<u>Ευθυμίου Βασίλειος,ics21035</u>

Ο Αλγόριθμος ονομάζεται **Median of medians** και έχει υπολογιστική πολυπλοκότητα **O(n)**:(κάτω περιλαμβάνονται κάποιες πληροφορίες για αυτόν τον αλγόριθμο)

Median of Medians

Class	Selection algorithm
Data structure	Array
Worst-case performance	$O(n)$
Best-case performance	$O(n)$
Worst-case space complexity	$O(\log n)$ auxiliary

<u>Βήματα του αλγορίθμου:</u>

Ο αλγόριθμος αυτός(αναφορά ως ΕΠΙΛΟΓΗ και παρακάτω) προσδιορίζει το i-οστό μικρότερο στοιχείο μίας συστοιχίας εισόδου με n>1(όπου n το πλήθος των στοιχείων) διαφορετικά στοιχεία ακολουθώντας τα ακόλουθα βήματα:(Εάν n=1, η ΕΠΙΛΟΓΗ απλώς επιστρέφει ως i-οστή μικρότερη τιμή το μοναδικό στοιχείο εισόδου της.)

1. Διαιρούμε τα n στοιχεία της συστοιχίας εισόδου σε floor(n/5) ομάδες των πέντε στοιχείων η κάθε μια και το πολύ μία ακόμα ομάδα αποτελούμενη από τα υπόλοιπα n mod 5 στοιχεία.

2. Βρίσκουμε τον διάμεσο κάθε μίας από τις floor(n/5) ομάδες, αρχικά ταξινομώντας με την insertion sort τα στοιχεία της κάθε ομάδας και στη συνέχεια επιλέγοντας τον διάμεσο από την ταξινομημένη λίστα των στοιχείων της ομάδας.(Σημείωση:Η ταξινόμηση πολυ μικρών λιστών όπως εδώ παίρνει γραμμικό χρόνο αφού αυτές οι υπολίστες έχουν 5 στοιχεία το πολύ και αυτό παίρνει O(n) χρόνο.

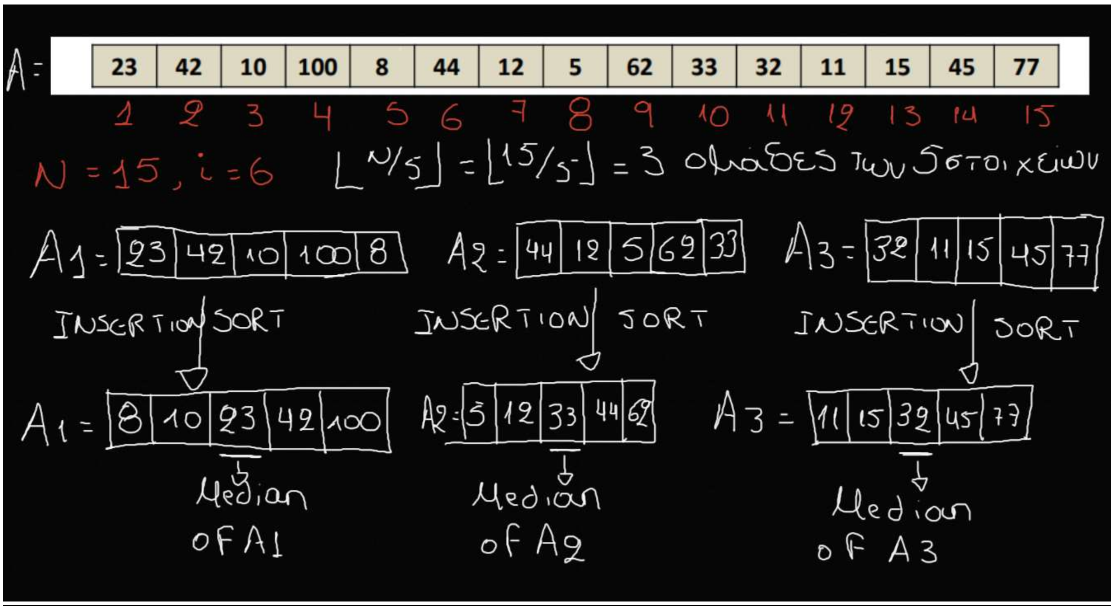
3. Χρησιμοποιώντας την διαδικασία/αλγόριθμο ΕΠΙΛΟΓΗ αναδρομικά, βρίσκουμε τον διάμεσο x των ceiling(n/5) διαμέσων που προσδιορίστηκαν στο βήμα 2. (ΠΡΟΣΟΧΗ,αν το πλήθος των διαμέσων είναι άρτιο, τότε κατά τη σύμβασή μας, ο x είναι ο κάτω διάμεσος. Ο διάμεσος σε μία ομάδα με άρτιο αριθμό στοιχείων θα είναι ο αριστερός διάμεσος(σύμφωνα με τα περισσότερα παραδείγματα σε βιβλία και ιστοσελίδες).Ωστόσο,είναι απόλυτα αποδεκτή η επιλογή του δεξιού διάμεσου ενόσω αυτή η στρατηγική ακολουθείται συνεχώς σε όλο τον αλγόριθμο.)

