Άσκηση 1 – Αλγόριθμοι

Στοιχεία:

Ονοματεπώνυμο : Παντελίδης Ιπποκράτης

Αριθμός Μητρώου : p3210150

Άσκηση 1)

Οι συναρτήσεις αναγράφονται με αύξουσα σειρά πολυπλοκότητας δηλαδή με τέτοιο τρόπο ώστε να ισχύει $f_i = O(f_{i+1})$ για i = 1,…,10 δηλαδή όσο το πλήθος των συναρτήσεων πλην της τελευταίας.

$\sqrt{\log_9 n}$, $\log\sqrt{n}$, $2^{\log n/2}$, $\log(4^n)$, $\log(n^n)$, $(\log(2^n))^2$, $8^{\log n}$, 2^n , $n^{\log n}$, $n!$, $2^{n\log n}$

Άσκηση 2)

(α’) Χρησιμοποιώντας τον ορισμό του big O πρέπει να δείξουμε ότι $n \cdot 2^n = O(3^n)$ => να βρούμε ότι ∃ σταθερά $c > 0$, ∃ φυσικός αριθμός $n_0 > 0$ τέτοιοι ώστε να ισχύει η ανισότητα $n \cdot 2^n <= c \cdot 3^n$, ∀ $n > n_0$

Θα την λύσουμε χρησιμοποιώντας επαγωγή. Ειδικότερα:

- Επαγωγική Βάση : Για n = 1 στην σχέση μας έχουμε ότι $1 \cdot 2^1 = 2 <= c \cdot 3^1$ που ισχύει για οποιοδήποτε c αφού $c > 0$ και έστω για $n_0 = 1$

- Επαγωγική Υπόθεση : Υποθέτουμε λοιπόν ότι η ανισότητα μας ισχύει για κάποιο αυθαίρετο n = k, επομένως ισχύει $k \cdot 2^k <= c \cdot 3^k$

- Επαγωγικό Βήμα : Θα πρέπει να δείξουμε χρησιμοποιώντας την επαγωγική υπόθεση ότι η ανισότητα ισχύει επίσης και για n = k + 1. Επομένως να δείξουμε ότι : $(k + 1) \cdot 2^{k+1} <= c' \cdot 3^{k+1}$

Ξεκινώντας με το αριστερό μέλος της ανισότητας έχουμε ότι :

Σχέση (1) : $(k + 1) \cdot 2^{k+1} = k \cdot 2^{k+1} + 2^{k+1} = 2 \cdot k \cdot 2^k + 2^{k+1} <= 2 \cdot c \cdot 3^k + 2 \cdot 2^k <=$

$<= 2 \cdot c \cdot 3^k + 2 \cdot 3^k = (2/3) \cdot c \cdot 3^{k+1} + (2/3) \cdot 3^{k+1} = (2/3) \cdot 3^{k+1} \cdot (c + 1) =$

$(2/3) \cdot (c + 1) \cdot 3^{k+1}$

Επομένως μπορούμε να πάρουμε ως c' την σταθερά $(2/3) \cdot (c + 1)$ και έτσι η απόδειξη ολοκληρώθηκε αφού ξεκινώντας από το αριστερό μέλος της ανισότητας καταλήγουμε στο δεξί.

Σημειώσεις :

- Χρησιμοποιούμε τον τόνο πάνω από το (c') για να δηλώσουμε ότι είναι διαφορετική σταθερά από αυτήν της επαγωγικής υπόθεσης (c).
- Η πρώτη ανισότητα στην σχέση (1) προκύπτει χρησιμοποιώντας την επαγωγική υπόθεση.
- Η δεύτερη ανισότητα στην σχέση (1) προκύπτει από το γεγονός ότι $2^k <= 3^k$, που ισχύει αφού $2 < 3 => 2^k <= 3^k \ \forall \ k >= 0$ με την ισότητα να ισχύει για k = 0.
- Η τρίτη ισότητα στην σχέση (1) προκύπτει διαιρώντας και πολλαπλασιάζοντας με 3 ώστε να σχηματιστεί το 2/3 και αντίστοιχα το 3^{k+1}.

