**Εθνικό Μετσόβιο Πολυτεχνείο**
**Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών**
**Αλγόριθμοι & Πολυπλοκότητα – 7ο Εξάμηνο 2018-2019**
**Αβραμίδης Κλεάνθης – ΑΜ: 03115117**

# 1η Γραπτή Άσκηση – Απαντήσεις

## ➢ *Θέμα 1*

**a)** Για την κατάταξη των συναρτήσεων σημειώνουμε τα εξής:

1. $n^2 = O(n^2)$
2. $2^{\log_2^4 n} = 2^{\log_2 n * \log_2^3 n} = n^{\log_2^3 n} = O(n^{\log^3 n})$
3. $\frac{\log(n!)}{\log^3 n} \leq \frac{n\log n}{\log^3 n} = O\left(\frac{n}{\log^2 n}\right)$
4. $n * 2^{2^{2^{100}}} = O(n)$
5. $\log\binom{n}{\log n} = \log\left(\frac{n!}{\log n!(n-\log n)!}\right) = \log\left(\frac{n(n-1)(n-2)\dots(n-\log n+1)}{\log n!}\right) \leq \log\left(n^{\log n}\right) - \log(\log n!) = O(\log^2 n)$
6. $\frac{\log^2 n}{\log(\log n)} = O\left(\frac{\log^2 n}{\log(\log n)}\right)$
7. $\log^4 n = O(\log^4 n)$
8. $(\sqrt{n})! = O\left(\sqrt{n}^{\sqrt{n}}\right)$
9. $\binom{n}{6} = \frac{n!}{6!(n-6)!} = \frac{n(n-1)\dots(n-6)}{6!} = O(n^6)$
10. $\frac{n^3}{\log^8 n} = O\left(\frac{n^3}{\log^8 n}\right)$
11. $(\log_2 n)^{\log_2 n} = a \leftrightarrow \log_2 a = \log_2 n * \log_2(\log_2 n) \leftrightarrow a = \left(2^{\log_2 n}\right)^{\log_2(\log_2 n)} = O(n^{\log\log n})$
12. $\log\binom{2n}{n} = \log\left(\frac{2n!}{n!n!}\right) = \log\left(\frac{2n(2n-1)(2n-2)\dots(n+1)}{n!}\right) \leq \log(2n)^n - \log(n!) = O(n\log n)$
13. $n\sum_{k=0}^{n}\binom{n}{k} = n(1+1)^n = O(n2^n)$
14. $\sqrt{n}^{\log_2 \log_2(n!)} \leq n^{\frac{1}{2}\log_2 n \log_2 n} = O\left(n^{\frac{\log(n\log n)}{2}}\right)$
15. $\sum_{k=1}^{n} k2^k = n2^{n+1} - 2^{n+1} + 2 = O(n2^n)$
16. $\sum_{k=1}^{n} k2^{-k} = 2^{-n}(-n + 2^{n+1} - 2) = O(1)$

Πέραν των προφανών σχέσεων διάταξης των διάφορων O(g(n)) σημειώνουμε τα εξής μη προφανή:

- $O(\log^4 n) < O\left(\frac{n}{\log^2 n}\right)$ αφού $\lim \frac{n}{\log^4 n * \log^2 n} = \infty$
- $O(n\log n) < O\left(\frac{n^3}{\log^8 n}\right)$ αφού $\lim \frac{\frac{n^3}{\log^8 n}}{n\log n} = \lim \frac{n^2}{\log^9 n} = \infty$
- $O\left(n^{\frac{\log(n\log n)}{2}}\right) < O(n^{\log^3 n})$ αφού $\lim \frac{n^{\log^3 n}}{n^{\frac{\log(n\log n)}{2}}} = \lim \frac{n^{\log n * \log n * \log n}}{n^{\frac{\log n + \log(\log n)}{2}}} = \infty$
- $O\left(\sqrt{n}^{\sqrt{n}}\right) = O\left(n^{\frac{\sqrt{n}}{2}}\right) > O\left(n^{\log^3 n}\right)$ αφού $\lim \frac{\sqrt{n}}{2\log^3 n} = \lim \frac{n}{\varepsilon\sqrt{n}} = \infty$
- $O\left(\sqrt{n}^{\sqrt{n}}\right) < O(n2^n)$ αφού $\lim \frac{n2^n}{\sqrt{n}^{\sqrt{n}}} = \infty$

