Θέματα προετοιμασίας ΚΕΦ. 1, 5, 6, 7, 8, 10 στο μάθημα «ΑΛΓΟΡΙΘΜΟΙ ΚΑΙ ΠΟΛΥΠΛΟΚΟΤΗΤΑ»

Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Πανεπιστήμιο Ιωαννίνων, Άρτα

τελευταία ενημέρωση: 20/10/2024

Κεφάλαιο 1

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
    a. Ένας αλγόριθμος με πολυπλοκότητα χρόνου O(nlogn) εκτελείται ταχύτερα από ότι ένας αλγόριθμος με πολυπλοκότητα O(n^2) για όλες τις περιπτώσεις εισόδου.
    b. Ο συμβολισμός του μεγάλου O παρέχει ένα κάτω όριο απόδοσης.
    c. Ο συμβολισμός του μεγάλου Ω παρέχει ένα κάτω όριο απόδοσης.
    d. Ο συμβολισμός του μεγάλου Θ παρέχει περισσότερη πληροφορία από το συμβολισμό του μεγάλου O.
    e. Ο συμβολισμός του μεγάλου O, Ω και Θ έχει να κάνει με το ρυθμό αύξησης του χρόνου εκτέλεσης καθώς τα δεδομένα εισόδου γίνονται πολύ μεγάλα.
    f. Το πρόβλημα της μέγιστης υποακολουθίας (MAXSUBARRAY) μπορεί να επιλυθεί με αλγόριθμο πολυπλοκότητας O(n^2) καθώς και με αλγόριθμο πολυπλοκότητας O(n).

Απάντηση: a (ΛΑΘΟΣ), b (ΛΑΘΟΣ), c (ΣΩΣΤΟ), d (ΣΩΣΤΟ), e (ΣΩΣΤΟ), f (ΣΩΣΤΟ)

2. Κατατάξτε τις ακόλουθες συναρτήσεις σε αύξουσα σειρά με βάση το ρυθμό αύξησής τους:
    a. log n
    b. n
    c. n^2
    d. n^(1/2)
    e. 2^n
    f. n!
    g. n^3
    h. log log n
    i. 1000
    j. n log n

Απάντηση: 1000 << log log n << log n << n^(1/2) << n << n log n << n^2 << n^3 << 2^n << n!

3. Ποιες από τις ακόλουθες συναρτήσεις είναι O(n^2);
    a. n
    b. n^(1/2)
    c. 2n^2
    d. n^2 + 1000n
    e. n^2/100 + 1
    f. n^2 + n + nlogn
    g. n^3
    h. n^3 + n^2
    i. 2^n
    j. n + logn

Απάντηση: n, n^(1/2), 2n^2, n^2 + 1000n, n^2/100 + 1, n^2 + n + nlogn, n + logn

4. Τι χρονική πολυπλοκότητα έχει ο αλγόριθμος εύρεσης της μεγαλύτερης τιμής σε μια ταξινομημένη ακολουθία n θέσεων που επιστρέφει την πρώτη τιμή του πίνακα;

Απάντηση: O(1)

5. Ποια χρονική πολυπλοκότητα χρησιμοποιείται συνήθως;

a. η χρονική πολυπλοκότητα καλύτερης περίπτωσης.
b. η χρονική πολυπλοκότητα μέσης περίπτωσης.
c. η χρονική πολυπλοκότητα χειρότερης περίπτωσης.

Απάντηση: c. η χρονική πολυπλοκότητα χειρότερης περίπτωσης.

6. Ποιο είναι το άθροισμα της σειράς 1+2+3+…+n;

Απάντηση: n(n+1)/2

7. Ποιο είναι το αποτέλεσμα της πράξης log_2 1048576 + log_10 10000 + log_3 27;

Απάντηση: 20 + 4 + 3 = 27

8. Δίνεται η ακολουθία 9 τιμών: -2, 1, -3, 4, -1, 2, 1, -5, 4.
   a. Ποια θα είναι η προθεματική ακολουθία (10 τιμές) που θα σχηματιστεί για την επίλυση του προβλήματος της μέγιστης υποακολουθίας (MAX SUBARRAY) με αλγόριθμο πολυπλοκότητας O(n^2);
   b. Ποια θα είναι η επιθεματική ακολουθία (10 τιμές) που θα σχηματιστεί για την επίλυση του προβλήματος της μέγιστης υποακολουθίας (MAX SUBARRAY) με τον αλγόριθμο του Kadane;

Απάντηση:
Η μέγιστη υποακολουθία είναι η: 4 + (-1) + 2 + 1 = 6
a) S_0 = 0, S_t = S_{t-1} + A_t

Ακολουθία τιμών (A)		**-2**	**1**	**-3**	**4**	**-1**	**2**	**1**	**-5**	**4**
Προθεματικά αθροίσματα (S)	**0**	**-2**	**-1**	**-4**	**0**	**-1**	**1**	**2**	**-3**	**1**
Δείκτες (t)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

b) M_0 = 0, M_t = max(0, M_{t-1} + A_t)

Ακολουθία τιμών (A)		**-2**	**1**	**-3**	**4**	**-1**	**2**	**1**	**-5**	**4**
Επιθεματικά αθροίσματα (M)	**0**	**0**	**1**	**0**	**4**	**3**	**5**	**6**	**1**	**5**
Δείκτες (t)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

9. Στην υλοποίηση μιας επεκτάσιμης ακολουθίας ποιο είναι το πλέον συμφέρον από τα ακόλουθα;
   a. Όταν γεμίζει η ακολουθία να αντιγράφονται τα περιεχόμενά της σε νέα ακολουθία με διπλάσιο μέγεθος.
   b. Όταν γεμίζει η ακολουθία να αντιγράφονται τα περιεχόμενά της σε νέα ακολουθία με μέγεθος μεγαλύτερο κατά 1 θέση
   c. Όταν γεμίζει η ακολουθία να αντιγράφονται τα περιεχόμενά της σε ακολουθία με μέγεθος μεγαλύτερο κατά 100 θέσεις

Απάντηση: a. Όταν γεμίζει η ακολουθία να αντιγράφονται τα περιεχόμενά της σε νέα ακολουθία με διπλάσιο μέγεθος.