(β') Όμοια με το (α') ερώτημα πρέπει χρησιμοποιώντας τον ορισμό του big O να δείξουμε ότι $n^2 \cdot 2^n + 10 = O(3^n)$ => να βρούμε ότι ∃ σταθερά $c > 0$, ∃ φυσικός αριθμός $n_0 > 0$ τέτοιοι ώστε να ισχύει η ανισότητα $n^2 \cdot 2^n + 10 <= c \cdot 3^n, \ \forall \ n > n_0$

Θα την λύσουμε και αυτήν χρησιμοποιώντας επαγωγή. Ειδικότερα :

- Επαγωγική βάση : Παρατηρούμε ότι αν αντικαταστήσουμε n = 1 στην σχέση μας έχουμε : $1^2 \cdot 2^1 + 10 <= c \cdot 3^1 => 1 \cdot 2 + 10 <= c \cdot 3 => 12 <= c \cdot 3$. Επομένως παίρνοντας $c >= 4$ και έστω $n_0 = 1$ ισχύει η ανισότητα μας.

- Επαγωγική Υπόθεση : Υποθέτουμε λοιπόν ότι η ανισότητα μας ισχύει για κάποιο αυθαίρετο n = k, επομένως ισχύει $k^2 \cdot 2^k + 10 <= c \cdot 3^k$

- Επαγωγικό Βήμα : Θα πρέπει να δείξουμε χρησιμοποιώντας την επαγωγική υπόθεση ότι η ανισότητα ισχύει επίσης και για n = k + 1. Επομένως να δείξουμε ότι : $(k + 1)^2 \cdot 2^{k+1} + 10 <= c' \cdot 3^{k+1}$

Ξεκινώντας με το αριστερό μέλος της ανισότητας έχουμε ότι :

Σχέση (2) : $(k + 1)^2 \cdot 2^{k+1} + 10 = (k^2 + 2k + 1) \cdot 2^{k+1} + 10 = (k^2 + 2k + 1) \cdot 2^k \cdot 2 + 10 =$

$= (2 \cdot k^2 + 4 \cdot k + 2) \cdot 2^k + 10 = 2 \cdot k^2 \cdot 2^k + 4 \cdot k \cdot 2^k + 2 \cdot 2^k + 10 = 2 \cdot (k^2 \cdot 2^k + 5)$

$+ 4 \cdot k \cdot 2^k + 2 \cdot 2^k <= 2 \cdot c_1 \cdot 3^k + 4 \cdot k \cdot 2^k + 2 \cdot 2^k <= 2 \cdot c_1 \cdot 3^k + 4 \cdot c_2 \cdot 3^k + 2 \cdot 2^k <=$

$<= 2 \cdot c_1 \cdot 3^k + 4 \cdot c_2 \cdot 3^k + 2 \cdot c_3 \cdot 3^k = (2/3) \cdot c_1 \cdot 3^{k+1} + (4/3) \cdot c_2 \cdot 3^{k+1} + (2/3) \cdot c_3 \cdot 3^{k+1}$

$= (2 \cdot c_1 + 4 \cdot c_2 + 2 \cdot c_3)/3 \cdot 3^{k+1}$

Επομένως μπορούμε να πάρουμε ως c’ την σταθερά $(2 \cdot c_1 + 4 \cdot c_2 + 2 \cdot c_3)/3$ και έτσι η απόδειξη ολοκληρώθηκε αφού ξεκινώντας από το αριστερό μέλος της ανισότητας καταλήγουμε στο δεξί.

Σημειώσεις :

- Χρησιμοποιούμε τον τόνο πάνω από το (c’) για να δηλώσουμε ότι είναι διαφορετική σταθερά από αυτήν της επαγωγικής υπόθεσης (c).
- Η πρώτη ισότητα στην σχέση (2) προκύπτει απλά «ανοίγοντας» την ταυτότητα και οι επόμενες δύο εφαρμόζοντας την επιμεριστική ιδιότητα.
- Η έκτη ισότητα προκύπτει κάνοντας παραγοντοποίηση ώστε να εμφανιστεί η ανισότητα της επαγωγικής μας υπόθεσης.
- Η πρώτη ανισότητα προκύπτει χρησιμοποιώντας την επαγωγική μας υπόθεση.
- Η δεύτερη ανισότητα προκύπτει χρησιμοποιώντας αυτό που αποδείξαμε στο προηγούμενο ερώτημα (α’) της άσκησης.
- Η τρίτη ανισότητα προκύπτει χρησιμοποιώντας την τρίτη σημείωση του προηγούμενου ερωτήματος της άσκησης.
- Όμοια πάλι με τον προηγούμενο ερώτημα της άσκησης διαιρούμε και πολλαπλασιάζουμε με 3 ώστε να σχηματιστεί στις τελευταίες ισότητες οι 2/3, 4/3, 2/3 καθώς αντίστοιχα και το 3^{k+1}.

