Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Αλγόριθμοι και Πολυπλοκότητα

Πρώτη σειρά γραπτών ασκήσεων

Σπουδαστής
Παπασκαρλάτος Αλέξανδρος (Α.Μ.: 03111097)

Ημερομηνία Υποβολής: 29 Οκτωβρίου 2018

Άσκηση 1

1α) Θα δείξουμε και θα αποδείξουμε απευθείας την τελική ταξινόμηση.
Στην κάτωθι αρίθμηση, οι συναρτήσεις είναι τοποθετημένες σε αύξουσα σειρά μεγέθους.
Οι συναρτήσεις (6) και (7) είναι ίδιας τάξης.
Οι συναρτήσεις (15) και (16) είναι ίδιας τάξης.

1. $\sum_{k=1}^{n} k\,2^{-k}$
2. $\frac{log^2 n}{loglogn}$
3. $log\,(\,(\,{}^{n}_{logn}\,)\,)$
4. $log^4 n$
5. $\frac{log(n!)}{log^3 n}$
6. $n\,2^{2^{2^{100}}}$ ~ 7. $log\,(\,(\,{}^{2n}_{n}\,)\,)$
8. n^2
9. $\frac{n^3}{log^2 n}$
10. $(\,{}^{n}_{6}\,)$
11. $(log_2 n)^{log_2 n}$
12. $(\sqrt{n})^{log_2 log_2 (n!)}$
13. $2^{(log_2 n)^4}$
14. $\sqrt{n}!$
15. $n \sum_{k=0}^{n} (\,{}^{n}_{k}\,)$ ~ 16. $\sum_{k=1}^{n} k\,2^k$

Απόδειξη ταξινόμησης

Σε κάθε βήμα, θα παίρνουμε την εκάστοτε συνάρτηση f και την αμέσως προηγούμενη g (ή ίσως την τάξη πολυπλοκότητας αυτών) και θα εξετάζουμε το όριο $\lim_{n\to\infty} f(n)/g(n)$.

Εάν το όριο δίνει πραγματικό αριθμό, f = Θ (g).
Εάν το όριο τείνει στο ∞ τότε η f είναι ασυμπτωτικά μεγαλύτερη της g.
Εάν το όριο έδινε 0, τότε η g θα ήταν ασυμπτωτικά μεγαλύτερη της f.

1. $\sum_{k=1}^{n} k\, 2^{-k}$

Για n → ∞, το παραπάνω άθροισμα δίνει αποτέλεσμα 2.
Συνεπώς, η συνάρτηση φράσσεται δεξιά από σταθερό αριθμό, οπότε έχει Θ(1).

2. f(n) = $\frac{log^2 n}{loglogn}$, g(n)=1

$$\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} \frac{log^2 n}{loglogn} = \lim_{n\to\infty} \frac{2\, log\, n\, (1/n)}{(1/logn)(1/n)} = \lim_{n\to\infty} 2\, log^2 n = \infty$$

Άρα f (n) = ω (1).

3. f(n) = $log\,(\,(\,{}^{n}_{logn}\,)\,)$, g(n) = $\frac{log^2 n}{loglogn}$
Είναι $(\,{}^{n}_{logn}\,)$ = n!/[(logn)! (n-logn)!] = n(n-1)(n-2)...(n-(logn -1)) / [logn (logn -1)…1]
Ο αριθμητής έχει logn το πλήθος όρους, ο μεγαλύτερος εκ των οποίων είναι ο n.
Συνεπώς $(\,{}^{n}_{logn}\,) \le n^{logn} \Leftrightarrow f(n) \le \log (n^{logn}) = \log^2 n$.
Επίσης, είναι $(\,{}^{n}_{logn}\,) \ge (n - logn +1)^{logn} /(logn)^{logn} \Leftrightarrow$
$\Leftrightarrow f(n) \ge \log ((n - logn +1)^{logn} /(logn)^{logn}) = logn\, log(n - logn +1) - logn\, loglogn$

Όμως $\lim_{n\to\infty} \log(n - logn +1)/logn = \lim_{n\to\infty} (n-1)/(n-logn+1) = \lim_{n\to\infty} n/(n-1) = 1$,
Άρα log(n - logn +1) = Θ (logn)

Οπότε logn log(n - logn +1) - logn loglogn = **Θ ($\log^2 n$)**, καθώς η loglogn είναι ασυμπτωτικά μικρότερη από την logn.

Συνεπώς, η f φράσσεται και από τις δύο μεριές από Θ ($\log^2 n$), άρα **f(n) = Θ ($\log^2 n$).**

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} \log^2 n/(\log^2 n/loglogn) = \lim_{n\to\infty} loglogn = \infty$

Άρα f (n) = ω ($\frac{log^2 n}{loglogn}$)

4. f(n) = $log^4 n$, g(n) = $\log^2 n$

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} \log^4 n/\log^2 n = \lim_{n\to\infty} \log^2 n = \infty$

Άρα f (n) = ω (log ($\binom{n}{logn}$))

5. f(n) = $\frac{log(n!)}{log^3 n}$, g(n) = $\log^4 n$

Είναι $\frac{log(n!)}{log^3 n} \sim \frac{nlogn}{log^3 n} = \frac{n}{log^2 n}$

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} (\frac{n}{log^2 n})/\log^4 n = \lim_{n\to\infty} (\frac{n}{log^6 n}) = \infty$

Άρα f (n) = ω ($\log^4 n$)

6. f(n) = $n\, 2^{2^{2^{100}}}$, g(n) = $\frac{n}{log^2 n}$

Είναι **f(n) = Θ(n).**

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} (\frac{n}{\frac{n}{log^2 n}}) = log^2 n = \infty$

Άρα f (n) = ω ($\frac{log(n!)}{log^3 n}$).

7. f(n) = log ($\binom{2n}{n}$) , g(n) = n

Είναι $\binom{2n}{n}$ = 2n!/(n! n!) = 2n (2n-1)...(n+1)/[n (n-1)...1]

Ο ι-οστός από του n όρους του αριθμητή είναι τουλάχιστον διπλάσιος από τον ι-οστό από τους n όρους του παρονομαστή.
Συνεπώς, $\binom{2n}{n} \geq 2^n \Leftrightarrow$ f(n) ≥ log (2^n) = n log2 = Θ(n), δηλαδή η f(n) φράσσεται αριστερά από τη Θ(n).

Επίσης, $\binom{2n}{n} \leq (2n)^n / n! = 2^n n^n / n!$.
Από την προσέγγιση του Stirling για το παραγοντικό
$n! \sim \sqrt{2\pi n}\,(\frac{n}{e})^n \Leftrightarrow n^n / n! \sim e^n / \sqrt{2\pi n}$
Είναι $\log(n^n / n!) = \Theta(\log(e^n / \sqrt{2\pi n}) = \Theta(\log(e^n) - \log(\sqrt{2\pi n}))$ =
= Θ (n - (½) logn) = Θ(n).
Έχουμε $\binom{2n}{n} \leq (2n)^n / n! = 2^n n^n / n! \Leftrightarrow$ f(n) ≤ log ($2^n n^n / n!$) = log(2^n) + log($n^n / n!$) =
= n log2 + log($n^n / n!$) = Θ(n+n) = Θ(n), δηλαδή η f(n) φράσσεται δεξιά από τη Θ(n).

Συνεπώς, η f φράσσεται και από τις δύο μεριές από τη Θ(n), άρα **f(n) = Θ(n)** .

Άρα f(n) = Θ($n\, 2^{2^{2^{100}}}$) .

8. f(n) = n^2 , g(n) = n

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty}(n) = \infty$

Άρα f (n) = ω (n).

9. f(n) = $\frac{n^3}{log^8 n}$, g(n) = n^2

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty}(n/log^8 n) = \infty$

Άρα f (n) = ω (n^2).

10. f(n) = $\binom{n}{6}$, g(n) = $\frac{n^3}{log^8 n}$

Είναι f(n) = $\binom{n}{6}$ = n!/[6! (n-6)!] = n (n-1) (n-2) (n-3) (n-4) (n-5) /6! = Θ(n^6)

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty}(\frac{n^6}{\frac{n^3}{log^8 n}}) = \lim_{n\to\infty}(n^3 log^8 n) = \infty$

Άρα f (n) = ω ($\frac{n^3}{log^8 n}$).

11. f(n) = $(log_2 n)^{log_2 n}$, g(n) = n^6

Είναι $\lim_{n\to\infty} f(n)/g(n)$ = $\lim_{n\to\infty} (log_2 n)^{log_2 n} / n^6$ = $\lim_{n\to\infty} 2^{\log n \log\log n} / 2^{6 \log n}$ = $\lim_{n\to\infty} 2^{\log n(\log\log n-6)} = \infty$

Άρα f (n) = ω ($\binom{n}{6}$).

