Αλγόριθμοι

1η Σειρά Ασκήσεων

Ονοματεπώνυμο: Λιαροκάπης Αλέξανδρος
Αριθμός Μητρώου: 03114860

Άσκηση 1

Μέρος α

- $g_1(n) = n^4 = \Theta(n^4)$
- $g_2(n) = 2^{\log^3 n} = \Theta(n^{\log^2 n})$
- $g_3(n) = \frac{\log(10n!)}{\log^9 n} = \Theta(\frac{\log(n!)}{\log^9 n}) = \Theta(\frac{n \log n}{\log^9 n}) = \Theta(\frac{n}{\log^8 n})$
- $g_4(n) = n3^{4^{5^6}} = \Theta(n)$
- $g_5(n) = \Theta(n)$
- $g_6(n) = \sum_{k=0}^{n} k^3 = n^4 = \Theta(n^4)$
- $g_8(n) = \sqrt{n!} = \Theta(\sqrt{n!})$
- $g_9(n) = \sum_{k=0}^{n} k2^{-k} = 2 = \Theta(1)$
- $g_{10}(n) = \frac{n}{\log^{10} n} = \Theta(\frac{n^5}{\log^{10} n})$
- $g_{11}(n) = n \sum_{k=0}^{n} \binom{n}{k} = n2^n = \Theta(n2^n)$
- $g_{12}(n) = O(n)$
- $g_{13}(n) = \binom{2n}{n/4} = \Theta(\binom{8n}{n}) = O(n!)$
- $g_{14}(n) = \sum_{k=0}^{n} k2^k = \Theta(n2^n)$

Για το $g_{12}(n)$ έχουμε, $n \log 2 \leq \log \binom{2n}{n} \leq n \log(2e)$, άρα $g_{12}(n) = \Theta(n)$.
Η $g_5(n)$ έχει άνω φράγμα μικρότερο απο $O(g_{12})$

Τελικά έχουμε την εξής ταξινόμηση,

$$O(g_9) \leq O(g_5) \leq O(g_3) \leq O(g_4) \leq O(g_{12}) \leq O(g_1) \leq O(g_6)$$
$$O(g_{10}) \leq O(g_7) \leq O(g_2) \leq O(g_{11}) \leq O(g_{14}) \leq O(g_{13}) \leq O(g_8)$$

Μέρος β

(i)

$$\begin{aligned}
T(n) &= 3T(\frac{n}{4}) + n\log\log^7 n \\
&= 3(3T(\frac{n}{4^2}) + \frac{n}{4}\log\log^7(\frac{n}{4})) + n\log\log n \\
&= 3^2T(\frac{n}{4^2}) + 3\frac{n}{4^1}\log\log^7(\frac{n}{4^1}) + n\log\log n \\
&= 3^2T(\frac{n}{4^2}) + \sum_{k=1}^{1} 3^k\frac{n}{4^k}\log\log^7\frac{n}{4^k} \\
&= 3^2(3T(\frac{n}{4^3}) + \frac{n}{4^2}\log\log^7(\frac{n}{4^2})) + \sum_{k=1}^{1} 3^k\frac{n}{4^k}\log\log^7\frac{n}{4^k} \\
&= 3^3T(\frac{n}{4^3}) + \sum_{k=1}^{2} 3^k\frac{n}{4^k}\log\log^7\frac{n}{4^k} \\
&\dots \\
&= 3^{\log_4 n}T(1) + \sum_{k=1}^{\log_4 n} 3^k\frac{n}{4^k}\log\log^7\frac{n}{4^k}
\end{aligned}$$

Επίσης,

$$\begin{aligned}
A(n) &= \sum_{k=1}^{\log_4 n} 3^k\frac{n}{4^k}\log\log^7\frac{n}{4^k} \\
&= 7\sum_{k=1}^{\log_4 n} 3^k\frac{n}{4^k}\log\log\frac{n}{4^k} \\
&= 7\sum_{k=1}^{\log_4 n} 3^k\frac{n}{4^k}\log(\log n - \log 4^k) \\
&= 7\sum_{k=1}^{\log_4 n} 3^k\frac{n}{4^k}\log(\log n - \frac{\log_4 4^k}{\log_4 2}) \\
&= 7\sum_{k=1}^{\log_4 n} 3^k\frac{n}{4^k}\log\log n - 7\sum_{k=1}^{\log_4 n} 3^k\frac{n}{4^k}\log\frac{\log_4 4^k}{\log_4 2} \\
&= 7n(\log\log n\sum_{k=1}^{\log_4 n}\frac{3}{4}^k - \sum_{k=1}^{\log_4 n}\frac{3}{4}^k\log\frac{k}{\log_4 2})
\end{aligned}$$

Ο δεύτερος όρος του $A(n)$ είναι $\Theta(1)$ και επομένως $A = \Theta(n\log\log n)$. Άρα $T = \Theta(n\log\log n)$.