10. Γράψτε κώδικα (η ψευδοκώδικα) που να αντιστρέφει μια ακολουθία τιμών χρησιμοποιώντας χώρο O(1).

Απάντηση: (κώδικας σε Python)

```
def reverse(a):
    for i in range(len(a) // 2):
```

```
        a[i], a[len(a) - 1 - i] = a[len(a) - 1 - i], a[i]



a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
reverse(a)
print(a)
```

11. Η επιμερισμένη ανάλυση (amortized analysis) αφορά:
    a. τον υπολογισμό της χρονικής πολυπλοκότητας μιας λειτουργίας.
    b. τον υπολογισμό της χρονικής πολυπλοκότητας ενός συνόλου λειτουργιών.
    c. το χρόνο που απαιτεί η εκτέλεση μιας λειτουργίας.
    d. το χρόνο που απαιτεί η εκτέλεση ενός συνόλου λειτουργιών.

Απάντηση: τον υπολογισμό της χρονικής πολυπλοκότητας ενός συνόλου λειτουργιών.

Κεφάλαιο 5

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
    a. Η έννοια της ουράς προτεραιότητας και του σωρού είναι ταυτόσημες.
    b. Ένας σωρός μπορεί να έχει διπλότυπα.
    c. Η δημιουργία ενός σωρού με διαδοχικές εισαγωγές τιμών είναι ταχύτερη από τη δημιουργία ενός σωρού από μια ακολουθία με heapify.
    d. Η ουρά προτεραιότητας είναι ένας αφηρημένος τύπος δεδομένων.
    e. Ένας σωρός ελαχίστων είναι ένας αφηρημένος τύπος δεδομένων.
    f. Η αποδοτικότερη υλοποίηση ενός σωρού είναι με δένδρα όπου κάθε κόμβος έχει 2 δείκτες προς τα παιδιά του.
    g. Σε έναν σωρό μεγίστων ένας κόμβος έχει μικρότερη τιμή από τα παιδιά του.
    h. Σε έναν σωρό ελαχίστων το ελάχιστο στοιχείο είναι τοποθετημένο στο χαμηλότερο επίπεδο του δένδρου στον πλέον δεξί κόμβο.
    i. Η λίστα [2, 4, 6, 10, 8, 15] μπορεί να αντιπροσωπεύει έναν σωρό ελαχίστων.

Απάντηση: a (ΛΑΘΟΣ), b (ΣΩΣΤΟ), c (ΛΑΘΟΣ), d (ΣΩΣΤΟ), e (ΛΑΘΟΣ), f (ΛΑΘΟΣ), g (ΛΑΘΟΣ), h (ΛΑΘΟΣ), i (ΣΩΣΤΟ)

2. Ποιες είναι οι χρονικές πολυπλοκότητες των ακόλουθων λειτουργιών σε έναν σωρό ελαχίστων;
    a. Λήψη μικρότερης τιμής.
    b. Αφαίρεση μικρότερης τιμής.
    c. Εισαγωγή νέας τιμής.
    d. heapify.
    e. heapsort.

Απάντηση:

a) Λήψη μικρότερης τιμής → O(1)
b) Αφαίρεση μικρότερης τιμής → O(logn)
c) Εισαγωγή νέας τιμής → O(logn)
d) heapify → O(n)
e) heapsort → O(nlogn)

3. Τι θα εμφανίσει ο ακόλουθος κώδικας σε Python;

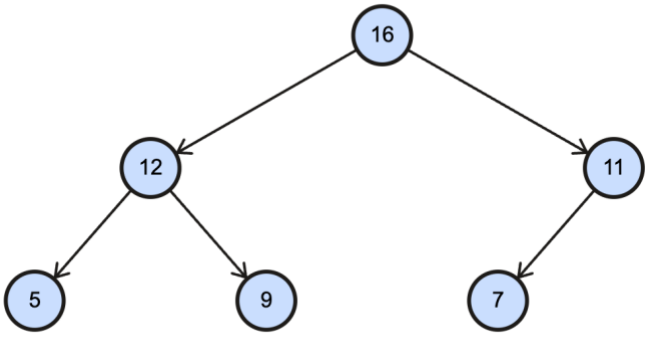
```
import heapq
li = [5, 7, 9, 1, 3]
li_minus = [-n for n in li]
heapq.heapify(li_minus)
print(-1 * heapq.heappop(li_minus))
```

Παρατήρηση: heapq είναι ένας σωρός ελαχίστων στην Python

Απάντηση: το αποτέλεσμα θα είναι η εξαγωγή της μέγιστης τιμής του li και η εμφάνισή της.