12. f(n) = $(\sqrt{n})^{log_2 log_2 (n!)}$, g(n) = $(log_2 n)^{log_2 n}$

Είναι f(n) = $2^{(½) \log n \log\log(n!)}$

Είναι $\lim_{n\to\infty} f(n)/g(n)$ = $\lim_{n\to\infty} 2^{(½) \log n \log\log(n!)} / 2^{\log n \log\log n}$ = $\lim_{n\to\infty} 2^{\log n [(½) \log\log(n!) - \log\log n]} = \infty$

όμως το (½)loglog(n!) είναι ασυμπτωτικά μεγαλύτερο του loglogn καθώς:

(½)loglog(n!) = Θ(log (nlogn)) = Θ (logn +loglogn) = Θ (logn), άρα

$\lim_{n\to\infty}$ (½)loglog(n!) - loglogn = ∞ ⇔ $\lim_{n\to\infty} 2^{\log n [(½) \log\log(n!) - \log\log n]} = \infty$.

Άρα f(n) = ω ($(log_2 n)^{log_2 n}$).

13. f(n) = $2^{(log_2 n)^4}$, g(n) = $(\sqrt{n})^{log_2 log_2 (n!)}$

Είναι $\lim_{n\to\infty} f(n)/g(n)$ = $\lim_{n\to\infty} 2^{(log_2 n)^4} / 2^{(½) \log n \log\log(n!)}$ = $\lim_{n\to\infty} 2^{\log n [log^3 n - (1/2) log log(n!)]}$,

όμως το $\log^3 n$ είναι ασυμπτωτικά μεγαλύτερο του (½)loglog(n!), καθώς

(½)loglog(n!) = Θ(log (nlogn)) = Θ (logn +loglogn) = Θ (logn), άρα

$\lim_{n\to\infty} \log^3 n$ - (½)loglog(n!) = ∞ ⇔ $\lim_{n\to\infty} 2^{\log n [log^3 n - (1/2) log log(n!)]} = \infty$

Άρα f(n) = ω($(\sqrt{n})^{log_2 log_2 (n!)}$).

14. f(n) = $\sqrt{n}!$, g(n) = $2^{(log_2 n)^4}$

Είναι $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} 2^{\log(\sqrt{n}!)} / 2^{(log_2 n)^4} = \lim_{n\to\infty} 2^{\log(\sqrt{n}!) - (log_2 n)^4}$,

όμως το log($\sqrt{n}!$) είναι ασυμπτωτικά μεγαλύτερο από το $\log^4 n$, καθώς

log($\sqrt{n}!$) = Θ($\sqrt{n}$ log $\sqrt{n}$) = Θ($(1/2)\, n^{1/2}\, logn$) = Θ($n^{1/2}$ logn), άρα

$\lim_{n\to\infty} log(\sqrt{n}!) - \log^4 n = \infty \Leftrightarrow \lim_{n\to\infty} 2^{\log(\sqrt{n}!) - (log_2 n)^4} = \infty$

Άρα f(n) = ω($2^{(log_2 n)^4}$)

15. f(n) = $n \sum_{k=0}^{n} \binom{n}{k}$, g(n) = $\sqrt{n}!$

Από τον τύπο του διωνυμικού αναπτύγματος $\sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k} = (a+b)^n$ προκύπτει

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

Οπότε **f(n) = Θ(n 2^n).**

Είναι $\lim_{n\to\infty} 2^n / \sqrt{n}! = \lim_{n\to\infty} 2^n / 2^{\log(\sqrt{n}!)} = \lim_{n\to\infty} 2^{n - \log(\sqrt{n}!)}$,

όμως το n είναι ασυμπτωτικά μεγαλύτερο από το log($\sqrt{n}!$), καθώς

log($\sqrt{n}!$) = Θ($\sqrt{n}$ log $\sqrt{n}$) = Θ($(1/2)\, n^{1/2}\, logn$) = Θ($n^{1/2}$ logn), άρα

$\lim_{n\to\infty} n - log(\sqrt{n}!) = \infty \Leftrightarrow \lim_{n\to\infty} 2^{n - \log(\sqrt{n}!)} = \infty$

Άρα f(n) = ω($\sqrt{n}!$).

16. f(n) = $\sum_{k=1}^{n} k\, 2^k$, g(n) = n 2^n

Προφανώς n 2^n ≤ f(n) καθώς αυτός είναι μόνο ο τελευταίος όρος της f, δηλαδή η f(n) φράσσεται αρίστερά από Θ(n 2^n).

Επίσης, είναι f(n) < $n \sum_{k=1}^{n} 2^k$ = n $(2^{n+1} - 2)$ = 2 n 2^n - 2n = Θ(n 2^n).

Συνεπώς, η f φράσσεται και από τις δύο μεριές από Θ(n 2^n) και άρα **f(n) = Θ(n 2^n)**.

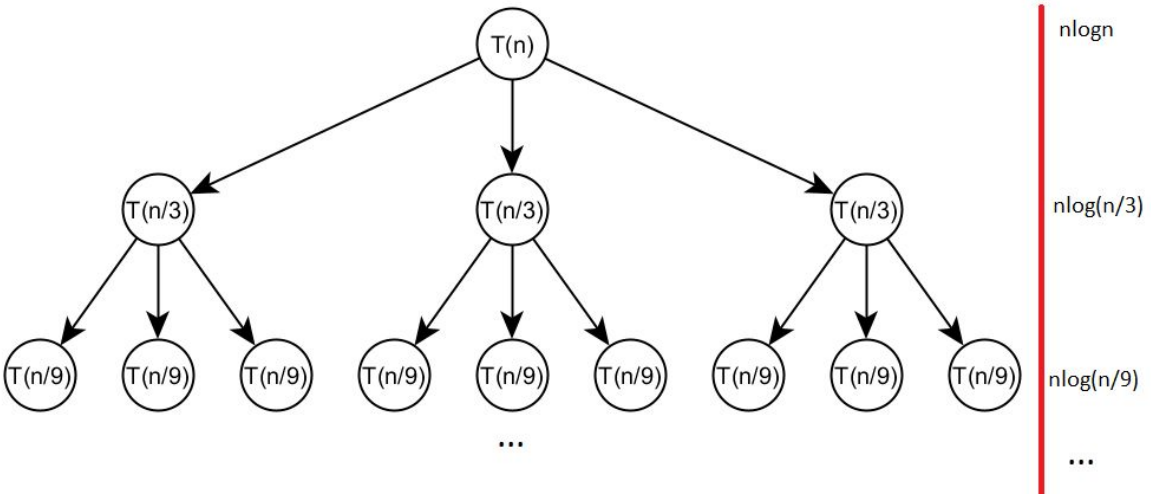
Άρα, f(n) = Θ($n \sum_{k=0}^{n} \binom{n}{k}$) .

1β)

1.T(n) = 2T(n/3) + nlogn

Είναι a=2, b=3, $n^{\log_3 2} < n$, και άρα σίγουρα ασυμπτωτικά μικρότερο από το f(n) = nlogn.
Επίσης, a f(n/b) = 2(n/3) log(n/3) < n logn = f(n).
Οπότε, από Master Theorem, **T(n) = Θ(nlogn)**.

2.T(n) = 3T(n/3) + nlogn



Είναι $n/3^h = 1 \Leftrightarrow h = \log_3 n$.

Συνολικά, έχουμε $\sum_{i=0}^{h} n \log (n/3^i) = n \log n \log_3 n - \sum_{i=0}^{h} \log(3^i) = n \log n \log_3 n - \log (\Pi(3^i)) =$

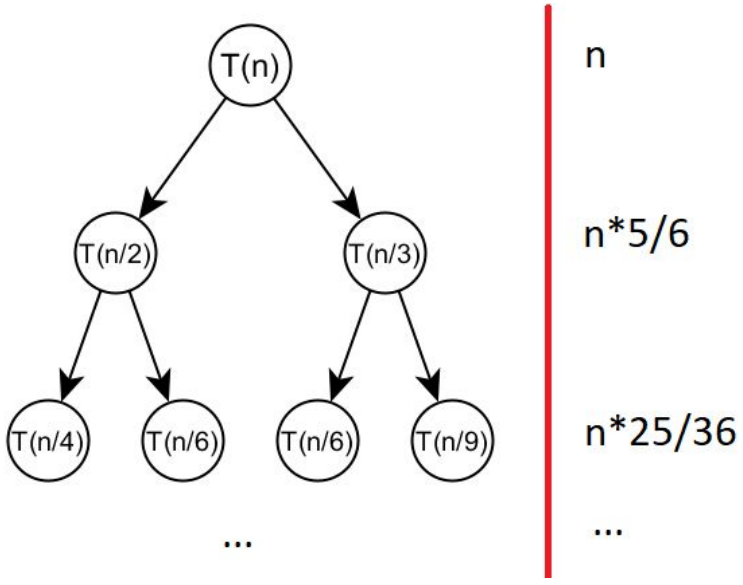
$= n \log n \log_3 - \log 3 \, (1 + \log_3 n) \, (\log_3 n)/2 \sim n \log^2 n - \log^2 n \sim n \log^2 n$

Δηλαδή, έχουμε τελική πολυπλοκότητα **Θ(n $\log^2$n)**.

3.T(n) = 4T(n/3) + nlogn

Είναι a=4, b=3, $n^{\log_3 4} > n$, και άρα σίγουρα ασυμπτωτικά μεγαλύτερο από το f(n) = nlogn.
Οπότε, από Master Theorem, T(n) = **Θ($n^{\log_3 4}$)**.

