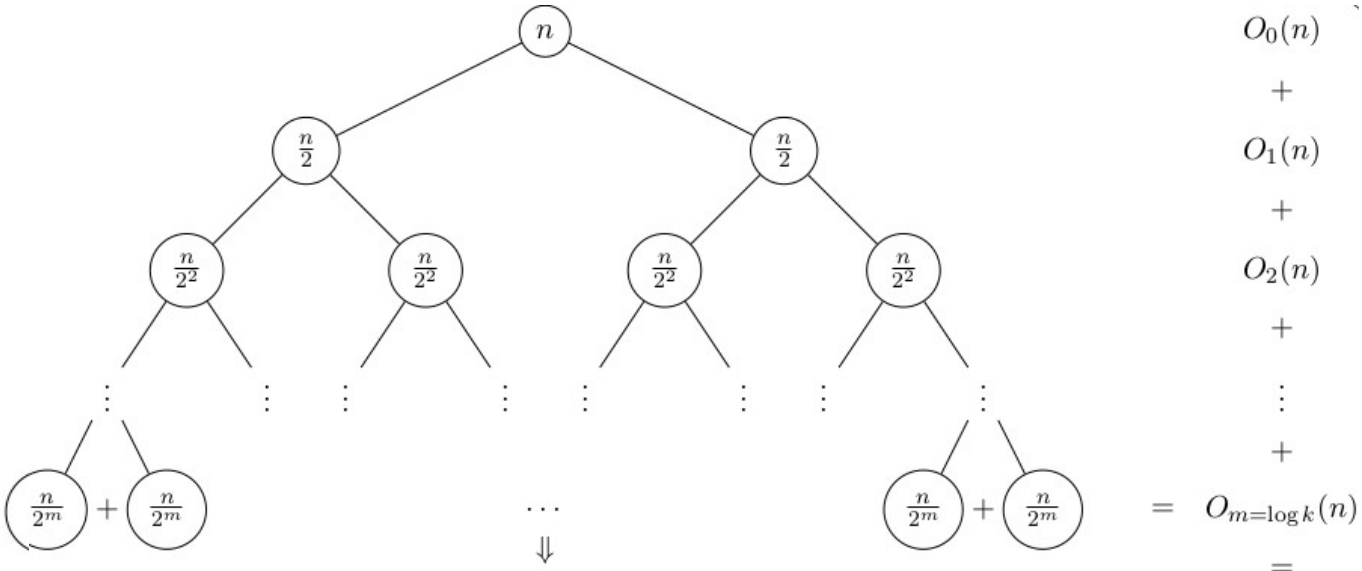
Η ιδέα του αλγορίθμου είναι να βρούμε το $(\frac{n}{k})-στό, (\frac{2n}{k})-στό, \ldots, (\frac{(k-1)n}{k})-στό, n-στό$

μικρότερο στοιχείο του πίνακα σε χρόνο $k*O(n)=O(k*n)$, όχι όμως απαραίτητα με αυτή τη σειρά.

Μπορούμε λοιπόν να βρούμε το μεσαίο στοιχείο από τα ζητούμενα, γνωστό και ως Median, δηλαδή το $\frac{n}{k}(\frac{k}{2})-στό$ στοιχείο.

Θα κάνω λοιπόν partition τον αρχικό πίνακα γύρω από το σημείο αυτό. Έπειτα θα εφαρμόσω την ίδια διαδικασία αναδρομικά σε κάθε υποπίνακα. Συνεπώς επιλέγω ένα divide and conquer αλγόριθμο.

