Αλγόριθμοι και Πολυπλοκότητα **Όνομα: Ανδρέας Χαραλαμπόπουλος**

Φυλλάδιο Ασκήσεων 1

AM: el19134

Άσκηση 1: Αναδρομικές Σχέσεις

Να υπολογίσετε την τάξη μεγέθους $\Theta()$ των λύσεων των παρακάτω αναδρομικών σχέσεων. Για όλες τις σχέσεις να θεωρήσετε ότι $T(1) = \Theta(1)$.

1. $T(n) = 4T(n/2) + \Theta(n^2 \log n)$
2. $T(n) = 5T(n/2) + \Theta(n^2 \log n)$
3. $T(n) = T(n/4) + T(n/2) + \Theta(n)$
4. $T(n) = 2T(n/4) + T(n/2) + \Theta(n)$
5. $T(n) = T(n^{1/2}) + \Theta(\log n)$
6. $T(n) = T(n/4) + \Theta(\sqrt{n})$.

Λύση:

1. $$\begin{aligned} T(n) &= 4T(n/2) + \Theta(n^2 \log n) = \\ &= \Theta(n^2 \sum_{i=0}^{\log(n)} \log(\frac{n}{2^i})) = \\ &= \Theta(n^2 \sum_{i=0}^{\log(n)} (\log(n) + \log(2^{-i})) = \\ &= \Theta(n^2 \log^2(n)) \end{aligned}$$
2. Από (MT) έχουμε αμέσως ότι $T(n) = \Theta(n^{\log_2(5)})$
3. Επειδή $\frac{1}{4} + \frac{1}{2} < 1$ από ειδική περίπτωση του (MT) έχουμε ότι $T(n) = \Theta(n)$
4. Από το αναδρομικό δέντρο βλέπουμε ότι έχουμε $\log(n)$ επίπεδα όπου σε κάθε επίπεδο το συνολικό κόστος αθροίζεται σε n και συνεπώς έχουμε $T(n) = \Theta(n \log(n))$
5. Έχουμε $T(n) \leq \Theta(\sum_{i=0}^{+\infty} \log(n) 2^{-i}) = \Theta(\log(n))$ και άρα $T(n) = \Theta(\log(n))$
6. $T(n) \leq \Theta(\sqrt{(}n) \sum_{i=0}^{+\infty} 2^{-i}) = \Theta(\sqrt{(}n))$ και άρα $T(n) = \Theta(\sqrt{(}n))$

Άσκηση 2: Προθεματική Ταξινόμηση

Έστω μη ταξινομημένος πίνακας $A[1 \dots n]$ αρχικοποιημένος σε μία μετάθεση του συνόλου $\{1, \dots, n\}$. Θέλουμε να ταξινομήσουμε τον πίνακα A χρησιμοποιώντας μόνο προθεματικές περιστροφές. Μια προθεματική περιστροφή υλοποιείται επιλέγοντας κάποιο $k \leq n$ και αντιστρέφοντας τη σειρά των k αριστερότερων στοιχείων του πίνακα ως εξής:

Στόχος μας είναι να ταξινομήσουμε τον πίνακα A (σε αύξουσα σειρά) κάνοντας όσο το δυνατόν λιγότερες προθεματικές περιστροφές στην χειρότερη περίπτωση.

(α) Να διατυπώσετε αλγόριθμο που ταξινομεί τον A χρησιμοποιώντας το πολύ $2n$ προθεματικές περιστροφές. Εδώ η ανάλυσή σας οφείλει να είναι κατά το δυνατόν ακριβής και να προσδιορίζει πλήρως και τις σταθερές (π.χ. το $2n - 3$ θεωρείται καλύτερο από το $2n$).