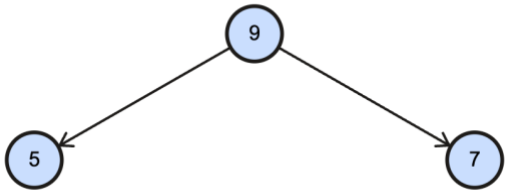
4. Σε έναν σωρό μεγίστων εισάγονται σταδιακά οι τιμές: 5, 12, 7, 9, 16, 11. Μετά από 3 εξαγωγές (της μέγιστης τιμής) ποια θα είναι τα περιεχόμενα του σωρού (να γράψετε τα στοιχεία με τη σειρά που θα εμφανίζονται στον πίνακα που υποστηρίζει το σωρό);

Απάντηση:
Ο σωρός μεγίστων μετά τη σταδιακή εισαγωγή των τιμών 5, 12, 7, 9, 16, 11:



-INF	16	12	11	5	9	7			
0	1	2	3	4	5	6	7	8	…

Μετά τις 3 εξαγωγές:



-INF	9	5	7						
0	1	2	3	4	5	6	7	8	…

5. Σε έναν σωρό σε ποιες θέσεις του πίνακα που τον υποστηρίζει εντοπίζονται τα παιδιά (αριστερό παιδί, δεξιό παιδί) του στοιχείου που βρίσκεται στη θέση 52. Σε ποια θέση βρίσκεται ο γονέας του ίδιου στοιχείου;
Απάντηση: Τα παιδιά του στοιχείου που βρίσκεται στη θέση i, βρίσκονται στις θέσεις 2*i και 2*i+1, ενώ ο γονέας του βρίσκεται στη θέση i // 2
Άρα, τα παιδιά του στοιχείου στη θέση 52, εφόσον υπάρχουν, βρίσκονται στις θέσεις 104 και 105 και ο γονέας στη θέση 26.

6. Ποια είναι η πλέον αποδοτική υλοποίηση μιας ουράς προτεραιότητας;
   a. με σωρό
   b. με αταξινόμητη λίστα
   c. με ταξινομημένη λίστα
   d. είναι το ίδιο και για τις 3 προηγούμενες περιπτώσεις

Απάντηση: Η αποδοτικότερη υλοποίηση είναι η α) με σωρό

7. Γράψτε κώδικα (ή ψευδοκώδικα) που να εμφανίζει τις 10 μικρότερες τιμές από 1000 τιμές που θα εισάγονται σε έναν σωρό ελαχίστων.
Απάντηση:

```
import random
import heapq
```

```
a = [random.randint(0, 100000) for _ in range(1000)]
heapq.heapify(a)    # οργάνωση του πίνακα a σε minheap


def print_heap(a, n):
    for i in range(n):
        print(heapq.heappop(a), end=" ")
    print()


print_heap(a, 10)
```

Κεφάλαιο 6

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
    a. Ο κατακερματισμός κλειστής διευθυνσιοδότησης (κατακερματισμός με αλυσίδες) μπορεί να έχει load factor πάνω από 100%.
    b. Ο κατακερματισμός ανοικτής διευθυνσιοδότησης μπορεί να έχει load factor πάνω από 100%.
    c. Μια συνάρτηση κατακερματισμού θα πρέπει για δύο διαφορετικές εισόδους να επιστρέφει την ίδια τιμή.
    d. Με ευκολία και αποδοτικά μπορούμε να διασχίσουμε ταξινομημένα τα κλειδιά που βρίσκονται σε έναν πίνακα κατακερματισμού.
    e. Οι κρυπτογραφικές συναρτήσεις κατακερματισμού χρησιμοποιούνται στην υλοποίηση πινάκων κατακερματισμού.
    f. Ο διπλός κατακερματισμός είναι μια επέκταση του κατακερματισμού κλειστής διευθυνσιοδότησης (κατακερματισμός με αλυσίδες) για τον χειρισμό των συγκρούσεων.
    g. Ο κατακερματισμός έχει εφαρμογή στην κρυπτογραφία.
    h. Οι πίνακες κατακερματισμού συνήθως έχουν ως μέγεθος έναν πρώτο αριθμό.
    i. Η συνάρτηση κατακερματισμού murmur είναι κρυπτογραφική.
    j. Η συνάρτηση κατακερματισμού SHA256 είναι κρυπτογραφική.

    Απάντηση: a (ΣΩΣΤΟ), b (ΛΑΘΟΣ), c (ΛΑΘΟΣ), d (ΛΑΘΟΣ), e (ΛΑΘΟΣ), f (ΛΑΘΟΣ), g(ΣΩΣΤΟ) ,h (ΣΩΣΤΟ), i (ΛΑΘΟΣ), j (ΣΩΣΤΟ)

2. Στον κατακερματισμό κούκου:
    a. Υπάρχουν δύο πίνακες κατακερματισμού όπου σε κάθε θέση τους υπάρχει μία συνδεδεμένη λίστα.
    b. Υπάρχει ένας πίνακα κατακερματισμού όπου σε κάθε θέση του υπάρχει μία συνδεδεμένη λίστα.
    c. Υπάρχει ένας πίνακα κατακερματισμού όπου σε κάθε θέση του υπάρχει ένα στοιχείο.
    d. Υπάρχουν δύο πίνακες κατακερματισμού όπου σε κάθε θέση τους υπάρχει ένα στοιχείο.