Άσκηση 3)

Γενικά : Η συνάρτηση μας εκτυπώνει τους αριθμούς από το 1 μέχρι και το όρισμα n που της δίνεται και καλεί αναδρομικά τον εαυτό της δύο φορές με όρισμα κάθε φορά τον μεγαλύτερο ακέραιο που είναι μικρότερος ίσος από το n/2.Η συνάρτηση

εκτελείται έως ότου το n να γίνει μικρότερο του 1.Επομένως μπορούμε να συμπεράνουμε ότι η συνάρτηση μας ακολουθεί την λογική του Divide & Conquer(Διαίρει και Βασίλευε),και έτσι μπορούμε να χρησιμοποιήσουμε την αναδρομική σχέση $T(n) = a \cdot T(n/b) + O(n^d)$ με διαφορετικές τιμές παραμέτρων ανάλογα με τον πρόβλημα που θέλουμε να λύσουμε.

(α') Για να βρούμε το πλήθος των αριθμών(όχι απαραίτητα διαφορετικών μεταξύ τους) που εκτυπώνονται αφού κληθεί η συνάρτηση f(n) για κάποιο n >= 2, μπορούμε να χρησιμοποιήσουμε την αναδρομική σχέση $T(n) = 2 \cdot T(n/2) + n$ όπου:

- T(n) είναι το πλήθος των αριθμών που εκτυπώνονται με είσοδο n
- a = 2 αφού το πρόβλημα μας χωρίζεται σε δύο υποπροβλήματα(δύο αναδρομικές κλήσεις)
- b = 2 αφού το μέγεθος του ορίσματος διαιρείται με 2 σε κάθε αναδρομική κλήση
- d = 1 αφού το κάθε υποπρόβλημα έχει ένα loop που εκτελείται n και επομένως θα πρέπει να έχει πολυπλοκότητα O(n),δηλαδή $O(n^1)$.

Έτσι αφού έχουμε βρει την αναδρομική σχέση μπορούμε τώρα να υπολογίσουμε την πολυπλοκότητα του προβλήματος χρησιμοποιώντας το Master Theorem.Αφού ισχύει ότι $b^d = a$ ($2^1 = 2$) ή ισοδύναμα $d = \log_b a$ ($1 = \log_2 2$) βρισκόμαστε στην $2^η$ περίπτωση του θεωρήματος, επομένως η πολυπλοκότητα του αλγορίθμου είναι $O(n^d \cdot \log_b n)$,δηλαδή $O(n^1 \cdot \log_2 n) = O(n \cdot \log n)$.

(β') Όμοια για να βρούμε το πλήθος των άσσων που εκτυπώνονται αφού κληθεί η συνάρτηση f(n) για κάποιο n >= 2 μπορούμε να χρησιμοποιήσουμε την αναδρομική σχέση $T(n) = 2 \cdot T(n/2) + 1$ όπου:

- T(n) είναι ο αριθμός των άσσων που εκτυπώνονται με είσοδο n.
- a = 2 αφού το πρόβλημα μας χωρίζεται σε δύο υποπροβλήματα(δύο αναδρομικές κλήσεις).
- b = 2 αφού το μέγεθος του ορίσματος διαιρείται με 2 σε κάθε αναδρομική κλήση.
- d = 0 αφού για κάθε υποπρόβλημα μας ενδιαφέρει η εκτύπωση μόνο του πρώτου αριθμού εντός του loop και όχι όλων των αριθμών μέχρι και το n θα πρέπει να έχει πολυπλοκότητα O(1), δηλαδή $O(n^0)$.