(β) Θεωρούμε τώρα ότι οι επιτρεπτές κινήσεις είναι οι προσημασμένες προθεματικές περιστροφές. Δηλαδή, κάθε φορά που ένα στοιχείο του τρέχοντος πίνακα συμμετέχει σε μία περιστροφή, αντιστρέφουμε το πρόσημό του:
Ο στόχος μας παραμένει να ταξινομήσουμε τα στοιχεία έτσι ώστε να εμφανίζονται τελικά σε αύξουσα σειρά και με τα αρχικά (θετικά) πρόσημα, παρόλο που στα ενδιάμεσα βήματα, κάποια πρόσημα μπορεί να γίνουν αρνητικά.
Να διατυπώσετε αλγόριθμο που ταξινομεί τον πίνακα A χρησιμοποιώντας το πολύ $3n$ προσημασμένες προθεματικές περιστροφές.

(γ) Θα λέμε ότι ένα ζεύγος στοιχείων $(A_t[i], A_t[i+1])$ σε έναν ενδιάμεσο πίνακα A_t (που προκύπτει από τον A_0 μετά από t διαδοχικές προσημασμένες προθεματικές περιστροφές) είναι συμβατό όταν τα στοιχεία $A_t[i]$ και $A_t[i+1]$ έχουν το ίδιο πρόσημο, η απόλυτη τιμή των στοιχείων $A_t[i]$ και $A_t[i+1]$ διαφέρει κατά 1 (άρα οι φυσικοί αριθμοί $|A_t[i]|$ και $|A_t[i+1]|$ είναι διαδοχικοί) και τα στοιχεία $A_t[i]$ και $A_t[i+1]$ είναι ταξινομημένα στον πίνακα A_t (λαμβάνοντας υπόψη και το πρόσημό τους).

1. Να δείξετε ότι για κάθε ενδιάμεσο πίνακα A_t, που δεν είναι ο $[-1, -2, \dots, -n]$, μπορούμε να κατασκευάσουμε ένα νέο συμβατό ζεύγος έπειτα από 2 το πολύ προσημασμένες προθεματικές περιστροφές. Να θεωρήσετε, καταχρηστικά, τη μετακίνηση του μεγαλύτερου στοιχείου του At στην τελευταία θέση, με θετικό πρόσημο, ως δημιουργία συμβατού ζεύγους.
2. Να διατυπώσετε αλγόριθμο που ταξινομεί τον A χρησιμοποιώντας το πολύ $2n$ προσημασμένες προθεματικές περιστροφές.

Λύση:

(α) Ταξινομούμε τα πρώτα $n - k$ μεγαλύτερα στοιχεία στις θέσεις τους με τον εξής τρόπο:

· Αν m_i η θέση του $i-$οστού μεγαλύτερου στοιχείου του πίνακα εκτελούμε μία περιστροφή $1 - m_i$ και μία $1 - i$ και το στοιχείο έχει μπει στη σωστή θέση (χωρίς να επηρεαστεί η θέση των $i-1$ μεγαλύερων του στοιχείων που έχουν τοποθετηθεί στη σωσή θέση νωρίτερα). Συνεπώς χρειαζόμαστε 2 κινήσεις για να τοποθετήσουμε το

κάθε στοιχείο στη σωστή θέση στη χειρότερη περίπτωση άρα συνολικά $2n$ περιστροφές στη χειρότερη περίπτωση. Μπορούμε να μειώσουμε και άλλο τις περιστροφές αν υπολογίσουμε εξαντλητικά τις

(β) Ο αλγόριθμος παραμένει ο ίδιος με τη διαφορά πως αν όταν φέρνουμε το στοιχείο που θέλουμε να ταξινομήσουμε στην 1η θέση είναι θετικό, τότε κάνουμε μία μοναδιαία προσημασμένη προθεματική περιστροφή και συνεχίζουμε μετά κανονικά. Τώρα κάνουμε το πολύ $3n$ περιστροφές.