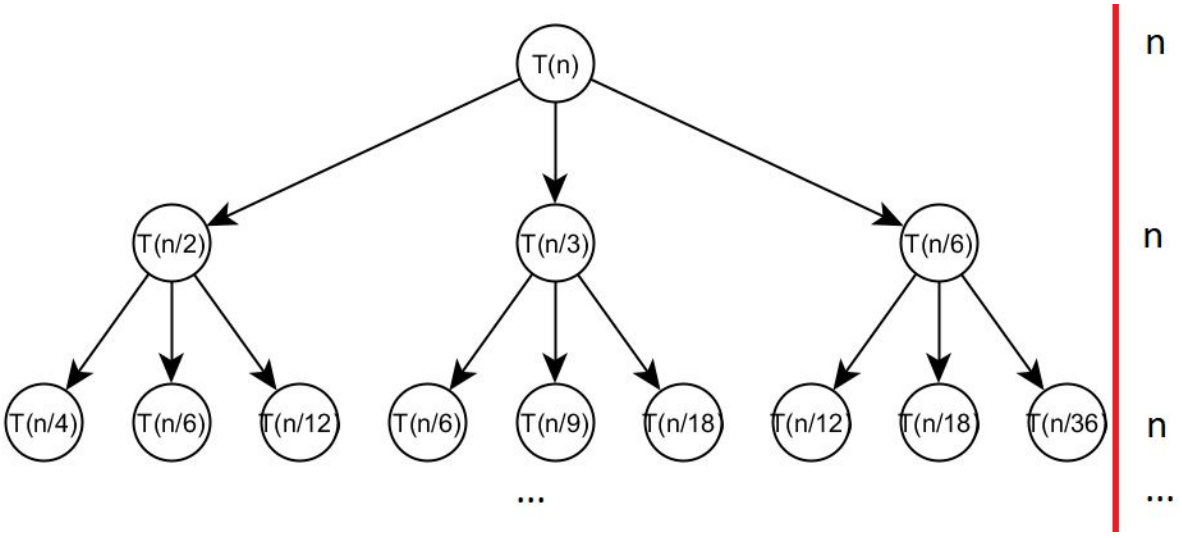
4.T(n) = T(n/2) + T(n/3) + n



Έστω h_1 το ύψος του κοντύτερου παρακλαδιού. Εδώ είναι το δεξιότερο παρακλάδι με $h_1 = \log_3 n$, αλλά μικρή σημασία έχει.
Έστω h_2 το ύψος του μακρύτερου παρακλαδιού (και άρα του δένδρου). Εδώ είναι το αριστερότερο παρακλάδι με $h_2 = \log_2 n$, αλλά μικρή σημασία έχει.

Συνολικά, έχουμε πολυπλοκότητα T(n) με
$\sum_{i=0}^{h_1}(5/6)^i\, n \;<\; T(n) \;<\; \sum_{i=0}^{h_2}(5/6)^i\, n \;\Leftrightarrow\; 6\,n\,(1-(\frac{5}{6})^{h1+1}) \;<\; T(n) \;<\; 6\,n\,(1-(\frac{5}{6})^{h2+1})$ και άρα το T(n) φράσσεται και από τις δύο μεριές από το n (η συνάρτηση $6n*(1-(5/6)^h)$ έχει πολυπλοκότητα Θ(n) ανεξαρτήτως του h).
Συνεπώς, **T(n) = Θ(n)**.

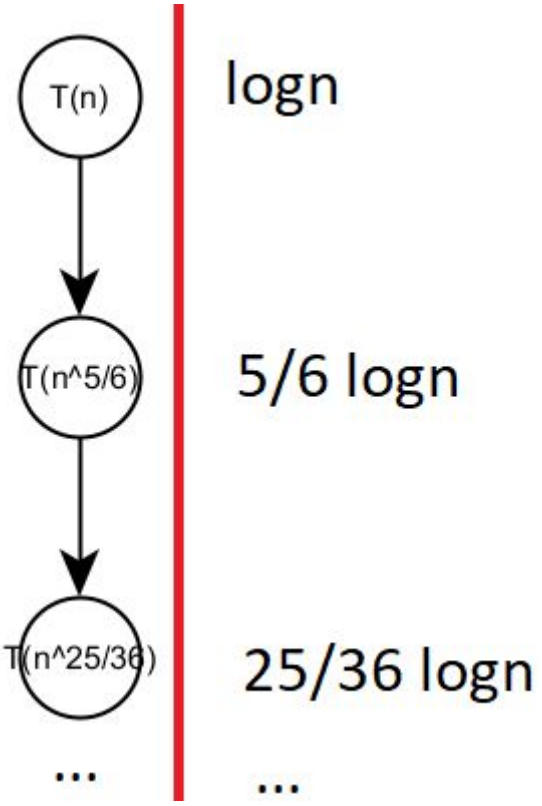
5.T(n) = T(n/2) + T(n/3) + T(n/6) + n



Έστω h_1 το ύψος του κοντύτερου παρακλαδιού. Εδώ είναι το δεξιότερο παρακλάδι με $h_1 = \log_6 n$.
Έστω h_2 το ύψος του μακρύτερου παρακλαδιού (και άρα του δένδρου). Εδώ είναι το αριστερότερο παρακλάδι με $h_2 = \log_2 n$.

Συνολικά, έχουμε πολυπλοκότητα T(n) με
$n \log_6 n < T(n) < n \log_2 n$ και άρα το T(n) φράσσεται και από τις δύο μεριές από το nlogn.
Συνεπώς, **T(n) = Θ(nlogn)**.

6.T(n) = T($n^{5/6}$) + Θ(logn)



Έστω h το ύψος του δέντρου. Τότε έχουμε πολυπλοκότητα

$$\sum_{i=0}^{h}(\tfrac{5}{6})^i \, logn \;=\; logn \sum_{i=0}^{h}(\tfrac{5}{6})^i \, l \;=\; 6\, logn\,(1-(\tfrac{5}{6})^{h+1})$$

Για h = 0, δηλαδή με μηδενικό ύψος δένδρου προκύπτει πολυπλοκότητα logn.
Για h→ ∞, δηλαδή με άπειρο ύψος δένδρου προκύπτει πολυπλοκότητα logn.
Επομένως, το T(n) με ύψος δένδρου h, όπου 0 ≤ h <∞, φράσσεται κι από τις δύο μεριές από το logn.
Συνεπώς, **T(n) = Θ (logn)**.

7.T(n) = T(n/4) + $\sqrt{n}$

Είναι a=1, b=4, $n^{\log_4 1} = n^0 = 1$, το οποίο είναι ασυμπτωτικά μικρότερο από το $f(n) = n^{1/2}$.
Επίσης, a f(n/b) = $\sqrt{n/4} \;=\; \sqrt{n}\,/2 \;<\sqrt{n} = f(n)$ **.**
Οπότε, από Master Theorem, **T(n) = Θ(** $\sqrt{n}$ **)**.

Άσκηση 2

2α.1) Έστω έχουμε πίνακα A με n στοιχεία (όπου n δύναμη του 2) και δοθέν k (όπου n πολλαπλάσιο του k).

Αλγόριθμος

Προκειμένου να ταξινομήσουμε τον A κατά k μέρη ακολουθούμε την εξής διαδικασία.

ι. Κάνουμε quickselect για να βρούμε τη διάμεσο, εννοώντας το στοιχείο που θα ήταν στη θέση n/2 εάν ο πίνακας ήταν ταξινομημένος.

ιι. Κάνουμε partition στο n με pivot τη διάμεσο. Συγκεκριμένα, θα χρησιμοποιήσουμε το Lomuto partition scheme, το οποίο εγγυάται πως στο τέλος του βήματος, το pivot θα βρίσκεται στη σωστή θέση, με όλα τα στοιχεία που είναι μικρότερα από το pivot να είναι στο Left και όλα τα στοιχεία που είναι μεγαλύτερα να είναι στο Right.

Επειδή το pivot είναι διάμεσος (από επιλογή) και χρησιμοποιήσαμε το Lomuto partition scheme, τα Left και Right είναι ισοπληθή (αν συμπεριλάβουμε το pivot στο Left) και έχουν μέγεθος n/2.

ιιι. Επαναλαμβάνουμε τη διαδικασία από την αρχή αναδρομικά στα διαστήματα Left και Right. Όμως, εάν το επιλεγμένο διάστημα έχει μήκος n/k, επιστρέφουμε.

Πολυπλοκότητα

Το βήμα (ι) έχει πολυπλοκότητα Θ(n) καθώς αυτή είναι η πολυπλοκότητα της quickselect. Μπορούμε να βασιστούμε στην τυχαία quickselect ή να χρησιμοποιήσουμε τη μέθοδο με τους medians των medians.

Το βήμα (ιι) έχει πολυπλοκότητα n καθώς για κάθε στοιχείο κάνουμε ακριβώς μια σύγκριση με το pivot.

Είναι $T(n) = 2T(n/2) + Θ(n)$, όμως σταματάμε στο $T(n/k)$ (και όχι στο $T(1)$).