Έτσι παρόμοια με το προηγούμενο ερώτημα αφού έχουμε βρει την αναδρομική σχέση μπορούμε τώρα να υπολογίσουμε την πολυπλοκότητα του προβλήματος χρησιμοποιώντας το Master Theorem.Αφού ισχύει ότι $b^d < a$ ($2^0 < 2 = 1 < 2$) ή ισοδύναμα $d < \log_b a$ ($0 < \log_2 2$) βρισκόμαστε στην $3^η$ περίπτωση του θεωρήματος, επομένως η πολυπλοκότητα του αλγορίθμου είναι $O(n^{\log_b a})$, δηλαδή $O(n^{\log_2 2}) = O(n)$.

Άσκηση 4)

(α') Σύμφωνα με τον αλγόριθμο που μας δίνεται στην εκφώνηση ξέροντας ότι το x είναι πλειοψηφικό στοιχείο του πίνακα A τότε θα εμφανίζεται σε αυτόν περισσότερες από n/2 φορές όπου n είναι το μέγεθος του πίνακα A. Έστω k οι φορές που το στοιχείο x εμφανίζεται στον A και ισχύει k > n/2. Μετά την πρώτη κλήση του αλγορίθμου θα σχηματιστούν στον A ⌊n/2⌋ ζεύγη και επομένως επειδή n είναι άρτιος n/2 ζεύγη. Το μέγεθος του πίνακα B το οποίο θα συμβολίσουμε |B| μπορεί να είναι έως και n/2 ακριβώς στοιχεία στην περίπτωση που όλα τα στοιχεία του A είναι σε ζευγάρια με ίδια στοιχεία. Έτσι ισχύει ότι |B| <= n/2(σχέση (1)). Από την άλλη πλευρά μπορούμε να συμπεράνουμε ότι το x θα εμφανίζεται στον B ⌊k/2⌋, στην περίπτωση που σε όλες τις εμφανίσεις του στον A είναι σε ένα ζευγάρι με μόνο x. Επομένως για να αποδείξουμε ότι το x είναι πλειοψηφικό στοιχείο του πίνακα B πρέπει να δείξουμε ότι οι εμφανίσεις του x στον B είναι περισσότερες από τα μισά στοιχεία του B, δηλαδή ότι ⌊k/2⌋ > |B|/2, το οποίο αποδεικνύεται ξεκινώντας από την σχέση (1)=>|B| <= n/2 < k => |B|/2 < k/2 => |B|/2 < ⌊k/2⌋ και η απόδειξη ολοκληρώθηκε.

(β') Στην περίπτωση που το μέγεθος n του πίνακα A είναι περιττός αριθμός και ξέροντας ότι το x είναι πλειοψηφικό στοιχείο μπορούμε πάλι να συμπεραίνουμε ότι αυτό εμφανίζεται στον πίνακα A πάνω από n/2 φορές. Μετά την πρώτη κλήση του αλγορίθμου στον A είτε θα βρούμε αμέσως το πλειοψηφικό στοιχείο του A με τον έλεγχο του περισσευούμενου στοιχείου είτε θα καταλήγαμε στον πίνακα B με το πολύ ⌊n/2⌋ στοιχεία εάν το περισσευούμενο στοιχείο δεν είναι το πλειοψηφικό του A. Έτσι στην συνέχεια της κλήσης δεν απασχολούμαστε άλλο με το συγκεκριμένο και έτσι επικεντρωνόμαστε σε έναν υποπίνακα του A με άρτιο μέγεθος n.Αυτό έχει σαν αποτέλεσμα ο αλγόριθμος να τρέχει σε αυτόν τον πίνακα και συμβαδίζουμε με την περίπτωση όπου το μέγεθος του πίνακα είναι άρτιος.

Επομένως η απόδειξη ολοκληρώνεται χρησιμοποιώντας την απόδειξη του ερωτήματος (α').

