# Τυχαιοποιημένοι Αλγόριθμοι

*(CLR, κεφάλαιο 8.3 και 10)*

Στην ενότητα αυτή θα μελετηθούν τα εξής θέματα:

*Ο τυχαιοποιμένος αλγόριθμος QuickSort*

*Αλγόριθμοι Επιλογής*

- *Τυχαιποιημένος Αλγόριθμος*
- *Ο αλγόριθμος των Blum, Floyd, Pratt, Rivest, Tarjan*

*Ο αλγόριθμος του Freivalds*

*Πρωτόκολα Μηδενικής Γνώσης*

# Τυχαιοποιημένοι Αλγόριθμοι

Ένας θησαυρός βρίσκεται κρυμμένος σε κάποιο νησί σε σημείο που αποκαλύπτεται σε ένα χάρτη τον οποίο δεν έχετε ακόμα πλήρως κατανοήσει. Μέχρι στιγμής όμως έχετε καταλάβει ότι ο θησαυρός βρίσκεται σε ένα από δύο πιθανά σημεία τα οποία απέχουν 5 μέρες το ένα από το άλλο και από το σημείο όπου βρίσκεστε. Αν δουλέψετε 4 ακόμα μέρες πάνω στο χάρτη, τότε θα μπορούσατε να αποφασίσετε σε ποιο από τα δύο σημεία βρίσκεται ο θησαυρός. Όμως γνωρίζετε ότι υπάρχει κάποιος δράκος ο οποίος κάθε βράδυ μεταφέρει ποσότητα x του θησαυρού στη σπηλιά του. Ένας νάνος σας προσφέρει να σας αποκαλύψει που βρίσκεται ο θησαυρός με αντάλλαγμα την ποσότητα που θα κουβαλούσε ο δράκος σε 3 νύχτες.

Σας συμφέρει να δεχτείτε την προσφορά του;

# Τυχαιοποιημένοι Αλγόριθμοι

- Τυχαιοποιημένες αποφάσεις
  - μερικές φορές είναι αποδοτικότερες κατά μέσο όρο από τον ακριβή υπολογισμό της σωστής απόφασης
  - διαφορετικές εκτελέσεις του ίδιου αλγορίθμου έχουν διαφορετική συμπεριφορά
  - μπορεί να δώσουν λανθασμένα αποτελέσματα (με κάποια μικρή πιθανότητα)

- Υπάρχουν τρεις κατηγορίες πιθανοτικών/τυχαιοποιημένων αλγόριθμων.
  1. Οι *τυχαιοποιημένοι αριθμητικοί αλγόριθμοι* οι οποίοι επιστρέφουν απαντήσεις της μορφής: το ζητούμενο βρίσκεται στο διάστημα [α..β] με πιθανότητα γ%.
  2. Οι *αλγόριθμοι Monte Carlo* οι οποίοι δίνουν σωστά αποτελέσματα με μεγάλη πιθανότητα (≤1)
  3. Οι *αλγόριθμοι Las Vegas* οι οποίοι όταν τερματίζουν δίνουν πάντα τη σωστή απάντηση και τερματίζουν με μεγάλη πιθανότητα (≤1).

# Quicksort

- Πρακτικά, ο πιό γρήγορος αλγόριθμος ταξινόμησης.

- Στη χείριστη περίπτωση ο αλγόριθμος quicksort είναι $O(n^2)$. Στη μέση περίπτωση ο αλγόριθμος είναι $O(n \log n)$.

- Περιγραφή Διαδικασίας Quicksort(A[p..r])
  - Αν $r \leq p$ δεν κάνουμε τίποτα, διαφορετικά
  - Αναδρομικά:

    Χωρίζουμε τον πίνακα σε δύο μέρη A[p..q] και A[q+1,r], όπου όλα τα στοιχεία του A[p..q] είναι μικρότερα από τα στοιχεία του A[q+1..r].

    Καλούμε αναδρομικά τον αλγόριθμο στα A[p..q] και A[q+1,r].

# Ο αλγόριθμοςQuicksort

```
Quicksort(A, p, r){
   if (p<r)
        q = Partition (A, p, r);
        Quicksort(A, p, q);
        QuickSort(A, q+1, r);
}
```

- όπου η διαδικασία **`Partition (A,p,r)`** αναδιοργανώνει τον πίνακα A[p…r] έτσι ώστε A[p..q] να περιέχει στοιχεία ≤ A[p], A[q..r] να περιέχει στοιχεία ≥A[p], και επιστρέφει την τίμη q.

