Αλγόριθμοι και Πολυπλοκότητα

$1^{η}$ Εργασία

Κωστόπουλος Κωνσταντίνος

03117043

Άσκηση 1

(α)

Ακολουθούν κάποιες αποδείξεις που είναι απαραίτητες για την ασυμπτωτική διάταξη των συναρτήσεων

- $\mathbf{2^{(\log\log n)^4} = o(n)}$ γιατί $n = 2^{\log n}$ και $(\log\log n)^4 = o(\log n)$
- $\mathbf{\log\binom{2n}{n} = \Theta(n)}$

  Ισχύει

  $$4^n = 2^{2n} = (1+1)^{2n} = \sum_{k=0}^{2n}\binom{2n}{k}$$

  Εφόσον το $\binom{2n}{n}$ είναι ο μεγαλύτερος όρος του αθροίσματος θα ισχύει

  $$\binom{2n}{n} \le 4^n \le (2n+1)\cdot\binom{2n}{n}$$

  Άρα περνώντας το λογάριθμο

  $$\log\binom{2n}{n} \le 2n \le \log(2n+1) + \log\binom{2n}{n}$$

  Εφόσον η $\log(2n+1)$ είναι ασυμπτωτικά μικρότερη από την $\log\binom{2n}{n}$ προκύπτει το ζητούμενο.

- $\mathbf{\frac{\log n!}{(\log\log n)^5} = o(n\log n)}$
  Το παραπάνω ισχύει επειδή $\frac{\log n!}{(\log\log n)^5} = \Theta\left(\frac{n\log n}{(\log\log n)^5}\right)$
- $\mathbf{\sqrt{n}^{\log\log n!} = \omega(n\log n)}$

  $$\begin{aligned}
  \log n! &\ge c_1 n\log n &\Leftrightarrow\\
  \log\log n! &\ge \log c_1 + \log n + \log\log n &\Leftrightarrow\\
  \log\log n! &\ge c\log n &\Leftrightarrow\\
  \sqrt{n}^{\log\log n!} &\ge n^{c\log n} = \omega(n\log n)
  \end{aligned}$$

- $\mathbf{\sum_{k=1}^{n} k\cdot 2^k = \Theta(n^2\cdot 2^n)}$

  Ισχύει

  $$\sum_{k=\frac{n}{2}}^{n} k\cdot 2^k \le \sum_{k=1}^{n} k\cdot 2^k \le n^2\cdot 2^n$$

  Όπου

  $$\sum_{k=\frac{n}{2}}^{n} k\cdot 2^k \ge \frac{n}{2}\cdot\sum_{k=\frac{n}{2}}^{n} n\cdot 2^k = \frac{n^2}{2}\cdot\left(\sum_{k=0}^{n}2^k - \sum_{k=0}^{\frac{n}{2}}2^k\right) = n^2\cdot\left(2^n - 2^{\frac{n}{2}}\right)$$

  Άρα καταλήγουμε στο ότι

  $$n^2\cdot\left(2^n - 2^{\frac{n}{2}}\right) \le \sum_{k=1}^{n} k\cdot 2^k \le n^2\cdot 2^n$$

  Το οποίο αρκεί για να αποδείξουμε το ζητούμενο.

Άρα η ασυμπτωτική διάταξη των συναρτήσεων είναι η εξής:

$$g_1 = \sum_{k=1}^{n} \frac{k}{2^k} \quad = \Theta(1), \qquad g_2 = 2^{(\log\log n)^4} \; = o(n)$$

$$g_3 = n \cdot 2^{2^{2^{100}}} \quad = \Theta(n), \qquad g_4 = \log \binom{2n}{n} \; = \Theta(n)$$

$$g_5 = \frac{\log n!}{(\log\log n)^5} = \Theta\left(\frac{n\log n}{(\log\log n)^5}\right), \quad g_6 = \sqrt{n}^{\log\log n!} = \Theta(2^{0.5\log n \log\log n!})$$

$$g_7 = n \cdot \sum_{k=0}^{n} \binom{n}{k} = \Theta(n \cdot 2^n), \qquad g_8 = \sum_{k=1}^{n} k \cdot 2^k \; = \Theta(n^2 \cdot 2^n)$$