    Απάντηση: a (ΛΑΘΟΣ), b (ΛΑΘΟΣ), c (ΛΑΘΟΣ), d (ΣΩΣΤΟ)

3. Σε έναν πίνακα κατακερματισμού με ανοικτή διευθυνσιοδότηση με χωρητικότητα 101 τιμών έχουν εισαχθεί 20 τιμές. Ποιο είναι το load factor;
    Απάντηση: Το load factor είναι περίπου 20%, 20/101

4. Συμπληρώστε τις πολυπλοκότητες χρόνου για τις ακόλουθες περιπτώσεις:
    a. Χρόνος χειρότερης περίπτωσης και επιμερισμένος χρόνος λειτουργίας εισαγωγής σε πίνακα κατακερματισμού ανοικτής διευθυνσιοδότησης
    b. Χρόνος χειρότερης περίπτωσης και επιμερισμένος χρόνος λειτουργίας διαγραφής σε πίνακα κατακερματισμού ανοικτής διευθυνσιοδότησης

c. Χρόνος χειρότερης περίπτωσης και επιμερισμένος χρόνος λειτουργίας εύρεσης σε πίνακα κατακερματισμού ανοικτής διευθυνσιοδότησης
d. Χρόνος χειρότερης περίπτωσης και επιμερισμένος χρόνος λειτουργίας εισαγωγής σε πίνακα κατακερματισμού κούκου
e. Χρόνος χειρότερης περίπτωσης και επιμερισμένος χρόνος λειτουργίας διαγραφής σε πίνακα κατακερματισμού κούκου
f. Χρόνος χειρότερης περίπτωσης και επιμερισμένος χρόνος λειτουργίας εύρεσης σε πίνακα κατακερματισμού κούκου

Απάντηση:
Κατακερματισμός ανοικτής διευθυνσιοδότησης:
a) εισαγωγή: χειρότερη περίπτωση O(n), επιμερισμένος χρόνος O(1)
b) διαγραφή: χειρότερη περίπτωση O(n), επιμερισμένος χρόνος O(1)
c) εύρεση: χειρότερη περίπτωση O(n), επιμερισμένος χρόνος O(1)
Κατακερματισμός κούκου:
a) εισαγωγή: χειρότερη περίπτωση O(n), επιμερισμένος χρόνος O(1)
b) διαγραφή: χειρότερη περίπτωση O(1), επιμερισμένος χρόνος O(1)
c) εύρεση: χειρότερη περίπτωση O(1), επιμερισμένος χρόνος O(1)

5. Δίνεται μια λίστα με διευθύνσεις IP επισκεπτών που επισκέφθηκαν μια ιστοσελίδα. Γράψτε αποδοτικό κώδικα (ή ψευδοκώδικα) που να εντοπίζει το πλήθος των μοναδικών επισκέψεων.
Απάντηση:

```
def count_unique_visits(ip_list):
    unique_ips = set(ip_list)
    return len(unique_ips)
```

6. Δίνονται 2 ακολουθίες, η ακολουθία A με M στοιχεία και η ακολουθία B με N στοιχεία αντίστοιχα. θεωρείστε ότι το M είναι σημαντικά μικρότερο από το N. Γράψτε κώδικα (ή ψευδοκώδικα) που να εξετάζει αποδοτικά το εάν οι ακολουθίες είναι ξένες μεταξύ τους. Τι υπολογιστική πολυπλοκότητα θα έχει ο αλγόριθμός σας;
Απάντηση:

```
def are_disjoint(sequence_A, sequence_B):
    set_A = set(sequence_A)

    for item in sequence_B:
        if item in set_A:
            return False  # Βρέθηκε κοινό στοιχείο, οπότε οι ακολουθίες δεν είναι ξένες

    return True  # Δεν βρέθηκε κανένα κοινό στοιχείο, οι ακολουθίες είναι ξένες


sequence_A = [1, 2, 3]  # Μικρότερη ακολουθία M στοιχείων
sequence_B = [4, 5, 6, 7, 8, 9, 10]  # Μεγαλύτερη ακολουθία N στοιχείων

disjoint_result = are_disjoint(sequence_A, sequence_B)
print("Οι ακολουθίες είναι ξένες μεταξύ τους:", disjoint_result)
```

Η πολυπλοκότητα είναι O(M+N), είναι αποδοτικότερο να μετατραπεί σε set η μικρή ακολουθία, παρά η μεγάλη ακολουθία

7. Γράψτε κώδικα σε python στον οποίο ο χρήστης θα δίνει ονόματα χρηστών. Αν ο χρήστης υπάρχει θα επιστρέφει το password του. Αν δεν υπάρχει θα τον αποθηκεύει και θα ζητάει από τον χρήστη να εισάγει το password του.

Απάντηση:

```
users = {}


while True:
    username = input("Enter username: ")
    if username in users:
        print(f"Password for {username} is: {users[username]}")
    else:
        password = input("User not found. Enter a password: ")
        users[username] = password
        print("User and password stored.")
```

Κεφάλαιο 7

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
   a. Η υλοποίηση της δομής ένωσης-εύρεσης με συνδεδεμένες λίστες είναι αποδοτικότερη από την υλοποίηση της με δένδρα.
   b. Η υλοποίηση της δομής-ένωσης εύρεσης με δένδρα στηρίζεται σε δομή δένδρων με δείκτες από τον κάθε κόμβο γονέα προς τα παιδιά του.
   c. Η δομή ένωσης-εύρεσης είναι γνωστή και ως δομή ξένων συνόλων.
   d. Η συμπίεση μονοπατιού είναι μια τεχνική που εφαρμόζεται κατά την εισαγωγή νέων τιμών σε μια δομή ένωσης-εύρεσης.
   e. Στη δομή ένωσης-εύρεσης στην ένωση κατά μέγεθος (union by size), το σύνολο με τα περισσότερα στοιχεία προσαρτάται ως παιδί στο αντιπροσωπευτικό στοιχείο του συνόλου με τα λιγότερα στοιχεία.