# Η διαδικασία Partition

- Βασική Ιδέα της Partition

  Επαναλαμβάνουμε τα εξής μέχρις ότου τα i και j να διασταυρωθούν.

  1. προχώρα το j προς τα αριστερά όσο τα στοιχεία που βρίσκεις είναι μεγαλύτερα του A[p],
  2. προχώρα το i προς τα δεξιά όσο τα στοιχεία που βρίσκεις είναι μικρότερα του A[p],
  3. αντάλλαξε τα στοιχεία που δείχνονται από τα i και j.
  4. Όταν τα i και j διασταυρωθούν επέστρεψε την τιμή του j.

# Η διαδικασία Partition

```
Partition(A, p, r){
   x = A[p];
   i = p;
   j = r;
   while TRUE {
      while (j≥p and A[j] > x) j--;
      while (i≤r and A[i] < x) i++;
      if (i<j)
          swap (A, i, j); i++; j--;
      else
          return j;
}
```

# Ανάλυση του Χρόνου Εκτέλεσης

- Έστω T(n) ο χρόνος εκτέλεσης του Quicksort με δεδομένο εισόδου μεγέθους n.
- Για τη βάση της αναδρομής έχουμε:

  $T(0) = T(1) = c$
- Για $n \geq 1$

  $T(n) = T(i) + T(n-i) + cn$

  όπου i είναι το μέγεθος είναι το μέγεθος του αριστερού κομματιού μετά από το Partition.
- *Χείριστη περίπτωση*: i = 0 ή n-1. Τότε

  $T(n) = T(n-1) + cn$ και $T(n) \in O(n^2)$.
- *Βέλτιστη περίπτωση*: i = n/2

  $T(n) = 2\,T(n/2) + cn$ και $T(n) \in O(n \log n)$.
- *Μέση περίπτωση*: Αν υποθέσουμε πως όλες οι περιπτώσεις για το δεδομένο εισόδου είναι εξίσου πιθανές, έχουμε επίσης, $T(n) \in O(n \log n)$.

# Τυχαιοποιημένος Αλγόριθμος Quicksort

- Κατά την ανάλυση μέσης περίπτωσης του Quicksort *υποθέσαμε ότι όλες οι διατάξεις του δεδομένου εισόδου είναι εξίσου πιθανές*. Με αυτή την υπόθεση ο χρόνος εκτέλεσης μέσης περίπτωσης είναι O(n lg n).
- Θα δούμε πως με τη χρήση *τυχαιοποιημένων αποφάσεων μπορούμε να αγνοήσουμε αυτή την υπόθεση* με αποτέλεσμα αλγόριθμους που *για οποιοδήποτε δεδομένο εισόδου ο μέσος χρόνος εκτέλεσης είναι της τάξης O(n lg n).*
- **Βασική ιδέα: αντί να υποθέτουμε την ύπαρξη κάποιας κατανομής, επιβάλλουμε την ύπαρξη κάποιας κατανομής.**
- Χρήση random-number generators.
- **`Random(a,b)`** επιστρέφει ένα ακέραιο x, a≤x≤b, με κάθε αριθμό να είναι εξίσου πιθανός (ομοιόρφη κατανομή).

  π.χ. Random(0,1) επιτρέφει είτε 0, με πιθανότητα 1/2, είτε 1, με πιθανότητα 1/2.

# Τυχαιοποιημένος Αλγόριθμος Quicksort

- Μια δυνατότητα για τυχαιοποιημένο αλγόριθμο ταξινόμησης είναι πριν να εφαρμόσουμε τον αλγόριθμο Quicksort να αναδιατάξουμε το δεδομένο εισόδου με διαδικασία η οποία επιβάλλει ομοιόμορφη κατανομή πιθανοτήτων για κάθε διάταξη. (Μπορεί να επιτευχθεί σε χρόνο O(n).)

- Δεύτερη δυνατότητα είναι να διαλέξουμε το pivot χρησιμοποιώντας τυχαιοποίηση.

- Παρατήρηση: και στις δύο περιπτώσεις η χείριστη περίπτωση εκτέλεσης οφείλεται στην τυχαιοποιημένη επιλογή και όχι στη μορφή του δεδομένου εισόδου.