Το δέντρο αναδρομής που προκύπτει είναι το ακόλουθο:



$$\Downarrow$$

$$O\left(\sum_{i=0}^{m} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{m} n\right) = O(m \cdot n) \Leftrightarrow O(n \cdot \log k)$$

Έχω $T(n)=T(\frac{n}{2})+\Theta(n)$ και $T(\frac{n}{k})=1$

Από τις σχέσεις αυτές προκύπτει $T(n)=\Theta(nlogk)$

Μένει πλέον να αποδειχθεί το κάτω όριο του αλγορίθμου. Θα αναζητήσουμε λοιπόν φράγμα κατώτερο του nlogk. Η ιδέα είναι πως ένας βέλτιστος αλγόριθμος δεν θα πρέπει να υπολογίζει όλους τους πιθανούς συνδυασμούς – φύλλα που μπορεί να προκύψουν, καθώς δεν μας ενδιαφέρει εσωτερική διάταξη κάθε υποπίνακα. Κάθε συγκριτικός αλγόριθμος πρέπει να αντιμετωπίζει την εσωτερική αναδιάτααξη ως μία περίπτωση. Έτσι θα κατασκευάζεται δυαδικό δέντρο (αφού η σύγκριση $i \leq j$ είναι επιστρέφει True ή False). Το δυαδικό δέντρο θα έχει φύλλα πλήθους (τουλάχιστον) L :

$$L = \frac{n!}{q_1!q_2!\dots q_k!} = \frac{n!}{((\frac{n}{k})!)^k}$$

Το ύψος του δυαδικού δέντρου θα είναι logL (τουλάχιστον).

Επίσης σύμφωνα με το Stirling approximation theorem : $\log(n!)=nlogn-n+O(logn)$

Χρόνος Εκτέλεσης $\geq$

$$\begin{aligned}
\log \frac{n!}{((\frac{n}{k})!)^k} &= \log n! - k \log \left(\frac{n}{k}\right)! \overset{\text{Stirling's Apx}}{=} \\
&= n \log n - n + O(\log n) - n \log \left(\frac{n}{k}\right) + n - kO(\log \left(\frac{n}{k}\right)) \\
&= n \log k + O(\log n) - kO(\log(n/k)) \\
&= \Theta(n \log k)
\end{aligned}$$

Άρα κάθε συγκριτικός αλγόριθμος έχει χρόνο χειρότερης περίπτωσης $\Omega(nlogk)$.

2.α.2.

Στην περίπτωση που περιγράφεται έχουμε $\frac{n}{k}$ (υπο)πίνακες, κάθε ένας εκ των οποίων έχει k στοιχεία.

Αν κάνω mergesort σε κάθε ένα υποπίνακα θα έχω κόστος $O(\frac{n}{k}*\log(\frac{n}{k}))$.

Επειδή έχω k τέτοιους πίνακες να ταξινομήσω, θα έχω συνολικό κόστος:
$O(\frac{n}{k}*k*\log(\frac{n}{k}))=O(n*\log(\frac{n}{k}))$

Όσον αφορά το ερώτημα γιατί κάθε συγκριτικός αλγόριθμος έχει κόστος $\Omega(n*\log(\frac{n}{k}))$,
η απάντηση είναι γιατί η Mergesort για ταξινόμηση πίνακα n στοιχείων στη γενική περίπτωση έχει $\Omega(nlogn)=O(nlogn)$. Την τελευταία πρόταση τη θεωρώ δεδομένη. Η φιλοσοφία της έγκειται στη λογική του ερωτήματος 2.α.1. όπου στηρίζεται στη δυαδικότητα της πράξης της σύγκρισης και το δυαδικό δέντρο που χρειαζόμαστε για να λάβουμε όλα τα φύλλα – πιθανούς συνδυασμούς.

2.β.1.

Αφού έχουμε μία σχεδόν ταξινομημένη στοίβα, μπορούμε να χρησιμοποιήσουμε την Insertion Sort.
Η Insertion Sort χρειάζεται $\Theta(n)$ για να διαβάσει τον πίνακα και θα κάνει αντιμεταθέσεις αν κάποιο στοιχείο προηγείται κάποιου μικρότερού του στη στοίβα. Αρκεί λοιπόν να υπολογίσουμε τον αιρθμό των αντιμεταθέσεων που θα πραγματοποιηθούν.