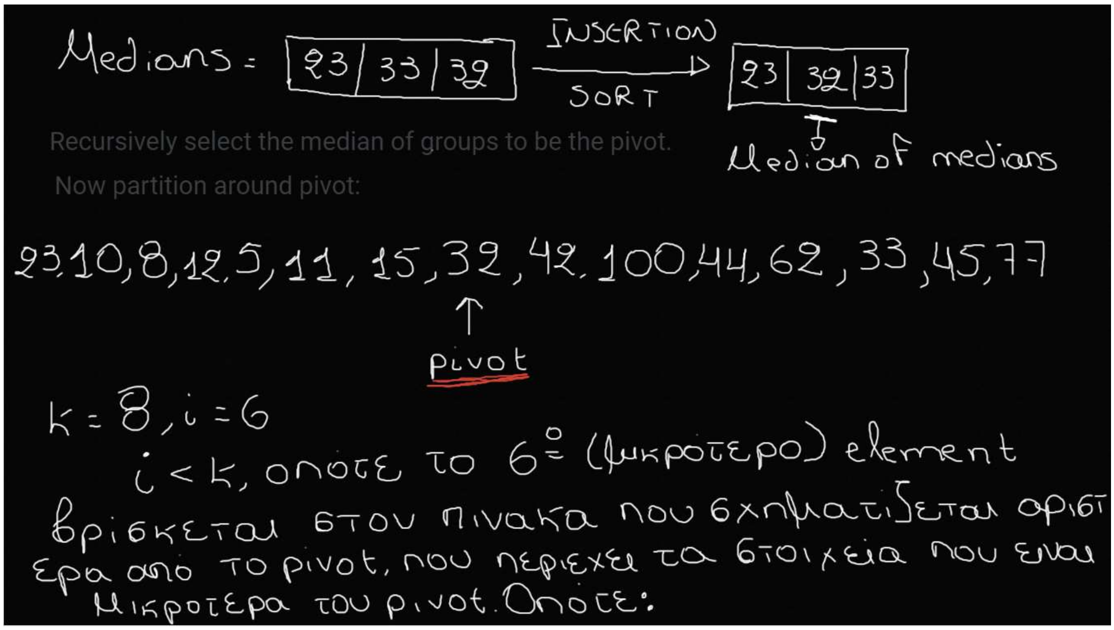
4. Διαμερίζουμε τη συστοιχία εισόδου γύρω από τον διάμεσο των διαμέσων x χρησιμοποιώντας την modified version of Partition**(1).**Ορίζουμε το k ίσο με το πλήθος των στοιχείων στο χαμηλότιμο σκέλος της διαμέρισης σύν ένα καθώς

σύμφωνα με τον ορισμό αυτόν το x είναι το k-οστό μικρότερο στοιχείο και υπάρχουν n-k στοιχεία στο υψηλότιμο σκέλος της διαμέρισης.

5. Εάν i=k,δίνουμε ως αποτέλεσμα το x. Διαφορετικά, χρησιμοποιούμε τη διαδικασία ΕΠΙΛΟΓΗ αναδρομικά για να προσδιορίσουμε το i-οστό μικρότερο στοιχείο στο χαμηλότιμο σκέλος εάν i<k,ή το (i-k)-οστό μικρότερο στοιχείο στο υψηλότιμο σκέλος εάν i>k.

<u>Εξήγηση λειτουργίας στιγμιοτύπου:</u>





Ls (Left Subarray):

$Ls =$

23	10	8	12	5	11	15

$N = 7$, $i = 6$, $\lfloor \frac{N}{5} \rfloor = \lfloor \frac{7}{5} \rfloor =$ 2 ομάδες

We use this algorithm recursively in order to find the 6th smallest element.