Απάντηση: a (ΛΑΘΟΣ), b (ΛΑΘΟΣ), c (ΣΩΣΤΟ), d (ΛΑΘΟΣ), e (ΛΑΘΟΣ)

2. Ποιες είναι οι λειτουργίες που υποστηρίζει μια δομή ένωσης-εύρεσης;

Απάντηση: Οι λειτουργίες είναι οι: makeset(e), union(A,B), find(e)

3. Έστω ο αλγόριθμος δημιουργίας λαβυρίνθων με δομή ένωσης-εύρεσης. Ποια θα είναι η ενδιάμεση (μη τελική) μορφή του ακόλουθου λαβυρίνθου αν επιλεγούν με τη σειρά οι τοίχοι 3-6, 2-3, 5-6 και 2-5. Ποια θα είναι τα ξένα σύνολα που θα έχουν δημιουργηθεί;

1	2	3
4	5	6

Απάντηση:

1 2 3 **4 5 6**	**1 2 3** **4 5 6**	**1 2 3** **4 5 6**	**1 2 3** **4 5 6**
Επιλογή τοίχου 3-6 (διαγράφεται)	**Επιλογή τοίχου 2-3 (διαγράφεται)**	**Επιλογή τοίχου 5-6 (διαγράφεται)**	**Επιλογή τοίχου 2-5 (δεν διαγράφεται)**

Τα ξένα σύνολα που έχουν δημιουργηθεί είναι τα {1}, {4}, {2,3,5,6}

4. Δίνεται ένα σύνολο από σχέσεις φιλίας ανάμεσα σε άτομα (κάθε άτομο αναπαρίσταται από έναν αριθμό και η σχέση φιλίας με ένα ζεύγος αριθμών). Γράψτε Python κώδικα που να εντοπίζει τις ομάδες ατόμων που είναι συνδεδεμένες χρησιμοποιώντας δομή ένωσης εύρεσης (π.χ. friends = [[1, 0], [2, 3], [1, 2], [4, 5]] με ομάδες [0,1,2,3] και [4,5]). Θεωρείστε ότι μπορείτε να χρησιμοποιήσετε τη βιβλιοθήκη της Python unionfind με την ακόλουθη λειτουργικότητα:

```
from unionfind import unionfind
u = unionfind(3) # There are 3 items.
u.unite(0, 2) # Set 0 and 2 to same group.
u.issame(1, 2) # Ask "Are 1 and 2 same?"
u.groups() # Return groups.
```

Απάντηση:

```
# https://pypi.org/project/unionfind/

from unionfind import unionfind

friends = [[ 1, 0 ], [ 2, 3 ], [ 1, 2 ], [ 4, 5 ]]

u = unionfind(6)
for pair in friends:
    u.unite(pair[0], pair[1])
print(u.groups())
```

Κεφάλαιο 8

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
    a. Το κάτω όριο για αλγόριθμους ταξινόμησης με συγκρίσεις είναι Ω(nlogn).
    b. Το κάτω όριο για αλγόριθμους ταξινόμησης με συγκρίσεις είναι O(nlogn).
    c. Το κάτω όριο για αλγόριθμους ταξινόμησης με συγκρίσεις είναι O(n^2).
    d. Ο αλγόριθμος γρήγορης ταξινόμησης συνήθως είναι ταχύτερος από τον αλγόριθμο ταξινόμησης με συγχώνευση και προτιμάται για ταξινόμηση πολλών δεδομένων που βρίσκονται στην κύρια μνήμη.
    e. Ο αλγόριθμος ταξινόμησης με συγχώνευση είναι τυχαιοποιημένος αλγόριθμος.
    f. Ένας αλγόριθμος σταθερής ταξινόμησης (stable sort) επιτρέπει σε μια λίστα με φοιτητές που ταξινομείται πρώτα κατά όνομα και μετά ταξινομείται ξανά κατά έτος να εμφανίζει τους φοιτητές του κάθε έτους ταξινομημένους σε αλφαβητική σειρά ονόματος.
    g. Η MergeSort περιέχει δύο αναδρομικές κλήσεις της.
    h. Η BinarySearch περιέχει δύο αναδρομικές κλήσεις της.
    i. Η εκτέλεση της QuickSort για μια ακολουθία n στοιχείων, προκαλεί αναδρομικές κλήσεις της QuickSort για 2 ακολουθίες μεγέθους n/2 η κάθε μια.