# Αλγόριθμος με τυχαιοποιημένη επιλογή pivot

```
Randomized_Partition(A,p,r){
   i=Random(p,r);
   swap(A[r], A[i]);
   return Partition(A,p,r)
}

Randomized_Quicksort(A,p,r){
  if (p<r)
    q=Randomized_Partition(A,p,r);
    Randomized_Quicksort(A,p,q);
    Randomized_Quicksort(A,q+1,r);
}
```

- Για την ανάλυση του χρόνου εκτέλεσης υποθέτουμε ότι όλα τα στοιχεία του δεδομένου εισόδου είναι ξεχωριστά.
- Μας ενδιαφέρει να υπολογίσουμε την πιθανότητα των διαφορετικών διαμερισμών του δεδομένου εισόδου (της τιμής του q).

# Αλγόριθμος με τυχαιοποιημένη επιλογή pivot

- Γνωρίζουμε ότι
  1. η τιμή του i είναι κατανεμημένη ομοιόμορφα στο διάστημα [p,…,r].
  2. η τιμή του q εξαρτάται από το rank(A[i]) = αριθμός των στοιχείων στο A[p,…,r] τα οποία είναι ≤A[i])

- Έστω ότι n είναι ο αριθμός των στοιχείων του A[p,…,r]. Έχουμε ότι
  1. αν rank(A[i])=1, τότε ο A[p,q] θα περιέχει ένα στοιχείο.
  2. αν rank(A[i])≥2, τότε A[p,q] θα περιέχει rank(A[i] – 1 στοιχεία.

- Άρα το δεδομένο εισόδου μοιράζεται
  - με πιθανότητα 2/n με 1 στοιχείο στον αριστερό υποπίνακα και n – 1 στοιχεία στο δεξιό υποπίνακα, και
  - με πιθανότητα 1/n με i στοιχεία στον αριστερό υποπίνακα και n – i στοιχεία στο δεξιό υποπίνακα, όπου 2≤i≤n-1.

# Χρόνος Εκτέλεσης

- Ο μέσος χρόνος εκτέλεσης του αλγορίθμου δίνεται από την εξίσωση

$$T_e(n) = \frac{1}{n}\left( T_e(1) + T_e(n-1) + \sum_{q=1}^{n-1} (T_e(q) + T_e(n-q)) + \Theta(n) \right)$$

- Λύνοντας την εξίσωση (CRL, pages 165-166) βρίσκουμε ότι $T_e(n) \in O(n \lg n)$.

# Αλγόριθμοι Επιλογής

- **<u>Το πρόβλημα</u>**

  Δεδομένο εισόδου: σύνολο A, $|A|=n$ και i, $1 \le i \le n$

  Δεδομένο εξόδου: το i-οστό στοιχείο του A, δηλαδή $x \in A$ το οποίο είναι μεγαλύτερο από ακριβώς i-1 στοιχεία.

- Λύση 1: Ταξινόμησε το σύνολο και επέστρεψε το i-οστό στοιχείο.

  Χρονική πολυπλοκότητα: $O(n \lg n)$

- Γνωρίζουμε ότι για τις περιπτώσεις i=1 και i=n (εύρεση ελάχιστου και μέγιστου στοιχείου) το πρόβλημα μπορεί να λυθεί με n-1 συγκρίσεις, δηλαδή, σε χρόνο $\Theta(n)$.

- Μπορούμε να λύσουμε το πρόβλημα επιλογής σε χρόνο $\Theta(n)$;

# Λύση 2: Αλγόριθμος διαίρει και βασίλευε

```
Randomized_Select(A,p,r,i){
   if (p=r)
        return A[p];
   else
        q=Randomized_Partition(A,p,r);
        k=q-p+1;
        if (i≤k)
           Randomized_Select(A,p,q,i);
        else
           Randomized_Select(A,q+1,r,i-k);
}
```

- Όπως και στην περίπτωση του τυχαιοποιημένου αλγορίθμου Quicksort, ο αλγόριθμος μοιράζει τα δύο μέρη ως προς κάποιο στοιχείο x σε αριστερό μέρος που περιέχει όλα τα στοιχεία που είναι ≤ του x και στο δεξιό μέρος που περιέχει όλα τα στοιχεία ≥ του x.