Συνεπώς, προκύπτει δένδρο αναδρομής με ύψος h όπου $n/2^h = n/k \Leftrightarrow h=\log k$.

Σε κάθε επίπεδο, έχουμε αθροιστικό κόστος cn.

Τελικά, προκυπτει πολυπλοκότητα $Θ(n \log k)$.

Ορθότητα

Σε κάθε iteration χωρίζουμε το διάστημα σε δύο ισομερή υποδιαστήματα. Όλα τα στοιχεία του Left είναι μικρότερα (ή ίσα) από όλα του Right. Φυσικά, όλα τα υποδιαστήματα που θα προκύψουν αναδρομικά στο Left θα έχουν στοιχεία μικρότερα (ή ίσα) από όλα τα στοιχεία που θα προκύψουν αναδρομικά στο Right.

Λοιπόν, όταν φτάσουμε σε k διαστήματα μεγέθους n/k, τα n/k στοιχεία του πρώτου μέρους θα είναι μικρότερα (ή ίσα) από τα n/k στοιχεία του δεύτερου μέρους τα οποία θα είναι μικρότερα (ή ίσα) από τα n/k στοιχεία του τρίτου μέρους κοκ.

Απόδειξη κάτω φράγματος στην πολυπλοκότητα

Η απόδειξη ακολουθεί τη λογική του κάτω φράγματος στην πολυπλοκότητα συγκριτικών αλγορίθμων ταξινόμησης.

Έστω τυχαία ακολουθία στοιχείων $a_1,a_2,...,a_n$ και έστω συγκριτικός αλγόριθμος που ταξινομεί την ακολουθία κάτα k μέρη.

Από το αντίστοιχο δένδρο αποφάσεων, θα πρέπει να μπορούμε να καταλήξουμε σε όλα τα δυνατά ενδεχόμενα για τις τιμές των $a_1,a_2,...,a_n$.
Ας παίξουμε λίγο με συνδυαστική. Θεωρούμε πως έχουμε k αριθμημένα μέρη όπου το καθένα θα περιέχει n/k στοιχεία. Παίρνοντας τα με τη σειρά, για το πρώτο μέρος έχουμε $\binom{n}{n/k}$ επιλογές για το πως θα το γεμίσουμε (καθώς δε μας νοιάζει η εσωτερική διάταξη).
για το δεύτερο μέρος θα έχουμε $\binom{n-n/k}{n/k}$ αφού γεμίσαμε το πρώτο μέρος με n/k στοιχεία. Λοιπόν, για το i-οστό μέρος θα έχουμε $\binom{n-i\,n/k}{n/k}$ ενδεχόμενα. Πολλαπλασιάζοντας το πλήθος ενδεχομένων για το κάθε μέρος, προκύπτει το συνολικό πλήθος $\frac{n!\,(n-n/k)!\,\ldots\,(n-(k-1)n/k)!\,(n-n)!}{(n/k)!^n\;\;(n-n/k)!\,(n-2n/k)!\,\ldots\,(n-(k-1)n/k)!} = \frac{n!}{(n/k)!^k}$ το οποίο είναι και το πλήθος των φύλλων του δένδρου αποφάσεων.
Συνεπώς, το ύψος του δένδρου για οποιονδήποτε τέτοιο αλγόριθμο είναι:
$$h \geq log(\frac{n!}{(n/k)!^k}) = log(n!) - k\,log((n/k)!) = \Omega\,(nlogn - k\,n/k\,log\,(n/k)\,) = \Omega\,(n\,logk)$$

<u>2α.2)</u> Εάν ένας πίνακας A με n στοιχεία είναι ταξινομημένος κατά k μέρη, άρα τα μέρη είναι μεταξύ τους ταξινομημένα. Όλα τα στοιχεία του πρώτου μέρους είναι μικρότερα από του δεύτερου, τα οποία είναι μικρότερα από του τρίτου κοκ.
Συνεπώς, αρκεί να ταξινομήσουμε εσωτερικά το κάθε τέτοιο μέρος.
Κάθε μέρος θα έχει n/k στοιχεία, οπότε η ταξινόμηση για το κάθε μέρος θα έχει πολυπλοκότητα Θ[(n/k) log(n/k)] και εφόσον έχουμε k τέτοια μέρη, συνολικά προκύπτει
Θ[k (n/k) log(n/k)] = **Θ[n log(n/k)]**.

Το κάτω φράγμα πολυπλοκότητας προκύπτει Ω(n log(n/k)) καθώς πρέπει σίγουρα να ταξινομήσουμε εσωτερικά ξεχωριστά το κάθε μέρος. Θυμίζουμε πως το κάτω φράγμα συγκριτικών αλγορίθμων ταξινόμησης n στοιχείων είναι Ω(n logn) (προκύπτει αποδεικνύοντας πως το αντίστοιχο δένδρο αποφάσεων έχει n! φύλλα και άρα ύψος h ≥ Ω(n logn).

<u>2β)</u> Θα εκτελέσουμε μια παραλλαγή της quicksort σε διπλά συνδεδεμένη λίστα.

<u>Αλγόριθμος</u>
ι. Επιλέγουμε ένα τυχαίο pivot. Χάριν απλότητας ας επιλέξουμε το πρώτο στοιχείο της λίστας.
ιι. Κάνουμε partition τη λίστα ως εξής: Διατρέχουμε τα στοιχεία ξεκινώντας ακριβώς μετά το pivot. Εάν το i στοιχείο είναι μικρότερο από το pivot, τοποθετείται ακριβώς πριν το pivot. Εάν είναι μεγαλύτερο, τοποθετείται ακριβώς μετά το pivot. Εάν είναι ίσο αυξάνουμε ένα μετρητή πλήθους duplicate που έχουμε στο pivot. Τα στοιχεία αριστερά από το pivot θεωρούνται το Left partition και τα δεξιά το Right. Για ευκολία, ας κρατάμε δείκτες που θα δείχνουν την αρχή και το τέλος του κάθε partition.
ιιι. Κάνουμε αναδρομικά αυτήν την παραλλαγή της quicksort στα Left και Right. Επιστρέφουμε όταν μηδενιστεί το μήκος του διαστήματος που πρέπει να κάνουμε partition.

Εάν θέλουμε να περάσουμε τα δεδομένα μας σε πίνακα, τα περνάμε με τη σειρά από την αρχή της τελικής μας λίστας. Απλά, εκεί που έχουμε μετρήσει k duplicate, θα περάσουμε k στοιχεία με την ίδια τιμή, προτού συνεχίσουμε στο επόμενο (διαφορετικό) στοιχείο.

Πολυπλοκότητα

Σε κάθε επίπεδο του αναδρομικού δένδρου έχουμε αθροιστικό κόστος n καθώς όλα (ή σχεδόν όλα) τα στοιχεία της λίστα θα συγκριθούν με κάποιο pivot. Στο τέλος, όλα τα διαφορετικά στοιχεία του πίνακα θα έχουν γίνει pivot (και μόνο αυτά). Τα duplicate του εκάστοτε pivot διαγράφονται και προσμετρώνται.

Έτσι, θεωρώντας πως σε κάθε επίπεδο, το κάθε διάστημα σπάει σε δύο υποδιαστήματα, θα χρειαστούμε $\text{loglog}^d n$ επίπεδα για να φτάσουμε μοναδιαία διαστήματα, καθώς το δένδρο έχει ύψος h, όπου $\log^d n/2^h = 1 \Leftrightarrow h = \text{loglog}^d n$ (τα duplicate θα διαγραφούν στην πορεία χωρίς να επηρεάσουν το πλήθος των επιπέδων).

Συνεπώς, προκύπτει συνολική πολυπλοκότητα $\Theta(n \text{ loglog}^d n)$ = **$\Theta(n \text{ loglogn})$**.

Σημείωση: Θεωρήσαμε πως για κάθε pivot, το διάστημα χωρίζεται σε δύο ισοπληθή υποδιαστήματα. Παρότι αυτό δεν ισχύει πάντα, για τη μέση περίπτωση προκύπτει πιθανοτικά η ίδια πολυπλοκότητα.

Ορθότητα

Η ορθότητα του αλγορίθμου είναι άμεση απόρροια της ορθότητας της quicksort.

Κάτω φράγμα πολυπλοκότητας

Στη γενική περίπτωση συγκριτικών αλγορίθμων ταξινόμησης n στοιχείων, θα πρέπει μέσα από το δένδρο αποφάσεων να μπορεί να ικανοποιηθεί οποιαδήποτε τελική σειρά στοιχείων.

Λοιπόν, έχουμε n! ενδεχόμενα από συνδυαστική (δυνατές μεταθέσεις n αντικειμένων). Ουσιαστικά, θεωρώντας πως γεμίζουμε τις τελικές θέσεις με τη σειρά, έχουμε n επιλογές για την πρώτη θέση, (n-1) για τη δεύτερη, κοκ. Άρα το δένδρο αποφάσεων έχει τουλάχιστον n! φύλλα και, συνεπώς, ύψος $h \geq \Omega(n \log n)$.