Η τελική σειρά με βάση την παραπάνω αρίθμηση: **16 – 6 – 5 – 7 – 3 – 4 – 12 – 1 – 10 – 9 – 11 – 14 – 2 – 8 – 13/15**

**b)**

1. $T(n) = 2 * T\left(\frac{n}{3}\right) + n\log n$

   Εφαρμόζουμε Master Theorem με παραμέτρους $a = 2,\ b = 3,\ f(n) = n\log n$, οπότε $n^{\log_3 2} = n^{0.63}$ . Η $f(n)$ είναι πολυωνυμικά μεγαλύτερη της $n^{0.63}$ οπότε εφαρμόζεται η περίπτωση (3): $\mathbf{T(n) = \Theta(n\log n)}$.

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

Εφαρμόζουμε την επαναληπτική μέθοδο:

$$T(n) = 3 * T\left(\frac{n}{3}\right) + nlogn$$

$$T(n) = 3 * \left(3 * T\left(\frac{n}{9}\right) + \frac{n}{3}\log\left(\frac{n}{3}\right)\right) + nlogn$$

$$T(n) = 3 * \left(3 * \left(3 * T\left(\frac{n}{27}\right) + \frac{n}{9}\log\left(\frac{n}{9}\right)\right) + \frac{n}{3}\log\left(\frac{n}{3}\right)\right) + nlogn$$

...

$$T(n) = 3^{i+1} * T\left(\frac{n}{3^{i+1}}\right) + \sum_{k=0}^{i}\left[3^k * \frac{n}{3^k}\log\left(\frac{n}{3^k}\right)\right]$$

$$T(n) = 3^{i+1} * T\left(\frac{n}{3^{i+1}}\right) + n * \sum_{k=0}^{i}[logn - k * log3]$$

$$T(n) = 3^{i+1} * T\left(\frac{n}{3^{i+1}}\right) + nlogn * (i+1) - nlog3 * \frac{i(i+1)}{2}$$

Θέτοντας τώρα $\frac{n}{3^{i+1}} = 1 \leftrightarrow i = \log_3 n - 1$, έχουμε:

$$T(n) = n * T(1) + nlogn * \log_3 n - nlog3 * \frac{\log_3^2 n - \log_3 n}{2} \rightarrow$$

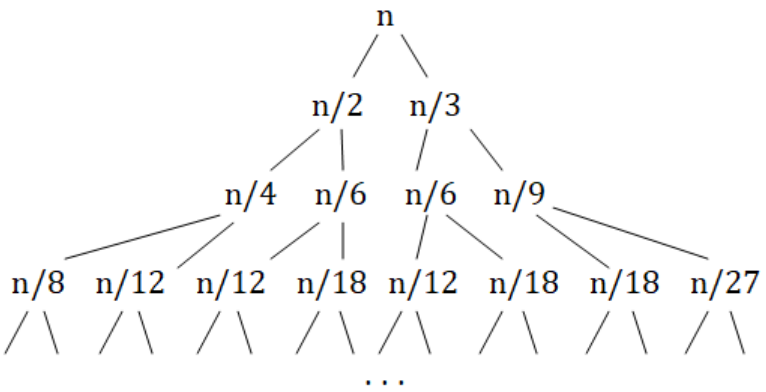
$$\mathbf{T(n) = \Theta\left(n + nlogn * \log_3 n - nlog3 * \frac{\log_3^2 n - \log_3 n}{2}\right) = \Theta(nlog^2 n)}$$

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

Εφαρμόζουμε Master Theorem με παραμέτρους $a = 4,\ b = 3,\ f(n) = nlogn$, οπότε $n^{\log_3 4} = n^{1.27}$. Η $f(n)$ είναι πολυωνυμικά μικρότερη της $n^{1.27}$ οπότε εφαρμόζεται η περίπτωση (1): $\mathbf{T(n) = \Theta(n^{1.27})}$.