Κάθε στοιχείο δημιουργεί το πολύ O (k) αντιμεταθέσεις. Επειδή έχω n στοιχεία, θα έχω συνολικό κόστος αντιμεταθέσεων $n*O(k)=O(kn)$.

Η χειρότερη περίπτωση είναι κάθε στοιχείο να βρίσκεται στην πιο "απομακρυσμένη" θέση που δύναται. Δηλαδή για το i στοιχείο η χείριστη περίπτωση είναι να βρίσκεται στην $k \pm i$ θέση.

Η ιδέα του αλγορίθμου είναι ότι για κάθε θέση από τις n, θέλουμε να την συμπληρώνουμε σε $O(logk)$ χρόνο. Οι ιδιότητες που πρέπει να εξυπηρετεί ο αλγόριθμος είναι η εύρεση ελαχίστου και η διαγραφή στοιχείου σε $O(logk)$ χρόνο.

Με χρήση Min Heap:
Δημιουργώ heap με τα πρώτα $k+1$ στοιχεία της στοίβας με κόστος $O(k)$.
Έπειτα για το πρώτο στοιχείο εκτελώ εύρεση min και διαγραφή από το Heap με κόστος $O(logk)$. Όμοια για το δεύτερο στοιχείο κόκ.

Άρα το συνολικό κόστος που έχω για Min Heap είναι $O(k+n*(1+logk))=O(nlogk)$

2.β.2.

Ένα υποσύνολο των στιγμιοτύπων του προβλήματος είναι οι πλήθους $\frac{n}{k}$ υποπίνακες μεγέθους k. Επίσης υπάρχουν k! πιθανές μεταθέσεις σε κάθε υποπίνακα. Επομένως, η πλήρης ταξινόμηση προκύπτει από ένα σύνολο πλήθους $(k!)^{(n/k)}$.

Αυτόματα προκύπτει ένα κάτω φράγμα $\frac{n}{k} * \log(k!) = \Omega(\frac{n}{k} * k * logk) = \Omega(n * logk)$

Επίσης ταυτίζεται με το άνω φράγμα, δηλαδή $O(nlogk)$. Η απόδειξη στηρίζεται στην απόδειξη της σχέσης $\log(n!) = \Omega(nlogn) = O(logn) = \Theta(logn)$ την οποία θεωρώ ως δεδομένη.

Άρα πολυπλοκότητα αλγορίθμου = κάτω φράγμα βέλτιστου αλγορίθμου = $\Theta(nlogk)$.

Άσκηση 3

3α.

Το πρόβλημα λύνεται σε σε λογαριθμικό χρόνο με δυαδική αναζήτηση.
Διαβάζουμε το πρώτο στοιχείο του πίνακα $(a[1])$ και ψάχνουμε να βρούμε τον ελάχιστο αριθμό k του πίνακα $a[k]$ για τον οποίο ισχύει $a[k]<a[1]$.

Βήμα 1: Το πρώτο βήμα είναι να ελέγξουμε το μεσαίο στοιχείο του πίνακα $a[\frac{n}{2}]$ με κόστος $O(1)$.

Βήμα 2: Αν είναι μεγαλύτερος από το $a[1]$ πετάμε το δεύτερο μισό του πίνακα και κάνουμε την ίδια εργασία στο πρώτο μισό ελέγχοντας το $a[\frac{n}{4}]$ κόκ. Διαφορετικά πετάμε το πρώτο μισό και ελέγχουμε το μεσαίο στοιχείο του δεύτερου μισού του αρχικού πίνακα, δηλαδή το $a[\frac{3n}{4}]$ κόκ.
Το βήμα 2 ολοκληρώνεται μόλις βρούμε τον πρώτο αριθμό που είναι μικρότερος από τον $a[i]$.
Η διακασία του βήματος 2 διαρκεί $O(logn)$.

Βήμα 3:
Έπειτα πρέπει να κάνουμε ακριβώς την ίδια διαδικασία προς τα αριστερά (δυαδική αναζήτηση) για να βρούμε τον ελάχιστο αριθμό που ικανοποιεί το ζητούμενο.Η διαδικασία του βήματος 3 έχει κόστος $O(logk)$, όπου προφανώς $k \leqslant n$.