Ωστόσο, εδώ, στην τελική ταξινόμηση των στοιχείων όλα τα duplicate στοιχεία θα είναι ομαδοποιημένα και δε θα υπάρχει διάκριση ανάμεσά τους. Συνεπώς, αρκεί να υπολογίσουμε τις μεταθέσεις μόνο των διαφορετικών στοιχείων, οι οποίες είναι $(\log^d n)!$ και όχι n! . Συνεπώς, δεν ισχύει το φράγμα του $\Omega(n \log n)$ σε αυτήν την περίπτωση. Προκύπτει $h \geq \Omega (\log^d n \log \log n)$.

Άσκηση 3

3α) Έστω δύο ταξινομημένοι πίνακες A_1 (με n_1 στοιχεία) και A_2 (με n_2 στοιχεία). Χωρίς βλάβη της γενικότητας, υποθέτουμε πως είναι ταξινομημένοι σε αύξουσα σειρά από τα αριστερά προς τα δεξιά.

Αλγόριθμος

ι. Έχουμε δύο "δείκτες" δ_1, δ_2 (έναν σε κάθε πίνακα). Ο δ_1 δείχνει στο πρώτο στοιχείο του A_1 και ο δ_2 στο πρώτο του A_2. Συγκρίνουμε τα δύο στοιχεία μεταξύ τους και βρίσκουμε το μεγαλύτερο "max" και το μικρότερο "min" εξ' αυτών. Εκτελούμε την πράξη dif = max - min. Θέτουμε minDif = dif.

ιι. Σε κάθε iteration, μετακινούμε δεξιά το δείκτη του min κατά μία θέση. Βρίσκουμε το νέο dif και ανν είναι μικρότερο από το minDif, θέτουμε minDif = dif. Τερματίζουμε, όταν με την μετακίνηση του δείκτη βγούμε εκτός ορίων σε έναν από τους δύο πίνακες.

Πολυπλοκότητα

Σε κάθε iteration κάνουμε δύο συγκρίσεις και μια απλή αφαίρεση (άρα Θ(1) ανά βήμα). Επίσης, σε κάθε iteration, κινούμε ακριβώς ένα δείκτη ακριβώς μία θέση δεξιά. Τερματίζουμε όταν ο πρώτος δείκτης ξεπεράσει το n_1 ή ο δεύτερος το n_2. Συνολικά θα κάνουμε το πολύ n_1+n_2 iteration.

Προκύπτει συνολική πολυπλοκότητα $\mathbf{\Theta(n_1+n_2)}$.

Ορθότητα

Έστω οι δύο πίνακες Α, Β.

Καταρχάς, αναφέρουμε πως σε κάθε iteration κινείται ο ένας δείκτης μία θέση δεξιά (σταθερή κατεύθυνση) και τερματίζουμε όταν ένας δείκτης ξεπεράσει τα όρια του αντίστοιχου πίνακα. Επειδή οι πίνακες έχουν πεπερασμένο πλήθος στοιχείων, ο αλγόριθμος σίγουρα θα τερματίσει.

Έστω χωρίς βλάβη της γενικότητας πως το μέγιστο στοιχείο του Α είναι μεγαλύτερο ή ίσο από το μέγιστο στοιχείο του Β. Τότε, ο δείκτης δ_2 θα περάσει από όλα τα στοιχεία του Β, δηλαδή θα γίνουν σίγουρα κάποια στιγμή min ή max (όπως ορίστηκαν τα "min", "max" νωρίτερα). Όμοια, για όλα τα στοιχεία του Α που δεν ξεπερνούν το μέγιστο στοιχείο του Β. Επιπλέον, ο δείκτης δ_1 θα περάσει και από το πρώτο στοιχείο του Α που είναι μεγαλύτερο του Β. Τα υπόλοιπα στοιχεία του Α, προφανώς δε χρειάζεται να εξεταστούν καθώς θα δώσουν μεγαλύτερες διαφορές από το πρώτο στοιχείο του Α υπό συνθήκη να είναι μεγαλύτερο του μεγίστου του Β.

Ο αλγόριθμος μας παίρνει μόνο διαφορές ανάμεσα σε 2 στοιχεία, ένα από τον Α και ένα από το Β. Συνεπώς, το μόνο κομμάτι που πρέπει να εξετάσουμε εδώ, είναι αν η διαφορά που θα διαλέξουμε εν τέλει είναι πράγματι η ελάχιστη δυνατή.

Θα αποδείξουμε την ορθότητα του αλγορίθμου μας.
- Έστω δύο πίνακες A, B. Έστω πως υπάρχουν στοιχεία a ϵ A, b ϵ B (χωρίς βλάβη της γενικότητας) τέτοια ώστε a - b = minDif, η βέλτιστη διαφορά .

- Από τις συνθήκες του προβλήματος, προκύπτει πως το b είναι το μεγαλύτερο δυνατό στοιχείο του B υπό συνθήκη να είναι μικρότερο από το a. Επίσης, το a είναι το μικρότερο δυνατό στοιχείο του A υπό συνθήκη να είναι μεγαλύτερο από το b.

- Κάποια στιγμή στον αλγόριθμό μας, ο δείκτης δ_1 θα φτάσει το a. Έστω το a γίνεται κάποια στιγμή "max" στον αλγόριθμό μας. Τότε, ο δείκτης δ_1 θα μείνει ακίνητος στο a και ο δείκτης θα κινείται στον B, μέχρις ότου να φτάσει σε στοιχείο του B μεγαλύτερο από τον a. Όμως, αναγκαία, στο iteration πριν γίνει αυτό, θα έχουμε λάβει ως min το μεγαλύτερο δυνατό στοιχείο του B το οποίο είναι μικρότερο από το a. Άρα, θα βρούμε σίγουρα τη διαφορά a - b.

- Εναλλακτικά, ας υποθέσουμε πως όταν ο δ_1 φτάνει στο a, αυτό τίθεται ως min (οπότε δε γίνεται καθόλου max κατά τη διάρκεια του αλγορίθμου). Αυτό όμως, δεν είναι εφικτό.
Πράγματι, προκειμένου το a να γίνει min, πρέπει ο δ_2 να έχει ήδη ξεπεράσει το b (και άρα κάποια στιγμή να το έφτασε).
Γνωρίζουμε πως ο εκάστοτε δείκτης θα κινηθεί ανν δείχνει στο min (όχι στο max). Αυτό σημαίνει πως για να ξεπεράστηκε το b, πρέπει το b να είχε γίνει min. Όμως αυτό δε γίνεται, καθώς το a είναι το ελάχιστο δυνατό στοιχείο του A το οποίο είναι μεγαλύτερο από το b και ο δ_1 έφτανε τότε στο a, το a θα γινόταν max. Καταλήγουμε σε άτοπο.

3β) Έστω m ταξινομημένοι πίνακες $A_1,...,A_m$ με αντίστοιχο πλήθος στοιχείων $n_1,...,n_m$.

Αλγόριθμος
ι. Έχουμε m δείκτες $\delta_1,...,\delta_m$ (έναν σε κάθε πίνακα). Αρχικά, ο κάθε δείκτης δείχνει στο πρώτο στοιχείο του αντίστοιχου πίνακα. Βρίσκουμε το μέγιστο "max" και ελάχιστο "min" ανάμεσα στα m στοιχεία που δείχνουν οι δείκτες. Εκτελούμε την πράξη dif = max - min. Θέτουμε minDif ←
dif.
ιι. Σε κάθε iteration, αυξάνουμε το δείκτη του min κατά 1. Βρίσκουμε το νέο max, min και έπειτα το dif και ανν είναι μικρότερο από το minDif, θέτουμε minDif ← dif. Τερματίζουμε, όταν με την αύξηση του δείκτη βγούμε εκτός ορίων σε έναν από τους m πίνακες.

Πολυπλοκότητα
Σε κάθε iteration βρίσκουμε το ελάχιστο και το μέγιστο (ανάμεσα στα m στοιχεία), κάνουμε μια απλή αφαίρεση και μία σύγκριση (άρα $\Theta(2m+1)=\Theta(m)$ ανά βήμα). Επίσης, σε κάθε iteration, αυξάνουμε ακριβώς ένα δείκτη ακριβώς μία μονάδα. Τερματίζουμε όταν κάποιος δείκτης ξεπεράσει το αντίστοιχο όριο του πίνακα. Συνολικά θα κάνουμε το πολύ $n_1+n_2+...+n_m = N$ το πλήθος iteration.
Προκύπτει συνολική πολυπλοκότητα **$\Theta(m N)$**.

Ορθότητα

Αντίστοιχα με το (α) ερώτημα, κινούμε πάντα το δείκτη του στοιχείου που ορίζει το κάτω όριο του εύρους.

Για δεδομένο max a ϵ A_1, ψάχνουμε το min ανάμεσα στα μέγιστα δυνατά στοιχεία από τους πίνακες A_2,...,A_m υπό συνθήκη να είναι μικρότερα του max (και τα στοιχεία των υπολοίπων πινάκων να παραμένουν εντός του εύρους).

Καθώς κινούμαστε μία θέση τη φορά, για δεδομένο max, θα περάσουμε σίγουρα από το ιδανικό min (και τα υπόλοιπα στοιχεία να παραμένουν εντός του εύρους).