$LS1 =$

23	10	8	12	5

INSERTION SORT

$LS1 =$

5	8	10	12	23

Median of LS1

$LS2 =$

11	15

INSERTION SORT

$LS2 =$

11	15

Median of LS2

Recursively select the median of groups to be the pivot

Medians =

10	11

$\xrightarrow[\text{SORT}]{\text{INSERTION}}$

10	11

→ Median of Medians

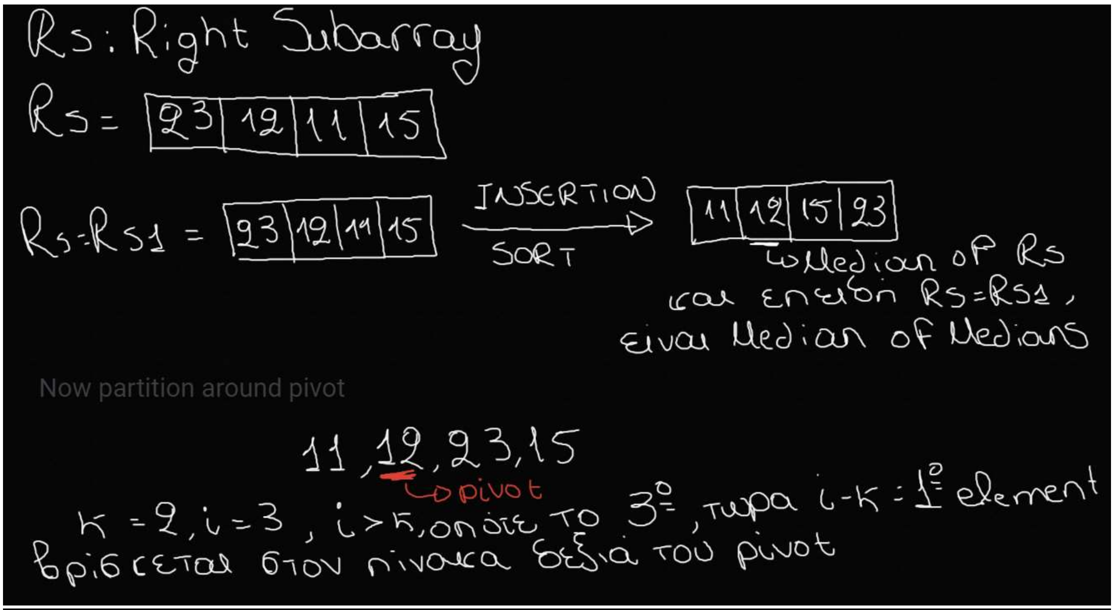
Now partition around pivot

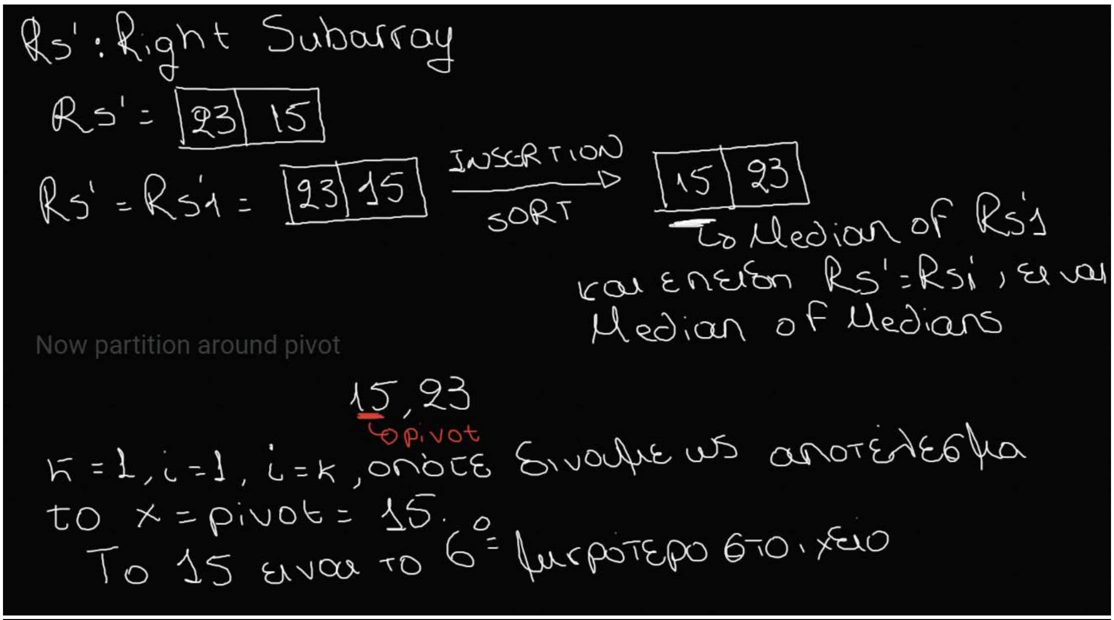
8, 5, 10, 23, 12, 11, 15

→ pivot

$k = 3$, $i = 6$, $i > k$, οπότε το 6ο element βρίσκεται στον πίνακα που σχηματίζεται δεξιά του pivot, που περιέχει τα στοιχεία που είναι μεγαλύτερα του pivot, οπότε:

(πλέον ψάχνουμε το $i - k = 6 - 3 = 3$ο στοιχείο στον δεξί υποπίνακα)





Τρόπος υπολογισμού πολυπλοκότητας:

Για να αναλύσουμε τον χρόνο εκτέλεσης του αλγορίθμου, αρχικά προσδιορίζουμε ένα κάτω φράγμα για το πλήθος των στοιχείων που είναι μεγαλύτερα από το στοιχείο διαμέρισης x. Τουλάχιστον οι μισοί από τους διαμέσους που προσδιορίζονται στο βήμα 2 είναι μεγαλύτεροι του διαμέσου των διαμέσων x ή ίσοι με αυτόν. Επομένως, όλες οι αντίστοιχες ομάδες(δηλαδή τουλάχιστον οι μισές από τις floor(n/5)ομάδες) συνεισφέρουν τουλάχιστον 3 στοιχεία που είναι μεγαλύτερα του x , εκτός από τη μία ομάδα που έχει λιγότερα από 5 στοιχεία