(β)

1. Ισχύει $n^{\log_b a} = n^{1+\log_3 2}$ και $f(n) = n^2 \cdot \log n$. Εφόσον $1 + \log_3 2 < 2$ και $6 \cdot f(\frac{n}{3}) \leq \frac{2}{3} f(n)$, είμαστε στην τρίτη περίπτωση του *Master Theorem* άρα
$$T(n) = \Theta(n^2 \cdot \log n)$$

2. Ισχύει $n^{\log_b a} = n^2$ και $f(n) = n^2 \cdot \log n$. Εφόσον οι εκθέτες του n είναι ίσοι, είμαστε στην δεύτερη περίπτωση του *Master Theorem* άρα
$$T(n) = \Theta(n^2 \cdot \log^2 n)$$

3. Ισχύει $n^{\log_b a} = n^{\log_3 11}$ και $f(n) = n^2 \cdot \log n$. Εφόσον $\log_3 11 > 2$, είμαστε στην πρώτη περίπτωση ττου *Master Theorem* άρα
$$T(n) = \Theta(n^{\log_3 11})$$

4. Ισχύει $f(n) = n$ και $\frac{1}{4} + \frac{1}{2} < 1$. Άρα είμαστε στην πρώτη περίπτωση του *Master Theorem* και
$$T(n) = \Theta(n)$$

5. Ισχύει $f(n) = n$ και $\frac{1}{4} + \frac{1}{4} + \frac{1}{2} = 1$. Άρα είμαστε στην δεύτερη περίπτωση του *Master Theorem* και
$$T(n) = \Theta(n \cdot \log n)$$

6. $T(n) = T(n^{\frac{2}{3}}) + \Theta(\log n)$. Άρα θα ισχύει $c_1 \cdot \log n + T(n^{\frac{2}{3}}) \leq T(n) \leq c_2 \cdot \log n + T(n^{\frac{2}{3}})$ για κάποια c_1, c_2. Αναπτύσσοντας το άθροισμα προκύπτει

$$c_1 \cdot \sum_{i=0}^{k} \log n^{\frac{2}{3}^i} \leq T(n) \leq c_2 \cdot \sum_{i=0}^{k} \log n^{\frac{2}{3}^i}$$
$$\Leftrightarrow c_1 \log n \cdot \sum_{i=0}^{k} \left(\frac{2}{3}\right)^i \leq T(n) \leq c_1 \log n \cdot \sum_{i=0}^{k} \left(\frac{2}{3}\right)^i$$

Όμως το $\sum_{i=0}^{\infty} \left(\frac{2}{3}\right)^i$ συγκλίνει ως άθροισμα γεωμετρικής προόδου με $\lambda < 1$. Άρα τελικά
$$T(n) = \Theta(\log n)$$

7. $n^{\log_b a} = n^{\log_3 1} = 1$ και $f(n) = n^{\frac{1}{2}}$. Ισχύει προφανώς ότι $f(\frac{n}{3}) \leq \frac{1}{\sqrt{3}} \cdot f(n)$ άρα είμαστε στην τρίτη περίπτωση του *Master Theorem* και
$$T(n) = \Theta(\sqrt{n})$$

Άσκηση 2

(α)

Για $i = n, n-1, \dots 1$:

- Αν $A[1] < A[i]$ τότε $rotate(A, i-1)$
- Αλλιώς αν $A[1] > A[i]$ τότε
  - $k = max(i, A[1])$
  - $rotate(A, k)$
  - $rotate(A, k-1)$

Ο παραπάνω αλγόριθμος διατηρεί σε κάθε επανάληψη το επίθημα του πίνακα A ταξινομημένο σε αύξουσα σειρά. Αν το επίθημα ενός πίνακα είναι ταξινομημένο μπορώ να προσθέσω ένα ακόμα στοιχείο σε αυτό και να το διατηρήσω ταξινομημένο με το πολύ δύο κινήσεις, χωρίς να χαλάσω τα ήδη ταξινομημένα στοιχεία, δηλαδή είτε δεξιά είτε αριστερά από το πρώτο στοιχείο του επιθέματος. Στην περίπτωση που το στοιχείο που θέλω να προσθέσω είναι μεγαλύτερο από το i, μπορώ να το προσθέσω κατευθείαν στην τελική του θέση, για αυτό και επιλέγω το μέγιστο ανάμεσα σε $i, A[1]$.