3γ) Η ζητούμενη λειτουργία είναι η εύρεση του ελαχίστου και μεγίστου σε κάθε iteration. Η διαδικασία μοιάζει πλεοναστική καθώς αγνοούμε τις πληροφορίες που έχουμε για τα μεγέθη από το προηγούμενο iteration.

Αλγόριθμος

ι. Στο πρώτο iteration, τοποθετούμε τα m στοιχεία σε ένα σωρό (ελαχίστου) με πολυπλοκότητα O(m) και παράλληλα κρατάμε την τιμή max του μεγίστου στοιχείου. Το ελάχιστο στοιχείο είναι η ρίζα του σωρού. Εκτελούμε την πράξη dif = max - min. Θέτουμε minDif ← dif.

ιι. Σε κάθε iteration, αυξάνουμε το δείκτη του min κατά 1. Διαγράφουμε το ελάχιστο στοιχείο (τη ρίζα του σωρού) και εισάγουμε το νέο στοιχείο στο σωρό. Παράλληλα με την εισαγωγή του νέου στοιχείου a, ελέγχουμε εάν είναι μεγαλύτερο από το max και αν είναι, θέτουμε max ← a.

Πολυπλοκότητα

Στο πρώτο iteration δημιουργούμε σωρό (ελαχίστου) m στοιχείων (άρα έχουμε πολυπλοκότητα m) και παράλληλα έχουμε μια εύρεση μεγίστου (έστω πολυπλοκότητα m).

Έπειτα, σε καθένα από τα υπόλοιπα iteration, διαγράφουμε τη ρίζα του σωρού, φυσικά διατηρώντας το σωρό ως δομή (με πολυπλοκότητα logm) και εισάγουμε νέο στοιχείο (με πολυπλοκότητα logm). Για το νέο στοιχείο κάνουμε μια σύγκριση με το max (Θ(1)). Επίσης, κάνουμε μια απλή αφαίρεση και μια σύγκριση για το diff.

Όπως και πριν, κάνουμε N iterations.

Συνολικά, προκύπτει πολυπλοκότητα **Θ(m + N logm)**.

(Ζήτω οι σωροί!)

Ορθότητα

Η λογική του αλγορίθμου είναι ίδια με το ερώτημα (β).

Μονάχα η υλοποίηση αλλάζει λίγο.

Άσκηση 4

4α) Έστω ότι έχουμε n =1.000.000 πανομοιότυπες φιάλες, μία εκ των οποίων περιέχει το πανίσχυρο μαγικό φίλτρο.

Αλγόριθμος

ι. Θέτουμε σε κάθε φιάλη ένα μοναδικό δυαδικό αριθμό. Θα χρειαστούμε $\log_2 n = 20$ bits. (Προφανώς, θα χρησιμοποιήσουμε τον ελάχιστο απαιτούμενο πλήθος bits).

ιι. Χρησιμοποιούμε $\log_2 n = 20$ εθελοντές. Αντιστοιχίζουμε καθένα από τους εθελοντές σε ένα bit (1 προς 1 αντιστοίχιση). Έτσι θα έχουμε το εθελοντή ε_0, τον ε_1 κοκ.

ιιι. Ένας εθελοντής θα δοκιμάσει από την εκάστοτε φιάλη ανν στον αντίστοιχο δυαδικό αριθμό της φιάλης, το bit που αντιστοιχεί στον εθελοντή είναι 1. Πχ από τη φιάλη 0000 0000 0001 0010 0101, θα δοκιμάσουν ακριβώς οι εθελοντές ε_0, ε_2, ε_5, ε_8.

ιν. Μετά την πάροδο 24 ωρών, θα ελέγξουμε ποιοι εθελοντές απέκτησαν υπερδυνάμεις (και ποιοι έμειναν παραπονεμένοι). Συμβολίζοντας με 1 την απόκτηση υπερδυνάμεων και με 0 το παράπονο, θα προκύψει ένας δυαδικός αριθμός $\varepsilon_{19}\varepsilon_{18}...\varepsilon_1\varepsilon_0$. Ο αριθμός αυτός θα ταυτίζεται με τον αριθμό της φιάλης που περιέχει το μαγικό φίλτρο.

Ορθότητα αλγορίθμου

Καταρχάς, κάθε αποτέλεσμα είναι μοναδικό. Δεν μπορούν να προκύψουν περισσότεροι από ένας δυαδικοί αριθμοί τη φορά καθώς ο κάθε εθελοντής θα δώσει σίγουρα είτε 1 ή 0 (xor) για το αντίστοιχο bit.

Θα αποδείξουμε πως δεν μπορούμε να καταλήξουμε σε λάθος φιάλη (με απαγωγή σε άτοπο).

- Έστω η σωστή φιάλη “a” έχει το bit $\varepsilon_\kappa = 1$ ενώ το αποτέλεσμά μας “b” έχει το bit $\varepsilon_\kappa = 0$ (για κάποιο k, $0 \leq k \leq 19$).
- Αυτό σημαίνει πως ο εθελοντής ε_κ δοκίμασε από την a (στο βήμα ιι του αλγορίθμου), καθώς το bit που του αντιστοιχεί είναι 1 στην a.
- Όμως, ο ίδιος εθελοντής δεν απέκτησε υπερδυνάμεις, καθώς δεν έθεσε 1 το αντίστοιχο bit του αποτελέσματος (στο βήμα ιν).
- Προκύπτει άτοπο, καθώς ο εθελοντής δοκίμασε από το μαγικό φίλτρο, αλλά δεν απέκτησε υπερδυνάμεις.

Αντίστοιχα, αν θεωρήσουμε πως η σωστή φιάλη “a” έχει το bit $\varepsilon_\kappa = 0$ ενώ το αποτέλεσμά μας “b” έχει το bit $\varepsilon_\kappa = 1$ (για κάποιο k, $0 \leq k \leq 19$), καταλήγουμε σε άτοπο, καθώς ο εθελοντής ε_κ δε δοκίμασε από το μαγικό φίλτρο, αλλά απέκτησε υπερδυνάμεις (βέβαια, μπορεί απλά να πίστεψε στον εαυτό του, the power was inside him all along…).

Τελικά, κάθε bit του αποτελέσματός μας, ταυτίζεται με το αντίστοιχο bit της φιάλης με το μαγικό φίλτρο.

Κάτω φράγμα πολυπλοκότητας

Έχουμε ένα νοητό δένδρο παρόμοιο με τα δένδρα αποφάσεων στους συγκριτικούς αλγορίθμους ταξινόμησης.

Ο μόνος τρόπος για να ξεχωρίσουμε τις φιάλες είναι από την επίδρασή τους σε κάποιον εθελοντή. Για αυτό, λοιπόν, μπορούμε να θεωρήσουμε τον έλεγχο της επίδρασης σε έναν έθελοντή ως ένα επίπεδο στο δένδρο μας. Καθώς η επίδραση είναι μανιχαϊστική (ο εθελοντής είτε θα αποκτήσει υπερδυνάμεις είτε όχι), προκύπτει δυαδικό δένδρο.
Σε κάθε επίπεδο, εξετάζουμε αν ο εκάστοτε εθελοντής απέκτησε υπερδυνάμεις. Εάν απέκτησε υπερδυνάμεις, συνεχίζουμε με τις φιάλες τις οποίες δοκίμασε ο εθελοντής, διαφορετικά, συνεχίζουμε με τις φιάλες τις οποίες δε δοκίμασε ο εθελοντής.
Το δένδρο οφείλει να μπορεί να καλύψει όλα τα n = 1.000.000 ενδεχόμενα (εννοώντας, ότι οποιαδήποτε από τις 1.000.000 φιάλες μπορεί να είναι η σωστή). Άρα, θα έχει n φύλλα και ύψος **h ≥ Ω(logn)**.

<u>Εναλλακτική υλοποίηση</u>
Παραπάνω χρησιμοποίησαμε μια μέθοδο η οποία βασίζεται στην αναπαράσταση του προβλήματος στο δυαδικό σύστημα. Η μέθοδος λειτουργεί, γιατί το δυαδικό σύστημα είναι είναι “αποδοτικό”, με την έννοια ότι δεν έχει πλεονασμούς ή αντιφάσεις.
Μια εναλλακτική υλοποίηση (αν και ουσιαστικά ίδια), λίγο πιο κοντά στα δεδομένα του μαθήματος είναι η εξής:
i. Θέτουμε τις 1.000.000 φιάλες σε μία σειρά.
ii. Τοποθετούμε έναν εθελοντή στη μέση του διαστήματος. Ο εθελοντής θα δοκιμάσει από όλες τις φιάλες στα δεξιά του, αλλά από καμία στα αριστερά του. Με αυτόν τον τρόπο, δημιουργούνται 2 υποδιαστήματα.
iii. Ο επόμενος εθελοντής θα δοκιμάσει μόνο από τις δεξιά μισές φιάλες του κάθε υποδιαστήματος. Επαναλαμβάνουμε το βήμα (iii) μέχρι να μηδενιστεί το μήκος των διαστημάτων.