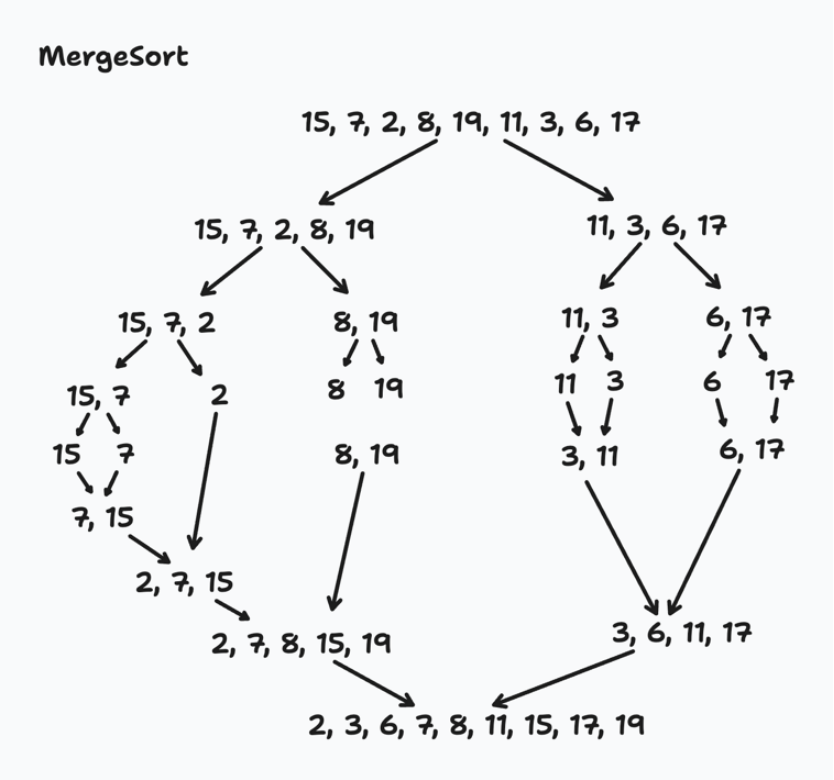
Απάντηση: a (ΣΩΣΤΟ), b (ΛΑΘΟΣ), c (ΛΑΘΟΣ), d (ΣΩΣΤΟ), e (ΛΑΘΟΣ), f (ΣΩΣΤΟ), g (ΣΩΣΤΟ), h (ΛΑΘΟΣ), i (ΛΑΘΟΣ)

2. Συμπληρώστε τις πολυπλοκότητες χρόνου για τις ακόλουθες περιπτώσεις;
    a. Πολυπλοκότητα χρόνου χειρότερης περίπτωσης για τον αλγόριθμο QuickSort.
    b. Πολυπλοκότητα χρόνου μέσης περίπτωσης για τον αλγόριθμο QuickSort.
    c. Πολυπλοκότητα χρόνου χειρότερης περίπτωσης για τον αλγόριθμο MergeSort.

d. Πολυπλοκότητα χρόνου χειρότερης περίπτωσης για τον αλγόριθμο InsertionSort.

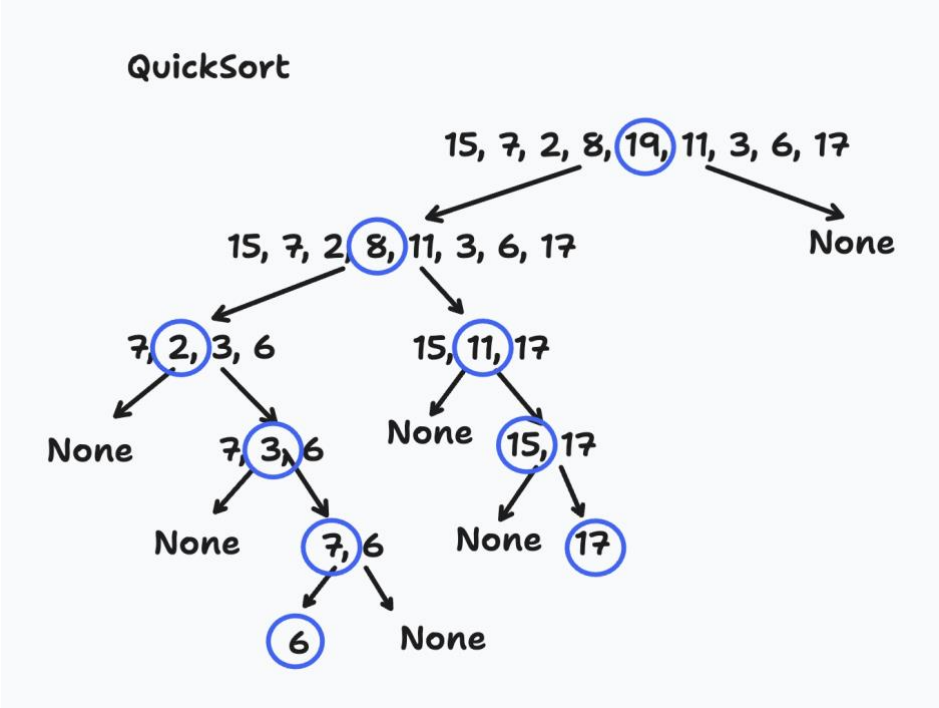
Απάντηση: a → O(n^2), b → O(nlogn), c → O(nlogn), d → O(n^2)

3. Δείξτε τη σταδιακή ταξινόμηση με MergeSort της ακολουθίας 15,7,2,8,19,11,3,6,17.
Απάντηση:



4. Δείξτε τη σταδιακή ταξινόμηση με QuickSort της ακολουθίας 15,7,2,8,19,11,3,6,17 θεωρώντας ως pivot το μεσαίο στοιχείο κάθε φορά της υποακολουθίας (αν το πλήθος στοιχείων της ακολουθίας είναι άρτιο, το αριστερό από τα δύο μεσαία στοιχεία).

Απάντηση:



5. Δίνεται η ακολουθία τιμών 15,7,2,8,19,11,3,6,17. Ποια θα είναι η μορφή της ακολουθίας αν εφαρμοστεί διαμερισμός με pivot το τελευταίο στοιχείο της ακολουθίας;
    a. χρησιμοποιώντας βοηθητικό πίνακα
    b. in-place.