(ii) Από Master Theorem, $T = \Theta(n\log n)$

(iii) Από Master Theorem, $T = \Theta(n^{\log_4 9})$

(iv) $T(n) = \sum_{k=1}^{n} + T(1) = \log(\prod_{k=1}^{n}) + T(1) = \log(n!) + T(1) = \Theta(n\log n)$

Άσκηση 2

Μπορούμε να αναπαραστίσουμε ένα δέντρο με τη χρήση ενός πίνακα A, με το στοιχείο $A[i]$ να έχει αριστερό και δεξί παιδί τα στοιχεία $A[2i]$ και $A[2i+1]$ αντίστοιχα. Ένα στοιχείο $A[i]$ αναπαριστά σωρό αν $A[2i], A[2i+1]$ είναι μικρότερα και αναπαριστούν επίσης σωρό. Μπορούμε να ορίσουμε τον αλγόριθμο heapify(A,i) όπου αν τα στοιχεία $A[2i]$, $A[2i+1]$ αναπαριστούν σωρούς, τότε ο πίνακας αλλάζει κατάλληλα ετσι ώστε και το $A[i]$ να αναπαριστά σωρό.

Ορισμός του heapify,

```
def heapify(A,i)
    left <- 2*i
    right <- 2*i + 1
    largest <- 1
    if left <= heap_length(A) and A[left] > A[largest] then
        largest <- left
    if right <= heap_length(A) and A[right] > A[largest] then
        largest <- right
    if largest != i then
        swap(A[i], A[largest])
        heapify(A, largest)
```

Ο heapify έχει πολυπλοκότητα $O(h)$ όπου h το ύψος του $A[i]$ στο αναπαριστόμενο δέντρο του πίνακα. Για να φτιάξουμε έναν σωρό απο έναν πίνακα A, μπορούμε να εκτελέσουμε συνεχόμενα τον heapify για κάθε κόμβο ανα επίπεδο αρχίζοντας απο τα χαμηλότερα επίπεδα. Αφού για ύψος h έχουμε το πολύ $\frac{n}{2^h}$ κόμβους, συνολικά το κόστος της δημιουργίας του σωρού θα είναι $\sum_{h=0}^{\log n} \frac{n}{2^h} O(h) = O(n \sum_{k=0}^{\log n} \frac{h}{2^h}) = O(n)$. Η ένωση των δύο σωρών επομένως μπορεί να γίνει αντιγράφοντας τους δύο σωρούς σε έναν τρίτο πίνακα που θα περιέχει όλα τα στοιχεία και μετά εκτελώντας τον παραπάνω αλγόριθμο έτσι ώστε να μετατραπεί σε σωρό. Συνολικά θα έχουμε το κόστος των αντιγραφών και του αλγόριθμου δημιουργίας και επομένως η πολυπλοκότητα θα είναι γραμμική στον αριθμό των συνολικών στοιχείων των σωρών.

Άσκηση 3

(α) Η ταξινόμηση μπορεί να γίνει με χρήση couting sort ο οποίος έχει πολυπλοκότητα $O(n+k)$, $O(k)$ για την αρχικοποίηση των κουβάδων και $O(n)$ για την εισαγωγή και εξαγωγή των στοιχείων απο αυτούς. Το κάτω φράγμα δεν ισχύει γιατι ο counting sort δεν είναι αλγόριθμος σύγκρισης.

(β) Μπορούμε να χρησιμοποιήσουμε έναν τροποποιημένο για n-αδικούς αριθμούς counting sort που κάνει sort με βάση ένα συγκεκριμένο ψηφίο των δοθέντων αριθμών και έχει πολυπλοκότητα O(ν+N). Με τη χρήση αυτού και κάνοντας διαδοχικές ταξινομίσεις στους δοθέντες αριθμούς απο το LSD στο MSD, μπορούμε να ταξινομίσουμε τους αριθμούς με κόστος $O(n+N)*O(d)=O(N)$ όπου d ο μέγιστος αριθμός ψηφίων των αριθμών και σταθερά.

Άσκηση 4

(β) Διαλέγουμε διαδοχικά στοιχεία. Αρχίζουμε απο το πάνω-δεξιά στοιχείο. Επαναλαμβάνουμε την εξής διαδικασία. Αν το στοιχείο είναι μεγαλύτερο του στοιχείου που ψάχνουμε, τότε διαλέγουμε το αμέσως αριστερό στοιχείο. Αν είναι μικρότερο διαλέγουμε το αμέσως κάτω στοιχείο. Μετά απο a κινήσεις κάτω και b κινήσεις πάνω, θα έχουμε ακυρώσει τις a πάνω σειρές και τις b δεξιότερες στήλες. Ο αλγόριθμος σταματά όταν βρεθεί στην αριστερότερη στήλη ή χαμηλώτερη σειρά και εκτελέση μία τελική γραμμική αναζητηση. Ο παραπάνω αλγόριθμος έχει πολυπλοκότητα $O(n+m)$.

Άσκηση 5

Ξεκινώ με έναν βουλευτή ο οποίος έχει τιμή 1. Τον χαιρετώ διαδοχικά με τυχαίους βουλευτές αφήνοντας όσους χαιρετάνε στην άκρη. Αν χαιρετηθεί θετικά με έναν του αυξάνω κατα 1 την τιμή αλλιώς την μειώνω κατα 1. Αν ο βουλευτής φτάσει το 0, παίρνω τον τελευταίο που χαιρετήθηκε, του βάζω την τιμή 1 και επαναλαμβανω την διαδικασία. Στο τέλος ο τελευταίος βουλευτής που θα έχουμε διαλέξει θα ανοίκει στο μεγαλύτερο κόμμα στην οποία περίπτωση μπορούμε να τον χαιρετήσουμε με όλους έτσι ώστε να μετρήσουμε ακριβώς τις ψήφους του κόμματος. Ο παραπάνω αλγόριθμος είναι γραμμικός.