Με αυτόν τον τρόπο δημιουργείται ένα νοητό δένδρο αποφάσεων όπου κάθε επίπεδο του δένδρου αντιστοιχεί σε μοναδικό εθελοντή. Τα φύλλα του δένδρου πρέπει να καλύπτουν όλα τα δυνατά ενδεχόμενα (δηλαδή να μπορούν να καταλήξουν σε οποιαδήποτε φιάλη πως έχει το μαγικό φίλτρο) και άρα χρειαζόμστε τουλάχιστον n = 1.000.000 φύλλα. Από αυτό προκύπτει πως **h ≥ Ω(logn)** (εδώ $h \geq \log_2 n = 20$).
Η εύρεση της σωστής φιάλης θα προκύψει όταν (αφού λάβουμε τα αποτελέσματα) ακολουθήσουμε το δένδρο αποφάσεων εξετάζοντας εάν ο εθελοντής που αντιστοιχεί στο εκάστοτε επίπεδο απέκτησε υπερδυνάμεις ή όχι.

Για να γίνει πιο κατανοητή αυτή η μέθοδος, έστω ένα παράδειγμα με n=32 φιάλες.
Συμβολίζοντας με 1 τη δοκιμή και 0 τη μη-δοκιμή, ο πρώτος εθελοντής θα δοκιμάσει
0000 0000 0000 0000 | 1111 1111 1111 1111
ο δεύτερος θα δοκιμάσει:
0000 0000 | 1111 1111 | 0000 0000 | 1111 1111
ο τρίτος:
0000 | 1111 | 0000 | 1111 | 0000 | 1111 | 0000 | 1111
κοκ

Σημείωση: Είπαμε πως οι δύο υλοποιήσεις είναι ουσιαστικά ίδιες, καθώς με αυτή τη μέθοδο δημιουργούμε κατά μία έννοια το δυαδικό σύστημα.

4β) Σημείωση: Η κάτωθι απάντηση προέκυψε από τη διάλεξη του κου Φωτάκη στο αμφιθέατρο. Ομολογουμένως, προσωπικά δεν ήμουν καν κοντά στη σωστή λύση, παρότι, εκ των υστέρων, μοιάζει απλούστατη.

Έστω θέλουμε να διανύσουμε τις επιμέρους αποστάσεις d_1, d_2,...,d_n με τη σειρά μέσα σε k μέρες (οπότε θέτοντας k-1 παύσεις στην ακολουθία των αποστάσεων).
Στόχος μας είναι η ελαχιστοποίηση της μέγιστης ημερήσιας απόστασης.
Έστω $d_1+d_2+...+d_n = \Sigma d$.

Εάν έχουμε μια δεδομένη ακέραια τιμή m μπορούμε να ελέγξουμε αν μπορεί να είναι μέγιστη ημερήσια απόσταση σε γραμμικό χρόνο O(n) (ίσως O(n+k), αλλά εφόσον n>k, προκύπτει O(n) όπως και να 'χει). Απλά, αθροίζουμε με τη σειρά τα d_1, d_2 μονάχα εφόσον το άθροισμα παραμένει κάτω από την τιμή m.
Αν το άθροισμα ξεπεράσει το m, ακυρώνουμε την τελευταία πρόσθεση και θέτουμε μια παύση σε εκείνο το σημείο και ξεκινάμε νέο άθροισμα από το επόμενο d.
Επαναλαμβάνουμε τη διαδικασία, μέχρι είτε να φτάσουμε επιτυχώς στο τέλος της ακολουθίας (οπότε και το m είναι αποδεκτό) ή να μας τελειώσουν οι k-1 παύσεις (οπότε το m απορρίπτεται).

Προφανώς, εάν κάποιο m είναι αποδεκτό, τότε και όλες οι τιμές μεγαλύτερες του m είναι αποδεκτές. Επίσης, εάν κάποιο m απορρίπτεται, τότε και όλες οι τιμές μικρότερες του m απορρίπτονται.
Σημειώνουμε δε πως το m=Σd είναι σίγουρα πάντα αποδεκτό, καθώς μπορούμε απλά να τοποθετήσουμε όλες τις αποστάσεις σε μία μόνο μέρα.

Από τα παραπάνω προκύπτει πως αρκεί να κάνουμε μια δυαδική αναζήτηση με όρια τα 1 και Σd για να βρούμε το ελάχιστο αποδεκτό m.
Σε κάθε iteration, ελέγχουμε σε γραμμικό χρόνο εάν το εκάστοτε m είναι αποδεκτό. Εάν είναι, θέτουμε ως νέο άνω όριο το m, ενώ, εάν δεν είναι, θέτουμε ως νέο κάτω όριο το m. Επαναλαμβάνουμε τη διαδικασία στο εκάστοτε νέο διάστημα μέχρις ότου βρούμε τιμές m, m-1, όπου το m είναι αποδεκτό, ενώ το m-1 απορρίπτεται. Συνεπώς, όταν τα όρια απέχουν μόνο μία μονάδα.

Ουσιαστικά, δημιουργείται ένας νοητός πίνακας (όπου εμείς ψάχνουμε τον πρώτο άσσο) :

m	1	2	...					...	Σd-1	Σd
αποδοχή 1/ απορριψη 0	0	0	0	0	**1**	1	1	1	1	1

Συνολικά προκύπτει **O (n log(Σd))**, καθώς έχουμε κόστος n σε κάθε iteration και χρειαζόμαστε log(Σd) iteration για τη δυαδική αναζήτηση.

Σημείωση: Ένα καλύτερο αρχικό κάτω όριο για τη δυαδική αναζήτηση θα ήταν το Σd/k, το οποίο αν ήταν αποδεκτή τιμή θα ήταν ιδανικό, καθώς προκύπτει αν μπορούμε να μοιράσουμε ισομερώς τις αποστάσεις στις k μέρες. Η ουσία του αλγορίθμου παραμένει ίδια.

Άσκηση 5

5α) Έστω έχουμε multiset S με n θετικούς ακέραιους με άνω φράγμα M.
Επίσης, έστω συνάρτηση $F_s(l) = |\{x \in S : x \leq l\}|$
Θέλουμε να βρούμε το k-οστό μικρότερο στοιχείο του S.

Λοιπόν, πρέπει να βρούμε σημείο l_0 τέτοιο ώστε $l_0 \geq k$ και $l_0-1<k$.
Η λογική θυμίζει τον πίνακα της (4β).

Αλγόριθμος
Κάνουμε δυαδική αναζήτηση με αρχικά όρια 0 και M. Σε κάθε iteration παίρνουμε νέο l και συγκρίνουμε το $F_s(l)$ με το k. Εάν είναι $F_s(l) \geq k$, θέτουμε ως νέο άνω όριο το l, διαφορετικά θέτουμε ως νέο κάτω όριο το l.
Τερματίζουμε τη διαδικασία όταν βρούμε l_0 τέτοιο ώστε $l_0 \geq k$ και $l_0-1<k$, δηλαδή όταν τα όριά μας απέχουν μόνο μία μονάδα.

Πρόκειται για δυαδική αναζήτηση με αρχικά όρια 0 και M, άρα έχουμε πολυπλοκότητα **O(logM)** (θεωρώντας πως η κλήση της συνάρτησης F_s έχει O(1)).

Ορθότητα
Έστω l_0 η τιμή του k-οστού μικρότερου στοιχείου. Αναγκαία, υπάρχουν λιγότερα από k στοιχεία με τιμή μικρότερη του l_0. Επίσης αναγκαία, για κάθε τιμή μεγαλύτερη ή ίση του l_0 υπάρχουν τουλάχιστον k στοιχεία μικρότερα. Ο αλγόριθμος προκύπτει άμεσα.

5β) Έστω πίνακας διαφορετικών θετικών ακεραίων A με μέγιστο στοιχείο M.
Θεωρούμε το πολυσύνολο S που περιέχει όλες τις μη αρνητικές διαφορές ζευγών στοιχείων του A.
Θέλουμε να βρούμε το k-οστό μικρότερο στοιχείο του S.

Η απλούστερη λύση θα ήταν να δημιουργήσουμε τον πίνακα διαφορών ($O(n^2)$) και να βρούμε το k-οστό μικρότερο στοιχείο του πίνακα διαφορών με n^2 στοιχεία ($O(n^2)$). Αυτός ο τρόπος θα είχε συνολική χρονική πολυπλοκότητα $O(n^2)$.

Ωστόσο, μπορούμε να δημιουργήσουμε αλγόριθμο με πολυπλοκότητα O(n*logM).

Αλγόριθμος
ι. Ταξινομούμε τον πίνακα A σε αύξουσα σειρά.
ιι. Υλοποιούμε συνάρτηση F(l), που για δεδομένο l, επιστρέφει το πλήθος των στοιχείων στον πίνακα διαφορών που δεν ξεπερνούν το l.
ιιι. Κάνουμε δυαδική αναζήτηση στο διάστημα 0 εώς M για να βρούμε σημείο l_0 τέτοιο ώστε $l_0 \geq k$ και $l_0-1<k$, όμοια με το (4α). Είναι προφανές, πως εάν το μέγιστο στοιχείο είναι M, η μέγιστη διαφορά είναι μικρότερη του M (καθώς ο A είναι πίνακας θετικών ακεραίων).