Το συνολικό κόστος του αλγορίθμου είναι $O(1+logn+logk)=O(logn)$

3.β.

Θα κάνω δυαδική αναζήτηση για να βρω τη μέγιστη ημερήσια απόσταση που μπορεί να καλυφθεί.
Η είσοδος είναι οι διαδοχικές αποστάσεις $d_1,\dots,d_n$ και το πλήθος των ημερών k.
Σε κάθε επανάληψη i ελέγχω αν είναι δυνατόν να χωρί σω σε k μέρες το ταξίδι, διανύοντας συνολική απόσταση κάθε μέρα (το πολύ) $L_i \in$ {m, . . . , r } καλύπτοντας όλες τις πόλεις και σταματώντας πάντα στο τέλος της ημέρας σε πόλη, όπου {m, . . . , r } είναι το διάστημα της αναζήτησης.

Κρατάω ένα μετρητή sum με τον αριθμό των χιλιομέτρων της ημέρας που εξετάζουμε κάθε φορά και έναν μετρητή ημερών dcount,αρχικοποιώντας και τους δύο μετρητές ίσους με μηδέν.

Βήμα 1: Για κάθε πολή j ελέγχω εάν $sum+d_j \leq L_i$

Αν χωράει, δηλαδή αληθεύει η παραπάνω ανισότητα, τότε τοποθετώ την απόσταση στην ημέρα που εξετάζω. Αυξάνω το sum κατά d_j και κάνω το βήμα 1 για την επόμενη πόλη, j+1.

Αν δε χωράει, αλλάζω μέρα. Αυξάνω το μετρητή ημερών dcount, μηδενίζω το μετρητή sum, και επαναλαμβάνω το βήμα 1.

Βήμα 2: Εάν όταν τελειώσουν οι πόλεις, έχω πλήθος ημερών $dcount \leq k$,αναζητώ μικρότερη μέγιστη ημερήσια απόσταση, θέτοντας m = L_i .Διαφορετικά οι πόλεις δε χωράνε στις k ημέρες με μέγιστη ημερήσια απόσταση L_i , οπότε πρέπει να αυξήσω το m κάτα 1 και να συνεχίσω.

Το μόνο που μένει να αποσαφηνιστεί είναι το διάστημα {m,…,r) στο οποίο θα γίνει η αναζήτηση.

Άνω όρια:

Αν όλο το ταξίδι γινόταν σε μία ημέρα είχαμε διανύσει $r_1=\sum_i^n d_i$,άρα $r \leq r_1$.

Για να χρησιμοποιηθούν όλες οι ημέρες, το ελάχιστο που πρέπει να γίνει είναι να μπει από μία πόλη στις πρώτες k-1 ημέρες και οι υπόλοιπες στην τελευταία. Έτσι, θα είχαμε μέγιστη διαδρομή ημέρας

$r_2=max(d_1,\dots,d_{k-1},\sum_{i=k}^n d_i)$, άρα $r \leqslant r2$.

Επιλέγω το r να είναι το ελάχιστο εκ των δύο οριακών περιπτώσεων r_1,r_2 , δηλαδή
$r=min(r_1,r_2)$

Κάτω όρια:

Η ελάχιστη τιμή που μπορεί να λάβει το m, με μία πρώτη σκέψη είναι $m \geqslant d_{max}$.

Επίσης, $m \geqslant m_1, m_1 = \frac{1}{k}(\sum_{i=1}^{n} d_i)$, όπου m_1 ο μέσος όρος των χιλιομέτρων που πρέπει να δυανύονται ανά μέρα. Είναι προφανές ότι πρέπει το ελάχιστο κάτω όριο της μέγιστης ημερήσιας απόστασης να μην υπερβαίνει το m_1 Αν δεν ίσχυει η προηγούμενη πρόταση, το άθροισμα όλων των διανυόμενων χιλιομέτρων όλων των ημερών συνολικά θα ήταν μικρότερο από το άθροισμα όλων των δοθέντων αποστάσεων.