(γ') Ένας «διαίρει και βασίλευε» αλγόριθμος σε ψευδογλώσσα που χρησιμοποιεί την ιδέα της εκφώνησης είναι ο παρακάτω :

Input: Ο αλγόριθμος παίρνει σαν είσοδο έναν πίνακα A με n στοιχεία, όπου n >=1
Output: Εκτυπώνει το πλειοψηφικό στοιχείο του A ή None αν δεν υπάρχει

```
FUNCTION majority_element(A):
n <- μέγεθος του πίνακα A
IF n = 1 THEN
    RETURN A[0]
END IF
pairs <- [(A[i], A[i+1]) FOR i <- 0 to n-1 step 2]
IF n is odd THEN
    remaining <- A[-1]
ELSE
    remaining <- None
END IF
B <- [ ]
FOR pair IN pairs DO
     IF pair[0] = pair[1] THEN
          B.append(pair[0])
     END IF
END FOR
IF remaining is not None AND A.count(remaining) > n/2 THEN
    RETURN remaining
ELSE IF length(B) > 0 THEN
    majority_B <- majority_element(B)
    IF majority_B is not None and A.count(majority_B) > n/2 THEN
         RETURN majority_B
     ELSE
          RETURN None
     END IF
ELSE
    RETURN None
END IF
END FUNCTION
```

Βήματα που ακολουθεί ο αλγόριθμος κατά την κλήση του :
1)Υπολόγισε το μέγεθος του πίνακα A και βάλε το σε μία μεταβλητή n.
2)Αν n == 1 εμφάνισε το στοιχείο του A επειδή είναι πλειοψηφικό.
3)Χώρισε τον πίνακα A σε ζευγάρια και βάλε τα σε έναν άλλο πίνακα με όνομα pairs έτσι ώστε το κάθε ζευγάρι να είναι ένα tuple,(i$^{-οστό}$ στοιχείο, i+1$^{-οστό}$στοιχείο).
4)Αν το μέγεθος του πίνακα n είναι περιττός τότε μετά τον χωρισμό σε ζευγάρια θα περισσεύει το τελευταίο στοιχείο. Βάλε αυτό το στοιχείο στην μεταβλητή remaining, ενώ αν n άρτιος δεν περισσεύει κανένα στοιχείο και βάλε στην μεταβλητή None.
5)Τα παραπάνω βήματα αναφέρονταν στον χωρισμό του προβλήματος(divide) και το επόμενο θα ασχοληθεί με την «κατάκτηση» (conquer). Αρχικοποίησε τον πίνακα B του επόμενου γύρου ως κενό.
6)Διάσχισε τον πίνακα pairs που φτιάχτηκε παραπάνω και για κάθε ζευγάρι που έχει δύο όμοια στοιχεία πάρε το ένα από αυτά(pair[0] ή pair[1]) και πρόσθεσε τα στον πίνακα B. Αν δεν είναι ίδια προσπέρασε τα.
7)Τώρα γίνεται η σύνδεση των παραπάνω(combine).Αν υπάρχει remaining στοιχείο και αυτό εμφανίζεται στον πίνακα A περισσότερες από n/2 φορές στον A τότε είναι πλειοψηφικό στοιχείο του A και επιστρέφεται.
8)Αν δεν υπάρχει, ο αλγόριθμος επαναλαμβάνεται αναδρομικά στη νέα λίστα B που περιέχει όλα τα στοιχεία που εμφανίστηκαν δυο φορές στο ίδιο ζευγάρι στο προηγούμενο βήμα. Εάν το πλειοψηφικό στοιχείο υπάρχει στη νέα λίστα B, τότε πρέπει να είναι το ίδιο με το πλειοψηφικό στοιχείο στην αρχική λίστα A. Αυτό ελέγχεται μετρώντας πόσες φορές το πλειοψηφικό στοιχείο στο B εμφανίζεται στο A. Εάν το πλειοψηφικό στοιχείο στο B εμφανίζεται περισσότερες από n/2 φορές στο A, τότε επιστρέφεται ως το πλειοψηφικό στοιχείο. Διαφορετικά, ο αλγόριθμος επιστρέφει None, υποδηλώνοντας ότι δεν υπάρχει πλειοψηφικό στοιχείο στην αρχική λίστα A.
9)Αν ο πίνακας B έχει μείνει κενός τότε δεν υπάρχει πλειοψηφικό στοιχείο στον A και επίστρεψε None.