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

Εφαρμόζουμε τη μέθοδο του δέντρου αναδρομής. Υποθέτοντας ότι τα Τ καταλήγουν σε 1 στα φύλλα του δέντρου, αρκεί να δούμε πώς μεταβάλλονται οι υπόλοιποι παράγοντες, δηλαδή το n:




Πιο αργά θα φτάσει στο 1 ο τέρμα αριστερά κλάδος του δέντρου, οπότε το ύψος του θα είναι $\log_2 n$. Παρατηρούμε πως αθροιζόμενοι οι συντελεστές στην τυχαία γραμμή k αποτελούν το $\left(\frac{1}{2} + \frac{1}{3}\right)^k * n = n\left(\frac{5}{6}\right)^k$. Αυτή είναι η συνεισφορά κάθε γραμμής. Συνολικά είναι λοιπόν:

$$\sum_{k=0}^{\log_3 n} n\left(\frac{5}{6}\right)^k = n * \frac{\left(1-\left(\frac{5}{6}\right)^{\log_3 n+1}\right)}{1-\frac{5}{6}} = 6n\left(1-\frac{5}{6}n^{\log_3\frac{5}{6}}\right) = 6n - 5n^{1-0.166} \rightarrow \mathbf{T(n) = \Theta(n)}$$

5. $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{3}\right) + T\left(\frac{n}{6}\right) + n$
   Αντίστοιχα με παραπάνω, φτιάχνουμε το δέντρο αναδρομής και λαμβάνουμε ύψος $\log_2 n$. Οι συντελεστές ανά γραμμή αποτελούν το $\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{6}\right)^k * n = n$. Συνολικά είναι $n * (\log_6 n + 1) \rightarrow \mathbf{T(n) = \Theta(nlogn)}$

6. $T(n) = T\left(n^{5/6}\right) + \Theta(\log n)$
   Εφαρμόζουμε τη μέθοδο της αλλαγής μεταβλητών. Συγκεκριμένα θέτουμε $m = \log n \leftrightarrow n = 2^m$ και η εξίσωση γράφεται $T(2^m) = T\left(2^{\frac{5m}{6}}\right) + \Theta(m)$ . Αν θεωρήσουμε $T(2^m) \equiv S(m)$, η εξίσωση γράφεται $S(m) = S\left(\frac{m}{6/5}\right) + \Theta(m)$ . Εφαρμόζουμε τώρα Master Theorem με παραμέτρους $a = 1,\ b = \frac{6}{5}$, οπότε $n^{\log_{6/5} 1} = 1$ . Η $\Theta(m)$ είναι πολυωνυμικά μεγαλύτερη οπότε εφαρμόζεται η περίπτωση (3): $S(m) = \Theta(m)$. Με αντικατάσταση λαμβάνουμε $\mathbf{T(n) = \Theta(logn)}$ .

7. $T(n) = T\left(\frac{n}{4}\right) + \sqrt{n}$
   Εφαρμόζουμε Master Theorem με παραμέτρους $a = 1,\ b = 4,\ f(n) = \sqrt{n}$, οπότε $n^{\log_4 1} = 1$ . Η $f(n)$ είναι πολυωνυμικά μεγαλύτερη οπότε εφαρμόζεται η περίπτωση (3): $\mathbf{T(n) = \Theta(\sqrt{n})}$.