Εφ' όσον θέλουμε να ικανοποιούνται και οι δύο συνθήκες που αναφέρθηκαν, θα πρέπει το m να είναι το μέγιστο από τα όρια m_1 και d_{max} , δηλαδή $m = max(m_1, d_{max})$.

Σχετικά με την πολυπλοκότητα του αλγορίθμου, έχουμε $\Theta(n)$ κόστος σε κάθε επανάληψη, ενώ ο αριθμός των επαναλήψεων είναι $\log(m-r)$.

Προφανώς ισχύει $(m-r) = O(\sum_{i=1}^{n} d_i)$, βάσει των όσων αναφέρθηκαν στον ορισμό των ορίων , όπου $\sum_{i=1}^{n} d_i \geq n$.

Συνεπώς η χρονική πολυπλοκότητα του αλγορίθμου είναι $O(nlog(\sum_{i=1}^{n} d_i))$

Άσκηση 4

Ο αλγόριθμος σε μορφή ψευδοκώδικα είναι ο ακόλουθος:

```
for ( i=0 to n-1 )
   while ( A[A[i]] != A[i] )
      swap( A[i] , A[A[i]] )
   end_of_while
end_of_for

for ( i=0 to n–1 )
   if ( ( A[i] != i ) && ( A[A[i]] ) then
      print A[i]
       A[A[i]] = i;
   end_of_if
end_of_for
```

Το πρώτο πέρασμα τροποποιεί τον πίνακα με τέτοιο τρόπο ώστε αν το στοιχείο x υπάρχει μία φορά τουλάχιστον, τότε μία από τις εισόδους που αντιστοιχούν στο x θα βρίσκεται στη θέση $A[x]$.

Ο αλγόριθμος τρέχει σε χρόνο O(n) : Η περίπτωση swap συμβαίνει μόνο εάν υπάρχει κάποιο στοιχείο i τέτοιο ώστε $A[i] \neq i$, ενώ κάθε swap θέτει τουλάχιστον ένα στοιχείο με τέτοιο τρόπο ώστε $A[i] = i$, όπου η συνθήκη αυτή δεν ήταν αληθής πριν από την “παρέμβαση” της swap. Αυτό σημαίνει ότι ο συνολικός αριθμός των swaps που γίνονται , άρα και των while loops θα είναι θα είναι το πολύ $N-1$.

Η δεύτερη loop τυπώνει τις τιμές των x για τις οποίες $A[x] \neq x$ και είναι τα x εκείνα τα οποία είναι διπλά στον αρχικό πίνακα Α.

Άσκηση 5

5.α.

Αρχικά υπολογίζουμε το άθροισμα $L+a[i]$. Το πρώτο βήμα για την επίλυση του προβλήματος είναι να υπολογίσουμε το πρώτο στοιχείο j_1 του πίνακα b για τον οποίο ισχύει $b[j_1] \geq L+a[i]$.
Στη συνέχεια υπολογίζουμε το άθροισμα $M+a[i]$.Πρέπει τώρα να υπολογίσουμε το πρώτο στοιχείο j_2 για το οποίο ισχύει $M+a[i]<b[j_2]$.
Η αναζήτηση των δύο περιπτώσεων που στοιχείων του πίνακα b που αναφέρθηκαν μόλις δεν θα γίνει γραμμικά, καθώς μπορεί να γίνει σε χρόνο O(logm) με δυαδική αναζήτηση για την εύρεση και των δύο στοιχείων (με την προϋπόθεση ότι δεν μας ενδιαφέρει η επιρροή των σταθερών για την πολυπλοκότητα).

Έτσι, λαμβάνουμε το διάστημα { $b[j_1], b[j_2-1]$ } για το οποίο ικανοποιείται η ζητούμενη συνθήκη για τον πρώτο στοιχείο του πίνακα a (το $a[i]$) και αρχικοποιούμε το μετρητή counter: $counter = j_2 - j_1$.