(γ) 1. Κάθε συμβατό ζεύγος το αντικαθιστούμε με ένα στοιχείο του οποίου το πρόσημο ταυτίζεται με το κοινό πρόσημο των στοιχείων του ζεύγους και άρα καταλήγουμε σε πίνακα χωρίς συμβατό ζεύγος.
Αν υπάρχει στοιχείο με θετικό πρόσημο:

- Διαλέγουμε τον μεγαλύτερο, έστω m.
- Αν $m = n$ τότε με δύο κινήσεις το πάμε στο τέλος.
- Αλλιώς το $m+1$ είναι αρνητικό. Αν το $m+1$ είναι αριστερά τότε με δύο κινήσεις φτιάχνουμε το $(x, x+1)$ ενώ αν είναι δεξιά τότε $(-x-1, -x)$.

Αλλιώς:

- Υπάρχει x για το οποίο το $-x-1$ είναι αριστερότερα, (αλλιώς θα είχαμε τον $[-1, -2, \ldots, -n]$).
- Με 2 κινήσεις $([\ldots, -(x+1), \ldots, -x, \ldots] \to [x+1, \ldots, -x, \ldots] \to (-x-1, -x))$

2. 1 Απαλείφουμε όλα τα συμβατά ζεύγη πάλι όπως πριν, μέχρι να μην

2 Αν ο πίνακας είναι ο $[-1, -2, \ldots, -n']$, τότε:

- Επανάληψη n' φορές:
Εκτελούμε μία προσημασμένη προθεματική περιστροφή όλων των στοιχείων και μετά μία προθεματική περιστροφή των $n'-1$ αριστερότερων στοιχείων.

3 Αλλιώς κατασκευάζουμε σύμφωνα με το προηγούμενο ερώτημα ένα συμβατό ζεύγος.

Άσκηση 3: Υπολογισμός Κυρίαρχων Θέσεων

Θεωρούμε πίνακα με n φυσικούς αριθμούς. Για κάθε η θέση που κυριαρχεί της θέσης i στον πίνακα A είναι η πλησιέστερη θέση που προηγείται της i και η τιμή της ξεπερνά την τιμή $A[i]$. Τυπικά, θεωρώντας ότι $A[0] = \infty$, η θέση που κυριαρχεί της θέσης i στον πίνακα A είναι η μέγιστη θέση για την οποία ισχύει (βλ. ότι η θέση που κυριαρχεί της 1 είναι η 0). Να διατυπώσετε αποδοτικό αλγόριθμο που υπολογίζει τη θέση που κυριαρχεί της θέσης i, για κάθε , στον πίνακα . Να αιτιολογήσετε την ορθότητα και την υπολογιστική πολυπλοκότητα του αλγορίθμου σας.

Λύση:

Ορίζουμε s μία στοίβα, αρχικά κενή, στην οποία θα βάζουμε όσες θέσεις εμφανίζουν πτώση στην τιμή του στοιχείου που έχουν σε σχέση με την προηγούμενη θέση.
Διαβάζουμε τον πίνακα από το τέλος και πηγαίνουμε προς την αρχή.
Αν καθώς διαβάζουμε τον πίνακα ανάποδα βρίσκουμε $a[i-1] \geq a[i]$ βάζουμε ότι $dominant[i] = i-1$ και συγκρίνουμε το $a[i-1]$ και με τα στοιχεία της στοίβας, αν το $a[i-1]$ είναι μεγαλύτερο βάζουμε το $i-1$ ως κυρίαρχη θέση και για το εκάστοτε στοιχείο της στοίβας, μέχρι να βρούμε στοιχείο της στοίβας μεγαλύτερο από $a[i]$, που τότε προχωράμε στο επόμενο στοιχείο του πίνακα.
Αν $a[i-1] < a[i]$ βάζουμε το i στην στοίβα και προχωράμε.
Ο παραπάνω αλγόριθμος σε κάθε βήμα κάνει τον έλεγχο $a[i-1] > a[i]$, τον έλεγχο του $a[i-1]$ με την κορυφή της στοίβας, και πιθανώς μερικούς έξτρα ελέγχους όσο βγάζουμε στοιχεία από τη στοίβα. Οι έξτρα έλεγχοι είναι το πολύ n (η περίπτωση που όλα τα στοιχεία του πίνακα έχουν μπει στην στοίβα και φτάνουμε στο $a[0] = \infty$ για να μπει αυτό ως κυρίαρχο όλων των στοιχείων), αφού είναι το πολύ όσα και τα στοιχεία που θα μπουν συνολικά στην στοίβα. Άρα σε κάθε βήμα έχουμε δύο σίγουρους ελέγχους και άλλους n το πολύ συνολικά ελέγχους σε όλη τη διάρκεια του αλγορίθμου. Συνεπώς, η πολυπλοκότητα είναι $O(n)$.

```
int * solve(int a[],int n){
    stack s;
    int dominant[n];
    for(i=n-1;i>0;i--){
        if(a[i]<a[i-1]){
            dominant[i]=i-1;
            while(a[i-1]>a[s.top()]){
                dominant[s.top()]=i-1;
                s.pop();
                if(s.empty())break;
            }
        }
        else{
            s.put(i);
        }
    }

    return dominant;
}
```