Απάντηση:

a) 15,7,2,8,19,11,3,6,17 → 15,7,2,8,11,3,6,17,19

b) 15,7,2,8,19,11,3,6,17 → 15,7,2,8,6,11,3,19,17 → 15,7,2,8,6,11,3,17,19

6. Γράψτε κώδικα (ή ψευδοκώδικα) συγχώνευσης 2 ταξινομημένων σε αύξουσα σειρά πινάκων με N τιμές ο καθένας, που να επιστρέφει τις N μικρότερες τιμές και από τους 2 πίνακες. Ο κώδικας να έχει χρονική πολυπλοκότητα O(N).

Απάντηση:

```
def merge_N(a, b, n):
    c = [None] * n
    i = 0
    j = 0
    for k in range(n):
        if a[i] < b[j]:
            c[k] = a[i]
            i += 1
        else:
            c[k] = b[j]
            j += 1
    return c


a = [1, 5, 7, 9, 13]
b = [2, 3, 16, 21, 23]
c = merge_N(a, b, 5)
print(c)
```

7. Γράψτε κώδικα (ή ψευδοκώδικα) που να πραγματοποιεί διαμερισμό σε έναν πίνακα, δεχόμενος ως είσοδο τον πίνακα και τη θέση του pivot. Να χρησιμοποιηθεί βοηθητικός πίνακας.

Απάντηση:

```
def partition_based_on_last_item(a):
    temp = [None] * len(a)
    pivot = a[-1]
    i = 0
    j = len(a) - 1
    for x in a:
        if x < pivot:
            temp[i] = x
            i += 1
        else:
```

```
            temp[j] = x
            j -= 1
    temp[i] = pivot
    return temp


a = [2, 3, 18, 4, 5, 6, 17, 9]
a = partition_based_on_last_item(a)
print(a)
```

Κεφάλαιο 10

1. Απαντήστε με ΣΩΣΤΟ/ΛΑΘΟΣ
    a. Η άπληστη μέθοδος επιστρέφει τη βέλτιστη λύση για το κλασματικό πρόβλημα σακιδίου.
    b. Η άπληστη μέθοδος επιστρέφει τη βέλτιστη λύση για το 0-1 πρόβλημα σακιδίου.
    c. Η άπληστη μέθοδος επιστρέφει τη βέλτιστη λύση για το πρόβλημα χρονοπρογραμματισμού εργασιών σε πολλαπλές ίδιες μηχανές με γνωστές ώρες έναρξης και τερματισμού της κάθε εργασίας, εφόσον οι εργασίες ανατεθούν στις μηχανές με σειρά μεγέθους.
    d. Η άπληστη μέθοδος επιστρέφει τη βέλτιστη λύση για το πρόβλημα χρονοπρογραμματισμού εργασιών σε πολλαπλές μηχανές με γνωστές ώρες έναρξης και τερματισμού της κάθε εργασίας, εφόσον οι εργασίες ανατεθούν στις μηχανές με σειρά ώρας έναρξης.
    e. Η άπληστη μέθοδος συνήθως λύνει βέλτιστα τα προβλήματα.
    f. Η κωδικοποίηση Huffman χρησιμοποιεί το ίδιο πλήθος δυαδικών ψηφίων για να κωδικοποιήσει κάθε χαρακτήρα του κειμένου.
    g. Η κωδικοποίηση Huffman είναι ένα είδος κρυπτογράφησης.
    h. Η κωδικοποίηση Huffman μπορεί να θεωρηθεί ως μια μορφή συμπίεσης.

Απάντηση: a (ΣΩΣΤΟ), b (ΛΑΘΟΣ), c (ΛΑΘΟΣ), d (ΣΩΣΤΟ), e (ΛΑΘΟΣ), f (ΛΑΘΟΣ), g (ΛΑΘΟΣ), h (ΣΩΣΤΟ)

2. Δίνονται τα ακόλουθα αντικείμενα τα οποία συμμετέχουν στο πρόβλημα του κλασματικού σακιδίου. Αντικείμενο Α με βάρος 100kg και αξία 20 ευρώ, αντικείμενο Β με βάρος 300kg και αξία 150 ευρώ, αντικείμενο Γ με βάρος 5kg και αξία 10 ευρώ. Με δεδομένο ότι το επιτρεπτό βάρος του σακιδίου είναι 150kg να υπολογιστεί η βέλτιστη λύση του προβλήματος.

Απάντηση: Αξία_Α = 20/100 = 0.2 ευρώ ανά κιλό, Αξία_Β = 150/300 = 0.5 ευρώ ανά κιλό, Αξία_Γ = 10/5 = 2 ευρώ ανά κιλό. Η βέλτιστη λύση θα χρησιμοποιήσει όλο το απόθεμα από το αντικείμενο Γ δηλαδή 5kg, και 145kg από το αντικείμενο Β. Η συνολική αξία της λύσης θα είναι 5*2 + 145*0.5 = 10 + 72.5 = 82.5 ευρώ.

3. Δίνεται μια λίστα αντικειμένων για τα οποία γνωρίζουμε την ποσότητά του κάθε αντικειμένου σε κιλά και την αξία του σε ευρώ. Αν θεωρήσουμε ότι μπορούμε να επιλέξουμε για κάθε αντικείμενο ένα αυθαίρετα μικρό ή μεγάλο ποσοστό της διαθέσιμης ποσότητας του, να επιλεγούν τα αντικείμενα που θα δώσουν τη μεγαλύτερη αξία δεδομένης της συνολικής ποσότητας σε κιλά που μπορεί να αποκτηθεί. Γράψτε κώδικα (ή ψευδοκώδικα) που να υλοποιεί τα παραπάνω.