- ***Θέμα 2***

**a)** Θεωρούμε τον πίνακα A n στοιχείων όπως δίνεται στην εκφώνηση, τον οποίο θέλουμε να ταξινομήσουμε κατά k μέρη. Το n είναι πολλαπλάσιο του k και χωρίς βλάβη της γενικότητας μπορούμε να θεωρήσουμε ότι αμφότεροι οι αριθμοί είναι δυνάμεις του 2 (αν δεν είναι, προσθέτουμε ∞, όσα απαιτούνται επιπλέον, στο τέλος του πίνακα). Έτσι κάθε υποπίνακας έχει n/k στοιχεία. Για τη ζητούμενη ταξινόμηση βρίσκουμε τη διάμεσο του A και διατάσσουμε τα υπόλοιπα στοιχεία, ώστε αριστερά να βρίσκονται όλα τα μικρότερα και δεξιά όλα τα μεγαλύτερα (ή ίσα). Έτσι ο A ταξινομείται κατά 2 ίσα μέρη. Στο επόμενο στάδιο εφαρμόζουμε την ίδια διαδικασία για κάθε υποπίνακα που σχηματίζεται (ταξινόμηση κατά 4 μέρη). Συνολικά την εφαρμόζουμε σε logk στάδια, έτσι ώστε να δημιουργηθούν k υποπίνακες και ο αρχικός πίνακας A να ταξινομηθεί κατά k ίσα μέρη. Η αναδιάταξη των στοιχείων γύρω τη διάμεσο δ σε δεδομένο πίνακα T[1…m] γίνεται σε γραμμικό χρόνο:

1. Τοποθετούμε 2 δείκτες: $p \rightarrow T[1]$ & $q \rightarrow T[m]$ .
2. Αν $*p \leq \delta$ τότε $p = p + 1$ . Αντίστοιχα, αν $*q > \delta$ τότε $q = q - 1$.
3. Επαναλαμβάνουμε το βήμα 2 μέχρι να μην είναι δυνατή άλλη μετατόπιση.
4. Κάνουμε swap τα στοιχεία $*p$ & $*q$ και επιστρέφουμε στο βήμα 2.
5. Επαναλαμβάνουμε τη διαδικασία μέχρι να ελεγχθούν όλα τα στοιχεία (m-2 μετατοπίσεις).

Ο πίνακας στον οποίο καταλήγουμε είναι (προφανώς λόγω των συνθηκών στο βήμα 2) ταξινομημένος κατά 2 μέρη και ο αλγόριθμος έχει (προφανώς λόγω της συνθήκης στο βήμα 5) πολυπλοκότητα $\Theta(m-2)$. Στο συγκεκριμένο πρόβλημα θα πρέπει να συμπεριλάβουμε στην πολυπλοκότητα και το χρόνο εύρεσης της διαμέσου για κάθε πίνακα που είναι γραμμική, σύμφωνα με τον αλγόριθμο *median-of-the-medians*[1]. Η πολυπλοκότητα τελικά της παραπάνω διαδικασίας είναι $\Theta(m + m - 2) = \Theta(m)$ και σε καθένα από τα logk στάδια, θα προσπελαθούν n στοιχεία. Καταλήγουμε έτσι στην συνολική πολυπλοκότητα: $O(nlogk)$.

Θα αποδείξουμε τώρα πως κάθε ανάλογος συγκριτικός αλγόριθμος έχει κάτω φράγμα στο χρόνο εκτέλεσης Ω(nlogk), ισοδύναμα ότι ο παραπάνω αλγόριθμος είναι βέλτιστος. Ένας συγκριτικός αλγόριθμος μπορεί να αναπαρασταθεί με το αντίστοιχο decision tree ύψους h, ίσου με τον αριθμό των συγκρίσεων στη χειρότερη

περίπτωση. Για τη ζητούμενη ταξινόμηση, το δέντρο θα έχει $\left(\frac{n}{\frac{n}{k}\frac{n}{k}\frac{n}{k}\dots\frac{n}{k}}\right) = \frac{n!}{\left(\frac{n}{k}!\right)^k}$ φύλλα (πιθανές διατάξεις n στοιχείων σε k ομάδες των n/k). Εφόσον η σύγκριση έχει 2 δυνατά αποτελέσματα, το δέντρο είναι δυαδικό και έχει το πολύ $2^h$ φύλλα (h το ύψος). Άρα θα πρέπει

$$h \geq \log\frac{n!}{\left(\frac{n}{k}!\right)^k} = \log(n!) - k * \log\left(\frac{n}{k}!\right) \geq n\log n - k * \frac{n}{k}\log\left(\frac{n}{k}\right) = n\log\left(\frac{n}{n/k}\right) = \Omega(n\log k)$$