# Λύση 2: Αλγόριθμος διαίρει και βασίλευε

- Σε αντίθεση με τον αλγόριθμο Quicksort ο πιο πάνω αλγόριθμος κάνει μια μόνο αναδρομική κλήση του εαυτού του: Αν το αριστερό μέρος περιέχει λιγότερα από i στοιχεία συνεχίζει την αναζήτηση του i-στού στοιχείου στο αριστερό μέρος, διαφορετικά αναζητεί το "κατάλληλο" στοιχείο στο δεξιό μέρος.

- Αφού η επιλογή του pivot είναι τυχαιοποιημένη η χείριστη περίπτωση εκτέλεσης του αλγορίθμου δεν εξαρτάται από το δεδομένο εισόδου, αλλά από την τυχαιοποιημένη επιλογή.

- Πότε παρουσιάζεται η χείριστη περίπτωση;

- Χρόνος εκτέλεσης χειρίστης περίπτωσης:

# Μέσος όρος χρόνου εκτέλεσης

- Γνωρίζουμε ότι η διαδικασία Randomized_Partition μοιράζει το δεδομένο εισόδου
  - με πιθανότητα 2/n με 1 στοιχείο στον αριστερό υποπίνακα και n-1 στο δεξιό υποπίνακα, και
  - με πιθανότητα 1/n με i στοιχεία στον αριστερό υποπίνακα και n-i στο δεξιό υποπίνακα, όπου $2 \le i \le n-1$.

- ΄Αρα ο μέσος χρόνος εκτέλεσης δίνεται από την αναδρομική ανίσωση

$$T_e(n) \le \frac{1}{n}\left( T_e(\max(1, n-1)) + \sum_{q=1}^{n-1} T_e(\max(q, n-q)) + \Theta(n) \right)$$

- Μπορούμε να αποδείξουμε με τη μέθοδο της αντικατάστασης ότι $T_e(n) \in O(n)$.

# Αλγόριθμος επιλογής Blum, Floyd, Pratt, Rivest, Tarjan

- Ο πιο κάτω αλγόριθμος έχει χρονική πολυπλοκότητα χείριστης περίπτωσης της τάξης O(n).

- Το άνω φράγμα O(n) επιτυγχάνεται με κάποια προεργασία του δεδομένου εισόδου η οποία εγγυείται ότι εφαρμογή της διαδικασία Partition θα οδηγήσει σε ισοζυγισμένο μοίρασμα του αρχικού πίνακα.

- Ο αλγόριθμος *δεν* είναι τυχαιοποιημένος.

- Μεσαίο στοιχείο ενός πίνακα με n στοιχεία ονομάζουμε το (k+1)-οστό μεγαλύτερο στοιχείο του πίνακα όπου $k=\lfloor n/2 \rfloor$.