Απάντηση:

```
def fractional_knapsack(items, capacity):
    items.sort(key=lambda x: x['value'] / x['weight'], reverse=True)


    total_value = 0
    for item in items:
        if capacity > item['weight']:
```

```
            total_value += item['value']
            capacity -= item['weight']
        else:
            # Πάρε μόνο το μέρος του αντικειμένου που χωράει και σταμάτα
            total_value += item['value'] * (capacity / item['weight'])
            break

    return total_value

items = [{'weight': 100, 'value': 20},
         {'weight': 300, 'value': 150},
         {'weight': 5, 'value': 10}]

# Συνολική ποσότητα σε κιλά που μπορεί να αποκτηθεί
capacity = 150

max_value = fractional_knapsack(items, capacity)
print(f"Η μέγιστη αξία που μπορεί να αποκτηθεί είναι: {max_value} ευρώ")
```

4. Δίνεται μια λίστα εργασιών με γνωστούς χρόνους έναρξης και τερματισμού. Κάθε εργασία μπορεί να εκτελεστεί σε μια μηχανή και υπάρχουν πολλές διαθέσιμες ίδιες μηχανές. Γράψτε κώδικα (ή ψευδοκώδικα) που να αναθέτει βέλτιστα τις εργασίες σε μηχανές έτσι ώστε να χρησιμοποιούνται συνολικά οι λιγότερες δυνατές μηχανές.

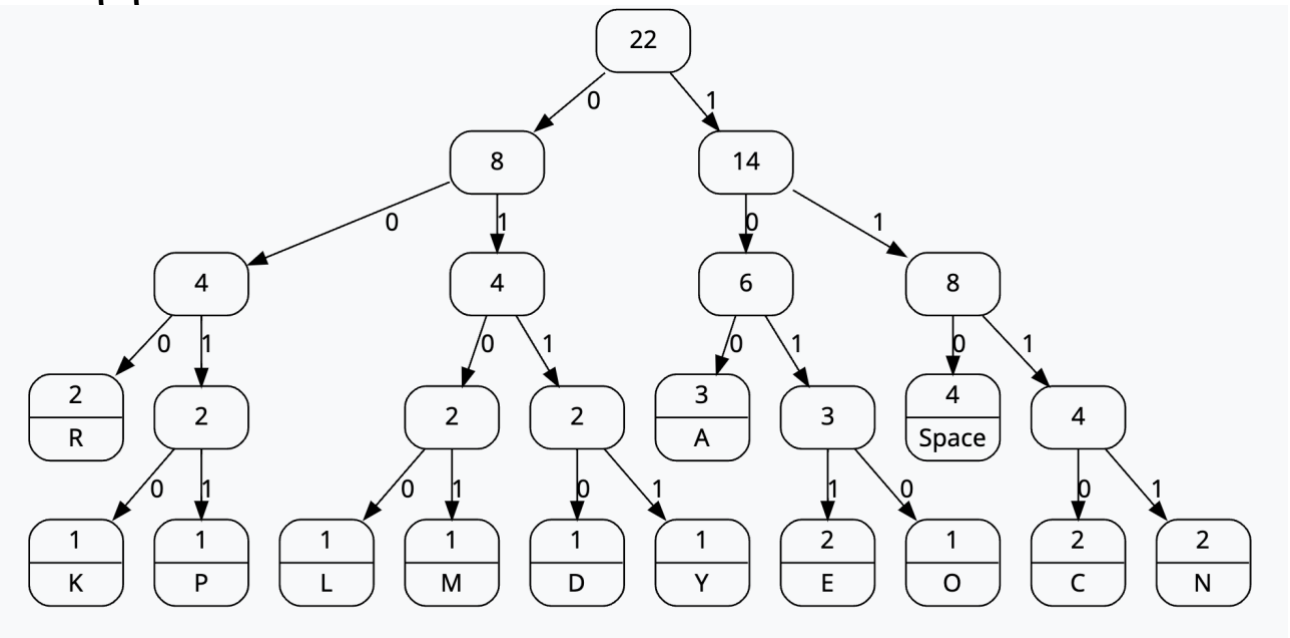
Απάντηση:

```
def schedule_min_machines(tasks):
    tasks_per_machine = [] # Λίστα με λίστες εργασιών που εκτελούνται σε κάθε μηχάνημα
    sorted_tasks = sorted(tasks, key=lambda x: x[0]) # Ταξινόμηση των εργασιών με βάση την αρχή τους
    for task in sorted_tasks:
        # Αν υπάρχει μηχάνημα που έχει τελειώσει την εργασία του πριν την έναρξη της εργασίας
        # τότε προσθέτουμε την εργασία σε αυτό το μηχάνημα
        for machine in tasks_per_machine:
            if machine[-1][1] <= task[0]:
                machine.append(task)
                break
        else:
            # Αν δεν υπάρχει τέτοιο μηχάνημα, τότε δημιουργούμε ένα νέο μηχάνημα
            tasks_per_machine.append([task])
    return tasks_per_machine

tasks = [[1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]] # Παράδειγμα από το βιβλίο Goodrich-Tamassia
tasks_per_machine = schedule_min_machines(tasks)
print(tasks_per_machine)
```

5. Δίνεται η ακόλουθη φράση «KEEP CALM AND CARRY ON». Χρησιμοποιήστε την κωδικοποίηση Huffman για την κωδικοποίηση της.

Απάντηση:



https://www.csfieldguide.org.nz/en/interactives/huffman-tree/