(δ’) Για τον υπολογισμό του χρόνου από το Master Theorem έχουμε :

- a = 1 : αφού έχουμε ένα μόνο υποπρόβλημα(1 αναδρομική κλήση).
- b = 2 : αφού το μέγεθος του πίνακα υποδιπλιάζεται στην χειρότερη περίπτωση σε κάθε αναδρομική κλήση.
- d = 1 : καθώς θέλουμε O(n) για το ένα μέτρημα και O(1) για τις συγκρίσεις και τα return.

Επομένως προκύπτει η αναδρομική σχέση $T(n) = T(n/2) + O(n)$ αντικαθιστώντας τις τιμές a, b, d στην $T(n) = a \cdot T(n/b) + O(n^d)$ και μπορούμε να βρούμε τώρα την πολυπλοκότητα του αλγορίθμου χρησιμοποιώντας το Master Theorem.Αφού ισχύει ότι $b^d > a$ ($2^1 > 1$) ή ισοδύναμα $d > \log_b a$ ($1 > \log_2 1 = 0$) βρισκόμαστε στην $1^η$ περίπτωση του θεωρήματος επομένως η πολυπλοκότητα του αλγορίθμου είναι $O(n^d) = O(n^1) = O(n)$.

Άσκηση 5)

(α') Ο αλγόριθμος μας ξεκινάει αρχικοποιώντας ένα πίνακα n θέσεων, η κάθε θέση του οποίου αντιπροσωπεύει ένα κουτί στο οποίο μπορούν να μπουν μέχρι 20 μπάλες έως και δύο διαφορετικών χρωμάτων. Σύμφωνα με την εκφώνηση ο αλγόριθμος παίρνει σαν όρισμα έναν πίνακα A[1…n] με τον αριθμό των μπάλων κάθε χρώματος σε κάθε θέση. Στην συνέχεια ταξινομούμε τον πίνακα A σε φθίνουσα σειρά έτσι ώστε στην αρχή του να βρίσκεται το χρώμα με τον μεγαλύτερο αριθμό από μπάλες. Έπειτα ξεκινάμε να διασχίζουμε τον πίνακα A από το τέλος του προς την αρχή του, για παράδειγμα με ένα for loop από το n – 1 μέχρι το 0 με βήμα -1.Σε κάθε επανάληψη βλέπουμε εάν το $i^{οστό}$ στοιχείο χωράει στο $i^{οστό}$ κουτί. Αν χωράει ακριβώς(σπάνια περίπτωση όπου όλοι τα χρώματα έχουν το ίδιο πλήθος από μπάλες) τότε συνεχίζουμε στο for loop.Αν χωράει στο κουτί άλλα το κουτί δεν έχει γεμίσει τότε συμπληρώνουμε σε αυτό το κουτί από τις μπάλες του πρώτου μέχρι να μην χωράει κάποια άλλη. Και τέλος αν ο αριθμός των μπαλών του $i^{οστού}$ στοιχείου του A χωρέσει ολόκληρος και περισσεύει(δεν υπάρχει περίπτωση να συμβεί στην $1^η$ επανάληψη) τότε γεμίζουμε το $i^{οστό}$ κουτί όπως την πρώτη περίπτωση και όσα απομένουν τα βάζουμε στο $(i – 1)^{οστό}$ κουτί. Σε κάθε περίπτωση πριν περάσουμε στην επόμενη επανάληψη ενημερώνουμε τον πίνακα A, δηλαδή κάνουμε 0 το $i^{οστό}$ στοιχείο(αφού έχουμε κατανείμει όλες τις μπάλες αυτού του χρώματος) και μειώνουμε επίσης το πρώτο στοιχείο του πίνακα A σε περίπτωση που χρειάστηκε να συμπληρώσει μπάλες για το γέμισμα κάποιου κουτιού. Αφού ολοκληρωθεί το for loop ο πίνακας κατανομής των χρωματιστών μπαλών έχει σχηματιστεί με τον τρόπο που θέλουμε οπότε απλά τον επιστρέφουμε.