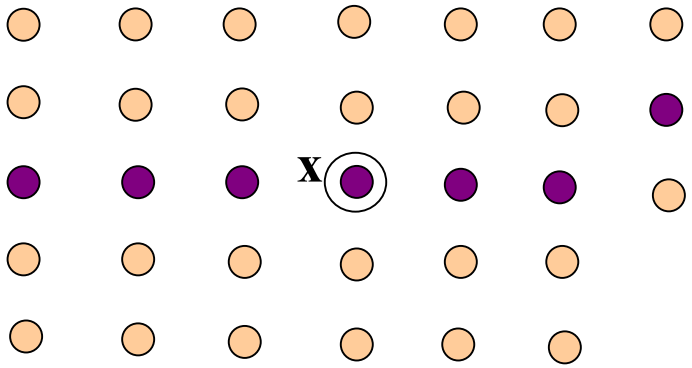
# Ο Αλγόριθμος

**`Select(A,p,q,i)`**

1. μοίρασε το δεδομένο εισόδου σε $\lfloor n/5 \rfloor$ υποπίνακες με 5 στοιχεία ο καθένας, εκτός από τον τελευταίο ο οποίος πιθανόν να έχει από 1 μέχρι 5 στοιχεία.
2. ταξινόμησε τους υποπίνακες και επέτρεψε το μεσαίο στοιχείο κάθε υποπίνακα
3. Κάλεσε αναδρομικά τη διαδικασία Select για εύρεση του μεσαίου στοιχείου x των μεσαίων στοιχείων που επιστράφηκαν στο βήμα 2.
4. Αντάλλαξε το στοιχείο x με το A[p] και εφάρμοσε στον αρχικό πίνακα τη διαδικασία Partition(A). Έστω r η θέση του τελευταίου στοιχείου στον αριστερό υποπίνακα και k ο αριθμός των στοιχείων του αριστερού υποπίνακα.
5. Αν $i \leq k$ κάλεσε αναδρομικά τη διαδικασία Select(A,p, r, i) διαφορετικά, αν $i>k$, κάλεσε αναδρομικά τη διαδικασία Select(A,r+1,q,i-k)

# Ανάλυση Χρόνου Εκτέλεσης

- Παρατηρούμε ότι ο αριθμός των στοιχείων του Α που είναι μικρότερα από το στοιχείο x είναι τουλάχιστον 3n/10 -6:
- Τουλάχιστον τα μισά από τα n/5 μεσαία στοιχεία που επιστρέφονται από το βήμα 2 είναι μικρότερα του x. Από αυτά τουλάχιστον όλοι εκτός από δύο από τους αντίστοιχους υποπίνακες (δηλαδή εκτός από τον τελευταίο υποπίνακα και τον πίνακα στον οποίο ανήκει το x) προσφέρουν τουλάχιστον 3 στοιχεία που είναι μικρότερα από το x.



# Ανάλυση Χρόνου Εκτέλεσης

- Άρα συνολικά τουλάχιστον

  $3(\lceil 1/2 \lceil n/5 \rceil \rceil -2) \geq 3n/10 -6$

  στοιχεία είναι μικρότερα του x.

- Επομένως, στη χείριστη περίπτωση, η κλήση της διαδικασίας Select στο βήμα 5 του αλγορίθμου καλείται σε πίνακα με το πολύ $7n/10 + 6$ στοιχεία.

- Έστω $T(n)$ ο χρόνος εκτέλεσης της διαδικασίας σε δεδομένο εισόδου μεγέθους n. Έχουμε ότι

  $$T(n) = T(\lceil n/5 \rceil) + T(7n/10) + O(n)$$

- Με τη μέθοδο της αντικατάστασης παίρνουμε ότι $T(n) \in O(n)$.

# Ο αλγόριθμος του Freivalds

- **Έστω n×n πίνακες A, B και C.** **Ερώτημα: AB=C;**

**Προσπάθεια 1:**

- Πολλαπλασίασε τους πίνακες A και B και σύγκρινε το γινόμενό τους με το C.
- Χρονική Πολυπλοκότητα:
- Το πρόβλημα μπορεί να λυθεί με ένα Monte Carlo αλγόριθμο σε χρόνο $O(n^2)$.

**Ο αλγόριθμος**

```
Freivalds(A,B,C){
  Διάλεξε  x ∈ {0,1}^n τυχαία και ομοιόμορφα
  if A(Bx) ≠ Cx
    return (AB ≠ C)
  else
    return (AB = C, probably)
}
```

# Ο αλγόριθμος του Freivalds

- Χρονική Πολυπλοκότητα:

  $O(n^2)$ για τους υπολογισμούς Bx, A(Bx) και Cx

  $O(n)$ για τον έλεγχο ABx = Cx

  Άρα συνολικά $O(n^2)$

- Ορθότητα

  Υπάρχουν δύο περιπτώσεις:

  1. Αν AB = C, τότε ABx = Cx για κάθε x, επομένως ο αλγόριθμος επιστρέφει τη σωστή απάντηση με πιθανότητα 1.
  2. Αν AB ≠ C υπάρχει κάποια πιθανότητα να ισχύει ABx = Cx. Δηλαδή υπάρχει κάποια πιθανότητα ο αλγόριθμος να επιστρέψει λανθασμένη απάντηση. Η πιθανότητα αυτή όμως δεν είναι πολύ μεγάλη:

# Ο αλγόριθμος του Freivalds

**ΛΗΜΜΑ:** Έστω ότι $AB \neq C$ και το διάνυσμα x επιλέγεται τυχαία και ομοιόμορφα από το σύνολο $\{0,1\}^n$. Τότε $Pr(ABx = Cx) \leq ½$