Φυσικά, το όλο νόημα αυτού του ερωτήματος είναι η υλοποίηση της συνάρτησης F.

Υλοποίηση της F

Έστω δίνεται τιμή l (0 < l < M).

Έστω δύο στοιχεία του A. Επειδή όλα τα στοιχεία είναι διαφορετικά, το ένα θα είναι μεγαλύτερο από το άλλο. Ονομάζουμε το εκάστοτε μεγαλύτερο big και το άλλο small.

Κάθε διαφορά δ προκύπτει από την πράξη dif = big - small, για κάποιο ζεύγος στοιχείων του πίνακα A.

Έστω f το εκάστοτε συνολικό πλήθος των διαφορών με τιμή μικρότερη ή ίση του l.

ι. Θέτουμε δείκτη δ_s που δείχνει στο πρώτο στοιχείο του πίνακα, έστω s η τιμή του και i_s η θέση του (εδώ $i_s = 1$), και δείκτη δ_b που δείχνει στο δεύτερο στοιχείο, έστω b η τιμή του και i_b η θέση του (εδώ $i_b = 2$). Σημειώνουμε πως μας επιτρέπεται ο συμβολισμός i_s, i_b καθώς κάθε στοιχείο είναι μοναδικό (από εκφώνηση).

ιι. Ελέγχουμε εάν η διαφορά b-s είναι αποδεκτή. Εάν είναι αποδεκτή τότε $f \leftarrow f + (i_b - i_s)$, μετακινούμε δεξιά τον δ_b και επαναλαμβάνουμε αυτό το βήμα (ιι).

Διαφορετικά, μετακινούμε το δ_s μία θέση δεξιά. Εάν ο δ_s είναι ακόμα αριστερά από τον δ_b, επαναλαμβάνουμε αυτό το βήμα (ιι). Εάν ο δ_s έφτασε τον δ_b, μετακινούμε το δ_b μία θέση δεξιά και επαναλαμβάνουμε αυτό το βήμα (ιι). Τερματίζουμε όταν ο δ_b ξεπεράσει τα όρια του πίνακα.

Η τελική τιμή του f είναι το πλήθος των διαφορών με τιμή μικρότερη ή ίση από l.

Πολυπλοκότητα

Στην αρχή της διαδικασίας, ταξινομούμε τον πίνακα n στοιχείων (O(nlogn)).

Έπειτα, καλούμε την F logM φορές όπως και στο (4α). Ένα καλύτερο άνω όριο θα ήταν το M-min, όπου min είναι το πρώτο στοιχείο στον ταξινομημένο πίνακα. Προφανώς, αυτή είναι η μέγιστη διαφορά.

Όσον αφορά την υλοποίηση της F, σε κάθε iteration κάνουμε το πολύ δυο αφαιρέσεις, μία πρόσθεση και μία σύγκριση. Επίσης, μετακινούμε τουλάχιστον τον ένα δείκτη κατά μία θέση δεξιά. Τερματίζουμε όταν ο δ_b φτάσει τα όρια του πίνακα οπότε έχουμε το πολύ 2n iteration.

Συνολικά, προκύπτει πολυπλοκότητα O(nlogn + nlogM) = **O (n logM)**. Θυμίζουμε πως έχουμε πίνακα θετικών διαφορετικών ακεραίων με μέγιστο στοιχείο M, οπότε αναγκαία n ≤ M (από αρχή του περιστερώνα).

Ορθότητα

Θεωρώντας δεδομένη την ορθότητα της F, η ορθότητα του συνολικού αλγορίθμου προκύπτει άμεσα όπως στο (α).

Ας μιλήσουμε για την ορθότητα της F:

Κάθε διαφορά δημιουργείται ανάμεσα σε δύο στοιχεία: big, small.

Έστω σταθερό στοιχείο big.

Θεωρούμε το σύνολο όλων των στοιχείων small, τέτοιων ώστε big - small ≤ l. Καθώς πρόκειται για πεπερασμένο σύνολο ακεραίων, σίγουρα ένα από αυτά τα στοιχεία είναι infinum (το ελάχιστο στοιχείο του συνόλου).

Είναι $small_{inf} \leq small < big \rightarrow$ $big\text{-}small_{inf} \geq big\text{-}small$ για κάθε στοιχείο small και $l > big\text{-}small_{inf}$. Ουσιαστικά, λέμε πως εφόσον έχουμε βρει το ελάχιστο small για το οποίο big-small ≤ l, τότε όλα τα στοιχεία μεγαλύτερα του small (μέχρι πριν το big) θα δώσουν ακόμα μικρότερη διαφορά, οπότε είναι αποδεκτά. Επίσης, επειδή είπαμε πως αυτό είναι το ελάχιστο στοιχείο το οποίο δίνει αποδεκτή διαφορά, δεν υπάρχει μικρότερο αποδεκτό small.

Επιπλέον, με την ίδια λογική, εάν ένα δεδομένο small δεν είναι αποδεκτό για κάποιο big, δηλαδή big-small>l, τότε προφανώς, το ίδιο small δε θα είναι αποδεκτό για κανένα στοιχείο μεγαλύτερο αυτού του big.

Με τον αλγόριθμό μας, ξεκινάμε από το ελάχιστο δυνατό small (το πρώτο στοιχείο) και big (το δεύτερο στοιχείο) και σε κάθε iteration, βρίσκουμε το $small_{inf}$ για το εκάστοτε big. Γνωρίζουμε πως το $small_{inf}$ θα είναι μεγαλύτερο ή ίσο από το αντίστοιχο $small_{inf}$ του ακριβώς μικρότερου big. Άπαξ και έχουμε το $small_{inf}$, θεωρούμε όλες τις διαφορές για το δεδομένο big, όπου ως small δεχόμαστε όλα τα στοιχεία από το $small_{inf}$ μέχρι πριν το big. Προσμετράμε αυτές τις διαφορές και συνεχίζουμε.

<u>Βελτιώσεις στην υλοποίηση της F</u>

Μπορούμε να βελτιώσουμε λίγο τον αλγόριθμο της F, αλλά η πολυπλοκότητα παραμένει γραμμική.

Η βελτίωση βασίζεται στο γεγονός πως ο πίνακας A είναι πια ταξινομημένος, οπότε μπορούμε εν μέρει να κάνουμε δυαδική αναζήτηση για κάποιες λειτουργίες.

Η ουσία του αλγορίθμου παραμένει ίδια (και φαίνεται στο βήμα iii).

i. Στην αρχή της διαδικασίας, βρίσκουμε το πλήθος των στοιχείων του A με τιμή μικρότερη ή ίση του l. Εφόσον ο πίνακας είναι πια ταξινομημένος αρκεί μια δυαδική αναζήτηση, αλλά ακόμα και η γραμμική αναζήτηση δε θα άλλαζε την τελική πολυπλοκότητα.

Έστω α αυτό το πλήθος. Τότε όλες οι εσωτερικές διαφορές ανάμεσα σε αυτά τα στοιχεία είναι σίγουρα μικρότερες του l. Το κάθε στοιχείο είναι μεγαλύτερο από όλα τα στοιχεία στα αριστερά του και μικρότερο από όλα τα στοιχεία στα δεξιά του. Για κάθε διαφορά dif, χρειαζόμαστε ένα big και ένα small. Σε αυτά τα α στοιχεία, το πρώτο (το ελάχιστο) δεν είναι μεγαλύτερο από κανένα στοιχείο, το δεύτερο είναι μεγαλύτερο μόνο από 1 κοκ. Έτσι, συνολικά το πλήθος των εσωτερικών διαφορών είναι 1 + 2 + 3 +...+ α-1 = [α*(α-1)]/2, οπότε τις μετρήσαμε με μία πράξη. Οπότε για την ώρα f ← [α*(α-1)]/2

ii. Στη συνέχεια παίρνουμε το πρώτο επόμενο στοιχείο, έστω b η τιμή του και i_b η θέση του, και θέτουμε ένα δείκτη δ_b στη θέση i_b. Με δυαδική αναζήτηση από το πρώτο στοιχείο εώς το στοιχείο b, βρίσκουμε το ελάχιστο στοιχείο με τιμή s και θέση i_s του πίνακα A για το οποίο b - s ≤ l. Θέτουμε ένα δείκτη δ_s στη θέση του s.

iii. Ελέγχουμε εάν η διαφορά b-s είναι αποδεκτή. Εάν είναι αποδεκτή τότε f ← f + (i_b-i_s) , μετακινούμε δεξιά τον δ_b και επαναλαμβάνουμε αυτό το βήμα (iii).

Διαφορετικά, μετακινούμε το δ_s μία θέση δεξιά. Εάν ο δ_s είναι ακόμα αριστερά από τον δ_b, επαναλαμβάνουμε αυτό το βήμα (iii). Εάν ο δ_s έφτασε τον δ_b, μετακινούμε το δ_b μία θέση δεξιά και επαναλαμβάνουμε αυτό το βήμα (iii). Τερματίζουμε όταν ο δ_b ξεπεράσει τα όρια του πίνακα.

Η τελική τιμή του f είναι το πλήθος των διαφορών με τιμή μικρότερη ή ίση από l.