Θέλουμε τέλος να ταξινομήσουμε πλήρως έναν ταξινομημένο κατά k μέρη πίνακα n στοιχείων, όπως ορίστηκε παραπάνω. Το πρόβλημα ανάγεται στην ταξινόμηση των στοιχείων σε καθένα από τα k groups των n/k στοιχείων, το οποίο έχει πολυπλοκότητα $\frac{n}{k} * log\frac{n}{k} * k = O\left(nlog\left(\frac{n}{k}\right)\right)$. Για να αποδείξουμε πως και αυτός ο αλγόριθμος είναι βέλτιστος, αρκεί να αποδείξουμε πως η διαδικασία της ταξινόμησης (με συγκριτικό αλγόριθμο) έχει κάτω φράγμα χρονικής εκτέλεσης $\Omega(n\log n)$. Ακολουθούμε πάλι την παραπάνω διαδικασία για $k = n$ :

$$h \geq \log(n!) \geq \sum_{i=\frac{n}{2}}^{n} \log i \geq \frac{n}{2} * \log\left(\frac{n}{2}\right) = \Omega(n\log n)$$

**b)** Για να εκμεταλλευτούμε το μικρό πλήθος των διαφορετικών στοιχείων του A, θα θεωρήσουμε ένα δέντρο αναζήτησης (πχ. AVL Tree) το οποίο σε κάθε κόμβο του, ένα για κάθε διαφορετικό στοιχείο, θα υπάρχει ένα πεδίο που θα καταγράφει πόσες φορές εμφανίστηκε. Ακολουθούμε γραμμικά για κάθε στοιχείο την εξής διαδικασία: Το αναζητούμε στο δέντρο (λογαριθμικός χρόνος). Αν το βρούμε, αυξάνουμε τον μετρητή του αντίστοιχου κόμβου κατά 1 (σταθερός χρόνος) ενώ στην αντίθετη περίπτωση το προσθέτουμε σε κατάλληλη θέση (λογαριθμικός χρόνος στη χειρότερη περίπτωση). Συνολικά λοιπόν απαιτείται χρόνος $O\left(n\log\left(\log^d n\right)\right)$. Για να ταξινομήσουμε τον πίνακα αρκεί τώρα να διασχίσουμε in-order το δέντρο τοποθετώντας σε ένα νέο πίνακα τα στοιχεία με τη σειρά και κάθε στοιχείο όσες φορές δείχνει ο μετρητής του. Αυτό γίνεται σε γραμμικό χρόνο αφού η ταξινόμηση έχει ήδη επιτευχθεί μέσω του δέντρου. Η πολυπλοκότητα της καθαρής αναζήτησης παραμένει $\Omega(n\log n)$ και στην προκειμένη περίπτωση. Ωστόσο, στην πραγματικότητα ταξινομούμε $\log n$ στοιχεία μέσω του δέντρου και τα υπόλοιπα που είναι επαναλήψεις τα λαμβάνουμε υπόψη μέσω των μετρητών. Οι επαναλήψεις δηλαδή των στοιχείων επιβαρύνουν απλά την πολυπλοκότητα ταξινόμησης $\log n * \log(\log n)$ σε $n * \log(\log n)$. Η συνολική πολυπλοκότητα του συγκεκριμένου αλγόριθμου θα είναι:

$$O\left(n\log\left(\log^d n\right) + n\right) = O\left(d * n\log(\log n)\right) = O\left(n\log(\log n)\right)$$