εάν το n δεν διαιρείται ακριβώς δια του 5, και τη μία ομάδα η οποία περιέχει το ίδιο το x. Αν αφαιρέσουμε αυτές τις δύο ομάδες, έπεται ότι το πλήθος των στοιχείων που είναι μεγαλύτερα του x είναι τουλάχιστον:

3(ceiling(1/2*ceiling(n/5))-2) >= 3n/10-6

Αντίστοιχα, τουλάχιστον 3n/10-6 στοιχεία είναι μικρότερα του x. Επομένως, στην αναδρομική κλήση της στο βήμα 5, η ΕΠΙΛΟΓΗ(ο αλγόριθμος που αναλύουμε) θα δεχθεί ως είσοδο το πολύ 7n/10+6 στοιχεία στην χειρότερη περίπτωση.

Μετά από αυτά,μπορούμε πλέον να καταστρώσουμε μία αναδρομική σχέση για τον χρόνο εκτέλεσης χειρότερης περίπτωσης T(n) του αλγορίθμου ΕΠΙΛΟΓΗ. Τα βήματα 1,2 ακι 4 απαιτο΄ν χρόνο O(n).(Το βήμα 4 συνίσταται σε O(n) κλήσεις της ενθετκής ταξινόμησης(Insertion sort) σε σύνολα μεγέθους O(1).)Το βήμα 3 απαιτεί χρόνο T(ceiling(n/5)), και το βήμα 5 απαιτεί χρόνο T(7n/10+6) το πολύ, αν υποθέσουμε ότι η συνάρτηση T είναι μονότονα αύξουσα. Δεχόμαστε, αν και εκ πρώτης όψεως ο λόγος για αυτήν την παραδοχή δεν είναι εμφανής οτι για οποιαδήποτε είσοδο με λιγότερα από 140 στοιχεία απαιτείται χρόνος O(1)(η προέλευση της μαγικής σταθεράς 140 θα αποσαφηνιστεί παρακάτω).Με την παραδοχή αυτή, παίρνουμε την ακόλουθη αναδρομική σχέση

T (n) <= O(1) if n < 140 ;