Μένει λοιπόν να συνεχίσουμε για τα υπόλοιπα n-1 στοιχεία του πίνακα a και να εντοπίσουμε τα αντίστοιχα διαστήματα.

Εξετάζουμε το στοιχείο $a[2]$:
Η ιδέα είναι ότι για τον εντοπισμό του αντίστοιχου “κάτω φράγματος” στον πίνακα b , δεν χρειάζεται να αναζητήσουμε μετά από το στοιχείο j_1 του πίνακα j_1 .Προφανώς αφού $a[2]>a[1]$,τότε και το $a[2]$ θα ικανοποιεί τη συνθήκη για το κάτω φράγμα, άρα πρέπει να κάνουμε δυαδική αναζήτηση “αριστερά” από το $b[j_1]$.

Για το άνω φράγμα τώρα πρέπει να γίνει δυαδική αναζήτηση δεξιά από το στοιχείο $b[j_2]$ που αναφέρθηκε προηγουμένως, λόγω της σχέσης $a[2]>a[1]$.

Ποιό είναι όμως το κόστος της εύρεσης του ζητούμενου διαστήματος για το στοιχείο $a[2]$;

Για το κάτω φράγμα κάνουμε δυαδική αναζήτηση σε j_1 στοιχεία , ενώ για το άνω φράγμα σε $m-j_2$ στοιχεία.

Συνεχίζοντας έτσι επαγωγικά για να εντοπίσουμε τα φράγματα για τα n στοιχεία του πίνακα a[n], θα πρέπει να κάνουμε n επαναλήψεις με κόστος 2*log(m) για την πρώτη επανάληψη και κόστος το πολύ 2*logm για τις υπόποιπες n-1 επαναλήψεις, αφού κάθε τμήμα του πίνακα b που “πετάμε”, δεν το ξαναεξετάζουμε.

Έτσι, ο αλγόριθμός μας έχει υπολογιστική πολυπλοκότητα O(n+2log(2m))=O(n+logm)!!!!! .

5.β.

Διαιρούμε το πίνακα διαδοχικά, κάθε φορά στο μισό, όπως ακριβώς στη mergesort, σε χρόνο O(logn). Στη συνέχεια,μόλις φτάσουμε σε πίνακα μοναδιαίων στοιχείων κάνουμε merge αναδρομικά όπως ακριβώς στη merge και σε χρόνο O(n). Η διαφορά είναι ότι αντί να κάνουμε αντιμετάθεση στοιχείων όπως στη merge, μετράμε το πλήθος των αντιμεταθέσεων που θα γινόντουσαν στον πίνακα αν

i) a[i]→a[i]+L και
ii) a[i]→a[i]+M.

Τέλος, αθροίζουμε το πλήθος αυτό που προκύπτει από κάθε βήμα.
Ο αλγόριθος έχει κόστος O(n*logn).

5.γ.

Αρχικά, ορίζω τη μεταβλητή sum, την οποία αρχικοποιώ sum→-L και τη μεταβλητή counter που μετράει τα διαστήματα που ζητούνται.
Έπειτα αθροίζω κάθε στοιχείο του πίνακα: sum+=a[i++] όσο ισχύει η σχέση 0<sum<M-L.

Αυξάνω το counter κατά ένα, αφαιρώ από το sum το a[1] και ελέγχω εάν sum<M.
Αν ναι ψάχνω το πρώτο στοιχειο που ισχύει 0<sum<M-L. Διαφορετικά, συνεχίζω μέχρι να βρω το a[i] που ικανοποιεί την προαναφερθείσα σχέση.

Σε κάθε βήμα που αναφέρθηκε αυξάνω το counter κατά ένα. Ουσιαστικά στο τέλος έχω διατρέξει τον πίνακα ακριβώς μία φορά, κάνοντας πράξεις σταθερού κόστους σε κάθε βήμα (2 συγκρίσεις).

Συνεπώς ο αλγόριθμός έχει πολυπλοκότητα O(n).