- ***Θέμα 3***

**a)** Τοποθετούμε 2 δείκτες στην αρχή των 2 ταξινομημένων πινάκων και καταγράφουμε τη διαφορά. Κάθε φορά επιλέγουμε τον δείκτη που δείχνει στο μικρότερο από τα 2 στοιχεία και τον αυξάνουμε κατά 1. Ελέγχουμε τη διαφορά και την καταγράφουμε, εφόσον είναι μικρότερη από την ήδη καταγεγραμμένη. Επαναλαμβάνουμε τη διαδικασία μέχρι να φτάσουμε στο τέλος των 2 πινάκων. Η διαφορά που θα έχουμε καταγεγραμμένη είναι η ζητούμενη. Ο αλγόριθμος αυτός δουλεύει καθώς βασίζεται στην αύξηση του μικρότερου από τους 2 διαθέσιμους κάθε φορά αριθμούς. Αυτή είναι η μόνη κίνηση που μπορεί να οδηγήσει σε μείωση της διαφοράς (αύξηση του μεγαλύτερου αριθμού σε ταξινομημένο πίνακα οδηγεί σίγουρα σε αύξηση της διαφοράς). Εφόσον ο αλγόριθμος απλά διατρέχει τους 2 πίνακες, είναι γραμμικού χρόνου με πολυπλοκότητα $O(n_1 + n_2)$ .

**b)** Κατ' αναλογία με πριν, τοποθετούμε m δείκτες στην αρχή των m ταξινομημένων πινάκων και σε κάθε επανάληψη αυξάνουμε τον δείκτη που δείχνει στο ελάχιστο στοιχείο. Στη συνέχεια ελέγχουμε τη διαφορά του μέγιστου από το ελάχιστο στοιχείο, από αυτά στα οποία δείχνουν οι δείκτες, κρατώντας κάθε φορά την ελάχιστη. Ο αλγόριθμος τώρα θα διατρέξει γραμμικά m πίνακες, όμως κάθε επανάληψη περιλαμβάνει την εύρεση του μέγιστου και του ελάχιστου στοιχείου, που απαιτούν χρόνο $O(m)$. Συνολικά λοιπόν η πολυπλοκότητα του αλγορίθμου θα είναι $O\left(m(n_1 + n_2 + \cdots + n_m)\right)$.

**c)** Ένας αλγόριθμος λύσης του παραπάνω προβλήματος θα πρέπει προφανώς να διατρέξει όλα τα διαθέσιμα στοιχεία, οπότε τον χρόνο εκτέλεσης καθορίζει και επιβαρύνει η αναζήτηση του μέγιστου και του ελάχιστου στοιχείου που γίνεται σε κάθε επανάληψη της διαδικασίας. Το μέγιστο στοιχείο στην πραγματικότητα δε χρειάζεται να αναζητηθεί, παρά μόνο την πρώτη φορά. Κατά τις επόμενες επαναλήψεις το μέγιστο θα είναι είτε το ίδιο με πριν, είτε θα είναι το στοιχείο στο οποίο δείχνει ο δείκτης που μόλις αυξήθηκε (όλα τα υπόλοιπα στοιχεία παραμένουν σταθερά).

Όσον αφορά την αναζήτηση του ελαχίστου, θα χρησιμοποιήσουμε min-heap, ένα δυαδικό δέντρο με κορυφή το μικρότερο στοιχείο του. Την πρώτη φορά θα χρειαστεί να γεμίσουμε το (άδειο) δέντρο με τα m πρώτα στοιχεία των m πινάκων, το οποίο γίνεται σε γραμμικό χρόνο[2]. Στη συνέχεια, σε κάθε επανάληψη θα χρειαστεί να ανανεώνουμε το δέντρο με το νέο στοιχείο (χρόνος logm για delete-insert) και να βρίσκουμε το min (σταθερός χρόνος). Συνολικά ο αλγόριθμος θα έχει πολυπλοκότητα τώρα: $\mathrm{O(m + (logm + 1) * N) = O(Nlogm)}$