Εφόσον ο παραπάνω αλγόριθμος ταξινομεί πίνακα μεγέθους 2 το πολύ σε μία περιστροφή και χρειάζονται 2 περιστροφές για να προστεθεί επιπλέον στοιχείο, θα γίνουν το πολύ $2n-3$ περιστροφές για πίνακα μεγέθους n.

(β)

Για $i = n, n-1, \dots 1$:

- Αν $|A[1]| < |A[i]|$ τότε
  - Αν $A[1] > 0$ τότε $rotate(A, 1)$
  - $rotate(A, i-1)$
- Αν $|A[1]| > |A[i]|$ τότε
  - Αν $A[1] > 0$ τότε $rotate(A, 1)$
  - $k = max(i, |A[i]|)$
  - $rotate(A, k)$
  - $rotate(A, k-1)$

Στην ουσία γίνονται τα ίδια με πριν, μόνο που όταν περιστρέφω για να μετακινήσω ένα στοιχείο προσέχω να είναι αρνητικό, ώστε μετά την περιστροφή να γίνει θετικό. Επίσης, λόγω του ότι όταν περιστρέφω πέρα από το πρώτο στοιχείο του επιθέματος, κάνω και δεύτερη περιστροφή, τα στοιχεία του επιθέματος διατηρούνται και ταξινομημένα και θετικά.

Εφόσον τον πίνακα 2 στοιχείων τον ταξινομώ το πολύ σε 3 περιστροφές και προσθέτω ένα ακόμα στοιχείο με 3, θα γίνουν το πολύ $3n-3$ περιστροφές για πίνακα n στοιχείων.

(γ)

- Μεταφέρουμε τον αριθμό n με μία περιστροφή στην πρώτη θέση και κάνουμε άλλη μία περιστροφή για να το μετακινήσουμε στην τελευταία θέση.
- Αν υπάρχουν θετικά στοιχεία στον πίνακα εκτός του n, επιλέγουμε το μεγαλύτερο από αυτά, k. Άρα το $k+1$ θα έχει αρνητικό πρόσημο. Αν βρίσκεται δεξιότερα του k, το μεταφέρουμε πριν από το k με δύο περιστροφές, με αποτέλεσμα να είναι και τα δύο αρνητικά, αλλά αποτελούν ζεύγος. Αντίθετα, αν βρίσκεται αριστερά του, κάνουμε τα ίδια με το k, με αποτέλεσμα να καταλήξουν ζεύγος θετικών.
- Αν όλα τα στοιχεία είναι αρνητικά και ο πίνακας δεν είναι $[-1, -2, \cdots -n]$ τότε κάποιο $-k$ θα βρίσκεται δεξιά του $-(k+1)$ οπότε με δύο περιστροφές το μετακινούμε στην προηγούμενη θέση από το $-k$.

Με βάση αυτά, μπορούμε να δημιουργούμε ζεύγη με δύο κινήσεις. Αρκούν n ζεύγη, αν υποθέσουμε ότι η μετακίνηση του n στην τελευταία θέση αποτελεί ένα ζεύγος, για να ταξινομηθεί ο πίνακας, οπότε αρκούν $2n$ προσημασμένες περιστροφές.

Άσκηση 3

1.

Ο αλγόριθμος επαναλαμβάνει το εξής για κάθε αριθμό της τράπουλας διαδοχικά:

- Πρόσθεσε τον επόμενο αριθμό στην πρώτη στοίβα απο αριστερά προς τα δεξιά που μπορεί να μπει. Τα κορυφαία στοιχεία της στοίβας θα είναι λοιπόν ταξινομημένα οπότε αρκεί να κάνουμε *binary search* για να βρούμε την κατάλληλη. Αν δεν χωράει σε καμία φτιάξε καινούργια δεξιά όλων των υπολοίπων.