T (n) <= T(ceiling(n/5))+ T(7n/10+6) + O(n) if n >= 140

Θα δείξουμε ότι ο χρόνος εκτέλεσης είναι γραμμικός με τη μέθοδο της αντικαταστασης. Συγκεκριμένα, θα δείξουμε ότι T(n) <= cn για κάποια επαρκώς μεγάλη σταθερά c για όλα τα n>0. Αρχικά υποθέτουμε οτι T(n) <= cn για κάποια επαρκώς μεγάλη σταθερά c για όλα τα n<140(η παραδοχή αυτή ισχύει εφόσον το c είναι αρκετά μεγάλο).Επιλέγουμε επίσης μία σταθερά α τέτοια ώστε η συνάρτηση την οποία αντιπροσωπεύει ο όρος O(n) στην παραπάνω σχέση(ο οποίος περιγράφει τη μη αναδρομική συνιστώσα του χρόνου εκτέλεσης του αλγορίθμου) να είναι φραγμένη εκ των άνω από την ποσότητα αn για όλα τα n>0. Αν ανατικαταστήσουμε αυτήν την επαγωγική υπόθεση στο δεξιό μέλος της αναδρομικής σχέσης, έχουμε:

T(n) <= c*ceiling(n/5) + c(7n/10+6) + αn

<= cn/5+c+7cn/10+6c+αn

= 9cn/10+7c+αn

=cn + (-cn/10+7c+αn).

Η ποσότητα αυτή είναι μικρότερη ή ίση του cn εάν

-cn/10+7c+αn <=0 (1.1)

Η ανισότητα (1.1) είναι ισοδύναμη με την ανισότητα c>= 10α(n/(n-70)) όταν n>70.Δεδομένου ότι υποθέτουμε πως n>=140,έχουμε ότι n/(n-70)<=2,και επομένως αν επιλέξουμε c>=20α η ανισότητα (1.1) ικανοποιείται.(ΣΗΜΕΙΩΣΗ:Δεν υπάρχει τίποτα ιδιαίτερο όσον αφορά τη σταθερά 140,θα μπορούσαμε να την αντικαταστήσουμε με οποιονδήποτε ακέραιο μεγαλύτερο του 70 και να επιλέξουμε το c ανάλογα.ΕΠΟΜΕΝΩΣ ο αλγόριθμος αυτός έχει γραμμικό χρόνο εκτέλεσης χειρότερης περίπτωσης.

Πηγές:

wikipedia: https://en.wikipedia.org/wiki/Median_of_medians

stackoverflow: https://stackoverflow.com/questions/17582726/median-of-medians , https://stackoverflow.com/questions/60606451/median-of-medians-algorithm-which-element-to-select-as-median-for-each-group

Introduction to Algorithms: https://edutechlearners.com/download/Introduction_to_algorithms-3rd%20Edition.pdf

BRILLIANT: https://brilliant.org/wiki/median-finding-algorithm/

(1)

Partition helper functions [edit]

See also: Dutch national flag problem

There is a subroutine called `partition` that can, in linear time, group a list (ranging from indices `left` to `right`) into three parts, those less than a certain element, those equal to it, and those greater than the element (a three-way partition). The grouping into three parts ensures that the median-of-medians maintains linear execution time in a case of many or all coincident elements. Here is pseudocode that performs a partition about the element `list[pivotIndex]`:

```
function partition(list, left, right, pivotIndex, n)
    pivotValue := list[pivotIndex]
    swap list[pivotIndex] and list[right]  // Move pivot to end
    storeIndex := left
    // Move all elements smaller than the pivot to the left of the pivot
    for i from left to right − 1 do
        if list[i] < pivotValue then
            swap list[storeIndex] and list[i]
            increment storeIndex
    // Move all elements equal to the pivot right after
    // the smaller elements
    storeIndexEq = storeIndex
    for i from storeIndex to right − 1 do
        if list[i] = pivotValue then
            swap list[storeIndexEq] and list[i]
            increment storeIndexEq
    swap list[right] and list[storeIndexEq]  // Move pivot to its final place
    // Return location of pivot considering the desired location n
    if n < storeIndex then
        return storeIndex  // n is in the group of smaller elements
    if n ≤ storeIndexEq then
        return n  // n is in the group equal to pivot
    return storeIndexEq // n is in the group of larger elements
```

The `partition5` subroutine selects the median of a group of at most five elements; an easy way to implement this is insertion sort, as shown below.[1] It can also be implemented as a decision tree.

```
function partition5( list, left, right)
    i := left + 1
    while i ≤ right
        j := i
        while j > left and list[j−1] > list[j] do
            swap list[j−1] and list[j]
            j := j − 1
        i :=  i + 1

    return floor((left + right) / 2)
```