- ***Θέμα 4***

**a)** Σε κάθε μία από τις $10^6$ φιάλες αντιστοιχούμε έναν εικοσάμπιτο δυαδικό αριθμό, από το $00000000000000000000\ (0_{10})$ έως το $11110100001000111111\ (999999_{10})$. Θέλουμε να βρούμε την τιμή της φιάλης με το φίλτρο και γι' αυτό θα χρειαστούμε 20 εθελοντές, έναν για τον έλεγχο κάθε ψηφίου του αριθμού. Συγκεκριμένα, ο εθελοντής για το LSB θα δοκιμάσει τις φιάλες που έχουν LSB = 1 (1, 3, 5, 7 κλπ). Αντίστοιχα, ο εθελοντής για το επόμενο ψηφίο θα δοκιμάσει τις φιάλες που έχουν 1 το ψηφίο αυτό (2, 3, 6, 7 κλπ). Ο εθελοντής για το MSB θα δοκιμάσει τις φιάλες $2^{19} - 999999$. Όποιος εθελοντής παρατηρηθεί να έχει πιει το μαγικό φίλτρο, συνεπάγεται πως το ψηφίο που του έχει αντιστοιχηθεί είναι 1, ενώ στην αντίθετη περίπτωση θα είναι 0. Κανείς δε χρειάζεται να πιει από την πρώτη φιάλη, καθώς αυτή αντιστοιχεί στο να μην επηρεαστεί κανένας εθελοντής. Σχηματίζεται έτσι ένας εικοσάμπιτος δυαδικός αριθμός που αντιστοιχεί στη φιάλη με το μαγικό φίλτρο. Γενικεύοντας, για n φιάλες θα χρειαστούμε $\mathrm{log}n$ εθελοντές.

Ο αριθμός των εθελοντών στον οποίο καταλήξαμε είναι ο ελάχιστος δυνατός: Έστω ότι είχαμε στη διάθεσή μας 19 εθελοντές. Καθένας αντιπροσωπεύει 2 πιθανές καταστάσεις στο συγκεκριμένο πείραμα (0 ή 1) οπότε ο συνδυασμός τους μπορεί να δώσει $2^{19}$ διαφορετικές καταστάσεις, λιγότερες από τα προς εξέταση φίλτρα.

**b)** Το πρόβλημα μοντελοποιείται ως: διαμέριση ενός πίνακα A n αριθμών σε k groups, έτσι ώστε το μέγιστο άθροισμα που απαντάται σε group να είναι το μικρότερο δυνατό. Θα απαντήσουμε στο πρόβλημα κάνοντας δυαδική αναζήτηση στο μέγιστο άθροισμα. Συγκεκριμένα, το μέγιστο άθροισμα μπορεί να πάρει τιμές από max(A) μέχρι $\sum_{i=1}^{n} A[i]$ . Εφαρμόζουμε τον αλγόριθμο δυαδικής αναζήτησης στο συγκεκριμένο διάστημα. Σε κάθε βήμα διαμερίζουμε γραμμικά τον A με βάση το εκάστοτε μέγιστο άθροισμα (ώστε κανένα group να μην έχει άθροισμα μεγαλύτερο από αυτό) και αν προκύψουν $\leq k$ groups αναζητούμε στο μισό με τα μικρότερα αθροίσματα μαζί με αυτό που μόλις εξετάστηκε. Αν από την άλλη προκύψουν $> k$ groups αναζητούμε στο άλλο μισό. Εκτελούμε εξαντλητικά τη δυαδική αναζήτηση, προκειμένου να καταλήξουμε στο μικρότερο δυνατό άθροισμα, με το οποίο ο A διαμερίζεται σε k groups. Η πολυπλοκότητα του αλγορίθμου με βάση την παραπάνω περιγραφή συνυπολογίζει λογαριθμικό χρόνο για τη δυαδική αναζήτηση και γραμμικό χρόνο σε κάθε βήμα της για τη διαμέριση του A. Συνολικά, στη χειρότερη περίπτωση έχουμε πολυπλοκότητα $\mathrm{O\big(nlog(sum(A) - maxA)\big)}$.