Άσκηση 4: Φόρτιση Ηλεκτρικών Αυτοκινήτων

Διαχειριζόμαστε ένα σταθμό φόρτισης ηλεκτρικών αυτοκινήτων. Γνωρίζουμε ότι στη διάρκεια των επόμενων T χρονικών μονάδων πρόκειται να μας επισκεφθούν για φόρτιση n ηλεκτρικά αυτοκίνητα, καθώς επίσης και το χρόνο άφιξης a_i κάθε αυτοκινήτου i. Υποθέτουμε ότι από τη στιγμή που ένα αυτοκίνητο συνδέεται σε μία υποδοχή φόρτισης, η φόρτιση διαρκεί 1 χρονική μονάδα. Στα πρόσφατα διαφημιστικά μας μηνύματα, κυριαρχεί η υπόσχεση ότι η καθυστέρηση ενός αυτοκινήτου στον σταθμό μας δεν ξεπερνά τις d χρονικές μονάδες. Με άλλα λόγια, υποσχόμαστε ότι κάθε αυτοκίνητο i, ολοκληρώνει τη φόρτισή του μέχρτι τη χρονική στιγμή $a_i + d$. Δεδομένων του d και των χρόνων άφιξης $1 \leq a_1 \leq \cdots \leq a_n \leq T$ των n ηλεκτρικών αυτοκινήτων, θέλουμε να υπολογίσουμε το ελάχιστο πλήθος s^* υποδοχών φόρτισης που χρειάζονται για να τηρήσουμε την υπόσχεσή μας. Να διατυπώσετε αποδοτικό αλγόριθμο που υπολογίζει το s^*. Να αιτιολογήσετε την ορθότητα και την υπολογιστική πολυπλοκότητα του αλγορίθμου σας.

Λύση: Έστω m ο μέγιστος αριθμός αμαξιών που έρχονται ταυτόχρονα στο βενζινάδικο, τότε οι ελάχιστες αντλίες s^* είναι $\frac{a_{max}}{d} \leq s^* \leq n$. Οπότε κάνουμε δυαδική αναζήτηση στον διάστημα $[\frac{a_{max}}{d}, n]$ μέχρι να βρούμε τον ελάχιστο αριθμό αντλιών, ως εξής:

```
int solve(int a[],int d, int n, int amax){
    bool found = false;
    int l = amax/d;
    int r = n;
    while (1) {
        int m = l + (r - l) / 2;
        if (a[m] == x)
            return m;
        int p[n] ;// time the i-th car will start refueling
        for(int i = 0; i<n; i++){
            if(i < m)p[i]=a[i];
            else if ( 1 + max( 0, 1 + p[i-m] - a[i]) -d > 0 ){// no free
   oil pump
                l = m + 1;
                break;
            }
        }else{// there is a free oil pump
            r = m;
            p[i] = a[i] +max(0,1+p[i-m]-a[i]);
        }
        if ( l >= m ){
            reuturn m;
        }
    }
}
```

Χρονική πολυπλοκότητα $O(n\log(n))$, αφού θα επιλέξουμε το πολύ $\log(n)$ αριθμούς αντλιών και για τον καθένα κάνουμε το πολύ $O(n)$ ελέγχους.