Αν ένας αριθμός δεν μπει στην πρώτη στοίβα που χωράει, τότε θα μπει σε κάποια με μεγαλύτερο κορυφαίο στοιχείο. Άρα θα αποκλείσει από αυτήν τη στοίβα περισσότερους αριθμούς. Άρα επιτυγχάνεται ο ελάχιστος αριθμός από στοίβες με αυτόν τον αλγόριθμο.

Εφόσον για κάθε στοιχείο γίνεται ένα *binary search*, η πολυπλοκότητά του είναι $O(n \log n)$.

2.

Έχοντας τις στοίβες που δημιούργησε ο παραπάνω αλγόριθμος μπορούμε να ταξινομήσουμε την τράπουλα αφαιρώντας το κορυφαίο χαρτί της πρώτης στοίβας που θα είναι και το μικρότερο και μετά να επαναλάβουμε τα εξής για τα υπόλοιπα:

- Κάνε *binary search* στα κορυφαία στοιχεία ψάχνοντας το τελευταίο αριθμό που αφαιρέθηκε. Το αποτέλεσμα θα είναι ο ακριβώς μεγαλύτερος αριθμός από αυτόν.

3.

Μετά το τέλος του αλγορίθμου οι στοίβες θα έχουν την εξής μορφή ([3, 2, 1], [4], [7, 5], [8, 6]). Το μήκος της μέγιστης αύξουσας υπακολουθίας είναι 4, δηλαδή η $[2, 4, 7, 8]$, όσο και το πλήθος των στοιβών.

4.

Αν είχαμε μόνο τα στοιχεία τη μέγιστης αύξουσας υπακολουθίας και αυτά ήταν k, θα χρειαζόμασταν k στοίβες. Προσθέτοντας και τους υπόλοιπους αριθμούς ενδιάμεσα, μπορεί να χρειαστούμε και περισσότερες. Άρα όντως χρειάζονται τουλάχιστον k στοίβες.

5.

Ο αλγόριθμος όπως αποδείχθηκε παράγει τις ελάχιστες δυνατές στοίβες. Για να δημιουργηθεί η τελευταία στοίβα, έχει εμφανιστεί στοιχείο σίγουρα μεγαλύτερο από όλα τα κορυφαία στοιχεία των υπόλοιπων στοιβών. Όλα αυτά τα κορυφαία στοιχεία των στοιβών εκείνη τη στιγμή, λοιπόν, αποτελούν αύξουσα υπακολουθία μήκους ίσου με το πλήθος των στοιβών. Άρα η μέγιστη αύξουσα υπακολουθία έχει μήκος τουλάχιστον ίσο με το πλήθος των ελάχιστων στοιβών.

Άρα λόγω και του προηγούμενου ερωτήματος, η μέγιστη αύξουσα υπακολουθία έχει μήκος ίσο με το πλήθος των ελάχιστων στοιβών, και ο αλγόριθμος υπολογίζει το μήκος της μέγιστης αύξουσας υπακολουθίας.

Η μέγιστη φθίνουσα υπακολουθία θα έχει μήκος το πολύ $n - k + 1$ γιατί δεν μπορεί να περιέχει πάνω από έναν αριθμό της μέγιστης αύξουσας υπακολουθίας.

6.

- Αν όλες οι στοίβες έχουν μήκος $l \leq m$, τότε χρειάζονται $n + 1$ στοίβες, γιατί σε n θα χώραγαν μόνο $n \cdot m$.
- Αν το πλήθος στοιβών είναι μικρότερο ή ίσο με n, τότε η μεγαλύτερη στοίβα δεν μπορεί να έχει μήκος $l \leq m$ γιατί τότε χωράνε $n \cdot m$ στοιχεία το πολύ. Άρα πρέπει να έχει τουλάχιστον $m + 1$.

Τα παραπάνω θυμίζουν την Αρχή του Περιστερώνα.