(β') Για να αποδείξουμε ότι η λύση που βρίσκει ο αλγόριθμος μας είναι σωστή πρέπει να δείξουμε ότι πάντα παράγει μία έγκυρη τοποθέτηση μπαλών σε κουτιά,

δηλαδή τέτοια ώστε το κάθε κουτί να περιλαμβάνει το πολύ 2 διαφορετικά χρώματα μπαλών. Για να το αποδείξουμε αυτό θα χρησιμοποιήσουμε μαθηματική επαγωγή. Ειδικότερα έχουμε :

- Επαγωγική Βάση : Παρατηρούμε ότι όταν έχουμε n = 2, δηλαδή 2 κουτιά και 2 διαφορετικά χρώματα όπως το $1^{ο}$ παράδειγμα της άσκησης 5,ο περιορισμός που δίνει η εκφώνηση μας ικανοποιείται όπως και αν αποφασίσουμε να χωρίσουμε τις 2·20 μπάλες στα 2 κουτιά, καθώς κάθε κουτί μπορεί να έχει το πολύ δύο διαφορετικά χρώματα.
- Επαγωγική Υπόθεση : Υποθέτουμε λοιπόν ότι η ιδιότητα αυτή ισχύει και για n = k κουτιά, δηλαδή υποθέτουμε ότι έχουμε διανείμει 20·k μπάλες k διαφορετικών χρωμάτων σε k κουτιά έτσι ώστε το κάθε κουτί να περιέχει το πολύ δύο διαφορετικά χρώματα.
- Επαγωγικό Βήμα : Αρκεί να δείξουμε τώρα χρησιμοποιώντας την επαγωγική υπόθεση ότι η ιδιότητα μας ισχύει και για n = k + 1. Αν τρέξουμε τον αλγόριθμο μας στα k + 1 κουτιά ξέρουμε μέσω της επαγωγικής υπόθεσης ότι θα δημιουργηθούν k (1,…,k) κουτιά έχοντας σωστά την ιδιότητα, τα οποία θα περιέχουν αντίστροφα τις αντίστοιχες μπάλες των τελευταίων k κουτιών μας, μαζί με συμπλήρωμα από τον $1^{ο}$ στοιχείο αν χρειαστεί. Δηλαδή το $1^{ο}$ κουτί θα περιέχει τις μπάλες του της τελευταίας θέσης του πίνακα μαζί με το συμπλήρωμα της πρώτης θέσης του πίνακα(αν χρειαστεί). Το $2^{ο}$ κουτί θα περιέχει μπάλες της προτελευταίας θέσης του πίνακα μαζί με συμπλήρωμα του πρώτου(αν χρειαστεί) και συνεχίζει με τον δεύτερο στοιχείο του πίνακα. Έτσι το πρώτο στοιχείο του πίνακα που ήταν αρχικά το μεγαλύτερο είτε θα έχει φτάσει με τις διαδοχικές μειώσεις κάτω από 20 οπότε θα μπει στο $k+1^{-οστό}$ κουτί μαζί με το περίσσευμα του $2^{ου}$ κουτιού είτε θα μείνει ακριβώς 20 και θα αποτελέσει το ίδιο από μόνο του το $k+1^{-οστό}$ κουτί. Και στις δύο περιπτώσεις συνεχίζει να ισχύει η ιδιότητα μας, οπότε αφού δείξαμε ότι ισχύει και για k + 1 η απόδειξη ολοκληρώθηκε.

(γ') Η πολυπλοκότητα του αλγορίθμου εξαρτάται από την πολυπλοκότητα του αλγορίθμου ταξινόμησης που χρησιμοποιούμε καθώς και από τον αριθμό των επαναλήψεων του for loop. Έστω ότι επιλέγουμε την γρηγορότερη πολυπλοκότητα αλγορίθμου ταξινόμησης που είναι O(n·logn).Για το πλήθος των επαναλήψεων, βρίσκουμε ότι το for loop τρέχει n φορές ,και εφόσον μας δίνεται από την εκφώνηση ότι το γέμισμα ενός κουτιού με 20 μπάλες καθώς και οι πράξεις της

πρόσθεσης και αφαίρεσης γίνονται σε ένα βήμα με χρόνο O(1),μπορούμε να συμπεράνουμε ότι είναι O(n).Επομένως συνδυάζοντας τα βρίσκουμε ότι η συνολική πολυπλοκότητα του αλγορίθμου είναι $O(n \cdot \log n) + O(n) = O(n \cdot \log n)$.