- ***Θέμα 5***

**a)** Εφαρμόζουμε δυαδική αναζήτηση (με στρογγυλοποίηση του mean προς τα κάτω) στο διάστημα $1 - \mathrm{M}$ και σε κάθε επανάληψη ελέγχουμε την τιμή της $\mathrm{F_s}$: Αν $\mathrm{Fs(x)} \geq \mathrm{k}$ συνεχίζουμε την αναζήτηση στο $\mathrm{min - x}$, ενώ στην αντίθετη περίπτωση αναζητούμε στο $\mathrm{(x + 1) - max}$. Απορρίπτουμε δηλαδή μόνο τα στοιχεία που δίνουν $\mathrm{F_s} < \mathrm{k}$ διότι αν το ζητούμενο στοιχείο a επαναλαμβάνεται στο multiset, θα ισχύει $\mathrm{F_s(\alpha)} > \mathrm{k}$. Συνεχίζουμε εξαντλητικά καταλήγοντας στο ζητούμενο στοιχείο. Αυτό γίνεται διότι όταν θα έχουν απορριφθεί όλα τα στοιχεία που δίνουν $\mathrm{F_s} < \mathrm{k}$ , ο αλγόριθμος θα καταλήξει στο min των υπολοίπων που δίνουν $\mathrm{F_s} \geq \mathrm{k}$. Το min αυτό είναι πράγματι το ζητούμενο στοιχείο, διότι τα υπόλοιπα είτε θα δίνουν μεγαλύτερη Fs (άρα κανένα δεν είναι το k-στο μικρότερο με δεδομένο το min) είτε θα δίνουν ίδια $\mathrm{F_s}$, οπότε δεν υπάρχουν στο σετ (αν υπήρχαν,

θα έδιναν $F_s$ τουλάχιστον ίση με του min +1, τον εαυτό τους). Το min αντίθετα υπάρχει στο σετ, αφού δίνει μεγαλύτερη $F_s$ από το προηγούμενό του που δίνει $Fs < k$. Στην αναζήτηση θα γίνουν $O(logM)$ κλήσεις της $F_s$ .

**b)** Το πρόβλημα με το σετ S που δημιουργείται μπορεί να αναχθεί πλήρως στο προηγούμενο ερώτημα, οπότε εφαρμόζουμε τον ίδιο αλγόριθμο. Απομένει να υλοποιήσουμε αποδοτικά την συνάρτηση κατανομής, ώστε η κλήση της να μην επιβαρύνει κατά πολύ την συνολική πολυπλοκότητα του αλγορίθμου. Καταρχήν απαιτείται να ταξινομήσουμε τον δοσμένο πίνακα, κάτι που θα μας επιβαρύνει με πολυπλοκότητα $O(nlogn)$. Η τιμή της Fs για δεδομένο όρισμα δίνεται πλέον σε γραμμικό χρόνο σύμφωνα με την εξής διαδικασία:

1. Τοποθετούμε 2 δείκτες right και left στην αρχή του πίνακα A.
2. Αυξάνουμε κατά 1 τον δείκτη right και ελέγχουμε τη διαφορά των στοιχείων στα οποία δείχνουν οι δείκτες. Αν η διαφορά είναι μικρότερη ή ίση του ορίσματος της συνάρτησης, αυξάνουμε έναν counter κατά τιμή right-left (πχ για τιμές δεικτών 0 και 1 έχουμε 1-0=1 διαφορά, για τιμές 0 και 2 έχουμε 2-0=2 νέες διαφορές, την 2-0 και την 2-1, τις οποίες προσθέτουμε στην προηγούμενη 1-0, σύνολο 3).
3. Επαναλαμβάνουμε το βήμα 2 μέχρι να προκύψει διαφορά μεγαλύτερη του ορίσματος. Τότε αυξάνουμε διαδοχικά τον δείκτη left κατά 1, μέχρι η διαφορά να πέσει σε αποδεκτές τιμές, οπότε και συνεχίζουμε όπως στο βήμα 2.
4. Μόλις και οι 2 δείκτες φτάσουν στο τέλος του πίνακα (χρόνος 2n) ο counter θα έχει την τελική τιμή που επιστρέφει η συνάρτηση κατανομής Fs.

Ο συνολικός χρόνος πλέον για το όλο πρόβλημα θα είναι $O(nlogn + nlogM)$.

- ✓ *Παρατηρήσεις επί των απαντήσεων:*

**[1]** https://rcoh.me/posts/linear-time-median-finding/

**[2]** https://www.quora.com/Inserting-an-element-in-a-heap-takes-O-log-n-Still-if-we-insert-n-elements-in-the-heap-it-comes-out-to-be-O-n?fbclid=IwAR2qiz6AisT2iV9W7JSDVVRJdGwUUedWtSlf3KR_vKAocQ9-upW8B7DZjWg