Άσκηση 4

1η Περίπτωση

Ο υπερυπολογιστής ουσιαστικά αποτελεί μία συνάρτηση η οποία υπολογίζει το $f(i,t)$, με $i \in (a_1, \dots a_n)$ ή $i \in (b_1, \dots b_n)$ και $t \in (0, \frac{L \cdot A}{V_{min}})$ όπου A είναι σταθερά που σχετίζεται με την ακρίβεια υπολογισμού του t, και η μέγιστη τιμή του υπολογίζεται αν υποθέσουμε ότι ένα σωματίδιο κινείται με την ελάχιστη ταχύτητα.

Θα κάνω *binary search* ως προς χρόνο, στην συνάρτηση f και θα κάνω τα εξής:

- Για την συγκεκριμένη στιγμή, βρες την μέγιστη θέση x_a ανάμεσα στα τύπου a και ελάχιστη x_b ανάμεσα στα τύπου b.
- Αν $x_a < x_b$ τότε δεν έχει υπάρξει σύγκρουση, οπότε κάνε *binary search* στο δεξί μισό.
- Αν $x_a > x_b$ τότε έχει ήδη υπάρξει σύγκρουση, οπότε κάνε *binary search* στο αριστερό μισό.

Η ορθότητα του αλγορίθμου προκύπτει από το γεγονός ότι για κάθε σωματίδιο καθώς αυξάνει ο χρόνος θα αυξάνει και η θέση του αν είναι τύπου a, ενώ θα μείνεται αν είναι τύπου b. Οπότε οι μέγιστες/ελάχιστες θέσεις της κάθε στιγμής είναι αύξουσες συναρτήσεις και μπορεί να γίνει *binary search*.

Η πολυπλοκότητα του παραπάνω αλγορίθμου είναι $O(n \cdot \log \frac{L}{V_{min}})$ επειδή η εύρεση μεγίστων/ελαχίστων απαιτεί $2n$ επαναλήψεις, και το *binary search* θα γίνει ανάμεσα στις δυνατές τιμές του χρόνου.

2η Περίπτωση

Τώρα ο υπολογιστής αποτελεί μία συνάρτηση $f(i,j)$, με $i \in (a_1, \dots a_n)$ και $j \in (b_1, \dots b_n)$. Ο αλγόριθμος επαναλαμβάνει τα εξής ξεκινώντας από το a_1:

- Βρες ελάχιστο χρόνο συνάντησης ανάμεσα στο τωρινό a_i με κάποιο b_j.
- Βρες ελάχιστο χρόνο συνάντησης ανάμεσα στο b_j με κάποιο a_k.
- Επανάλαβε ξεκινώντας με το a_k.

Η ορθότητα του αλγορίθμου προκύπτει από το γεγονός ότι αν είχε γίνει κάποια σύγρουση σε χρόνο μικρότερο από αυτόν στον οποίο συγκρούστηκε ένα a με το ταχύτερο b, αυτή θα έχει γίνει ανάμεσα σε αυτό το b με κάποιο a που πέρασε το προηγούμενο a. Άρα κάθε φορά βρίσκουμε συγκρούσεις ανάμεσα στα γρηγορότερα σωματίδια.

Το να βρω ελάχιστο χρόνο κάθε φορά γίνεται σε n επαναλήψεις, εφόσον πρέπει να διασχίσει την γραμμή/στήλη. Κάθε ελάχιστο που βρίσκω βελτιώνει το προηγούμενο κατά τουλάχιστον $\frac{L \cdot A}{V_{max}}$ καθώς η μέγιστη ταχύτητα είναι V_{max} και ανά δύο ελάχιστα έχουμε ένα κοινό σωματίδιο. Άρα αυτά θα επαναληφθούν $\frac{\frac{L \cdot A}{V_{min}}}{\frac{L \cdot A}{V_{max}}} = \frac{V_{max}}{V_{min}}$ φορές. Τελικά η πολυπλοκότητα του παραπάνω αλγορίθμου προκύπτει $O(n \cdot \frac{V_{max}}{V_{min}})$.