Απόδειξη

- Αφού $AB \neq C$, $(AB-C) \neq 0$, άρα υπάρχουν i και j για τα οποία $(AB-C)[i,j] \neq 0$.
- Έστω $(a_1, a_2, \ldots, a_n)$ η i-οστή σειρά του πίνακα (AB-C), και x, το τυχαία και ομοιόμορφα επιλεγμένο διάνυσμα, $x= (x_1, x_2, \ldots, x_n)^T$.
- Θεωρήστε το άθροισμα

$$\sum_{k=1}^{n} a_k x_k$$

# Ο αλγόριθμος του Freivalds

- Έχουμε ότι $\Pr((AB-C)x=0) \le \Pr(\sum_{k=1}^{n} a_k x_k = 0)$

$$= \Pr(x_j = \frac{-1}{a_j} \cdot \sum_{k \ne j} a_k x_k)$$

$$\le \frac{1}{2}$$

εφόσον το $a_j \neq 0$ και το $x_j \in \{0,1\}$ με ίσες πιθανότητες για κάθε περίπτωση.

- Συνεπώς για κάποια δεδομένα εισόδου, ο αλγόριθμος δίνει λανθασμένη απάντηση τις μισές φορές.
- Τρέχοντας όμως τον αλγόριθμο μέχρι k φορές, στην περίπτωση που μας δίνει τη θετική απάντηση, μειώνουμε την πιθανότητα λανθασμένης απάντησης στο $2^{-k}$ , χωρίς να μεταβληθεί η ασυμπτωτική πολυπλοκότητα του αλγορίθμου.

# Πρωτόκολα Μηδενικής Γνώσης

- Έστω ότι η Alice γνωρίζει κάποιο μυστικό το οποίο δεν γνωρίζει ο Bob. Είναι δυνατό η Alice να πείσει τον Bob για αυτή τη γνώση της χωρίς όμως να του αποκαλύψει το μυστικό;
- Δεδομένα: κάποιο αλγοριθμικό πρόβλημα P και στιγμιότυπο του προβλήματος, X.
- Ζητούμενο: αλγόριθμος ο οποίος πείθει τον Bob ότι P(X) = True χωρίς να αποκαλύπτει οτιδήποτε για την απόδειξη P(X)= True.
- Θα μελετήσουμε μια αντιπροσωπευτική μορφή του προβλήματος.
- Έστω P το πρόβλημα: οι κορυφές ενός γράφου μπορούν να χρωματιστούν με τρία χρώματα έτσι ώστε κανένα ζεύγος γειτονικών κορυφών να μην έχουν το ίδιο χρώμα. (Το πρόβλημα αυτό δεν μπορεί να λυθεί σε πολυωνυμικό χρόνο.) Έστω X κάποιος γράφος.
- Η Alice θέλει να πείσει τον Bob ότι P(X)=True, δηλαδή ότι ο γράφος X μπορεί να χρωματιστεί σύμφωνα με τις προδιαγραφές, χωρίς όμως να αποκαλύψει στον Bob κάποιο συγκεκριμένο χρωματισμό.

# Ο Αλγόριθμος

**Μέχρις ώτου πεισθεί ο Bob κάνε**

1. **Η Alice επιλέγει τρία χρώματα, τα αποκαλύπτει στον Bob και δημιουργεί κάποιο χρωματισμό του X καλύπτοντας όλες τις κορυφές.**
2. **Ο Bob ζητά να δεί κάποια ακμή e.**
3. **Η Alice αποκαλύπτει τις κορυφές στα άκρα της e=(u, v).**
4. **Αν οι κορυφές u και v έχουν το ίδιο χρώμα τότε ο Bob αποφασίζει ότι η Alice λέει ψέματα, διαφορετικά ο βρόχος επαναλαμβάνεται.**

- Έχουμε ότι
  1. Η Alice μπορεί να είναι ικανοποιημένη για το ότι δεν αποκαλύπτει κάποιο συγκεκριμένο χρωματισμό στον Bob.
  2. Ο Bob μπορεί να είναι σίγουρος ότι μετά από ένα μεγάλο αριθμό επαναλήψεων η πιθανότητα να τον ξεγελάσει η Alice ότι γνωρίζει το μυστικό, τείνει στο 0.
  3. Αν ο γράφος έχει N ακμές, μετά την k-οστή επανάληψη, η πιθανότητα αυτή είναι .....