Άσκηση 5: Επιλογή

(α) Έστω πολυσύνολο (multiset) S με n θετικούς ακεραίους που όλοι είναι μικρότεροι ή ίσοι δεδομένου ακεραίου M. Έχουμε πρόσβαση (μόνο) στην κατανομή F_S των στοιχείων της συλλογής. Συγκεκριμένα έχουμε στη διάθεσή μας συνάρτηση $F_S(l)$ που για κάθε φυσικό l, επιστρέφει το πλήθος των στοιχείων του S που δεν ξεπερνούν το l, δηλ. $F_S(l) = |\{x \in S : x \leq l\}|$. Να διατυπώσετε αποδοτικό αλγόριθμο που δέχεται ώς είσοδο φυσικό $k, 1 \leq k \leq n$, και υπολογίζει (καλώντας την F_S) το $k-$οστό μικρότερο στοιχείο του S. Να αιτιολογήσετε την ορθότητα του αλγορίθμου σας και να προσδιορίσετε το πλήθος των απαιτούμενων κλήσεων της F_S (στη χειρότερη περίπτωση). Προσπαθήστε το πλήθος το κλήσεων στην F_S να μην εξαρτάται από το n (μπορεί όμως να εξαρτάται από το M).

(β) Έστω πίνακας διαφορετικών θετικών ακεραίων $A[1 \ldots n]$ και έστω M το μέγιστο στοιχείο του A. Θεωρούμε το πολυσύνολο S που απότελείται από όλες τις μη αρνητικές διαφορές χευγών στοιχείων του A. Δηλαδή, έχουμε ότι:

$$S = \{A[i] - A[j] : i \neq j \text{ και } A[i] > A[j]\}$$

Να διατυπώσετε αποδοτικό αλγόριθμο που υπολογίζει το $k-$οστό μικρότερο στοιχείο του S. Να προσδιορίσετε την υπολογιστική πολυπλοκότητα του αλγορίθμου σας (συναρτήσει των n κια M) και να αιτιολογήσετε την ορθότητά του.

Λύση:

(α) Παρουσιάζουμε τον παρακάτω αλγόριθμο, σε $C++$, που βρίσκει τον μικρότερο αριθμό $first$ τέτοιο ώστε $F_s(first) \geq k$ (το οποίο είναι και το ζητούμενο).

```
int * solveA(int first, int last, int k){
    int d = last -  first; // length of interval of searching
    int i;
    int step=0;
    while(d>0){
        i = first;
        step = d/2;
        i += step;
        if(k >= fs[i]){
            first = i++; // search the upper half of the interval
            d -= step +1; // update the length of the interval
        }
        else{
            d = step; // half the interval down
        }
    }
    return first;
}

int main(){
    int answer = solve(0,M,k, fs[]);
}
```

(β) Ταξινομούμε τον πίνακα και κάνουμε δυαδική αναζήτηση στο διάστημα $[min_diff, a[n-1]-a[0]]$ χρησιμοποιώντας την συνάρτηση F_s.

```
int countPairs(int *a, int n, int mid){
    int res = 0;
    for (int i = 0; i < n; ++i)

        // solveA returns position
        // of next higher number than a[i]+mid in
        // a[i..n-1]. We subtract (a + i + 1) from
        // this position to count
        res += solveA(a+i, a+n, a[i] + mid) -
                                    (a + i + 1);
    return res;
}

// Returns k-th difference
int kthDiff(int a[], int n, int k){
    // Sort array
    sort(a, a+n);

    // Minimum absolute difference
    int low = a[1] - a[0];
    for (int i = 1; i <= n-2; ++i)
        low = min(low, a[i+1] - a[i]);

    // Maximum absolute difference
    int high = a[n-1] - a[0];

    // Do binary search for k-th absolute difference
    while (low < high)
    {
        int mid = (low+high)>>1;
        if (countPairs(a, n, mid) < k)
            low = mid + 1;
        else
            high = mid;
    }

    return low;
}
```