Εργαστήριο 6: Αναζήτηση, Ανάλυση Πολυπλοκότητας

Στην ενότητα αυτή θα μελετηθούν τα εξής επιμέρους θέματα:

- Αναζήτηση με linearSearch, binarySearch, ternarySearch
- Ανάλυση Πολυπλοκότητας ternarySearch
- Επαλήθευση Πολυπλοκότητας με master theorem

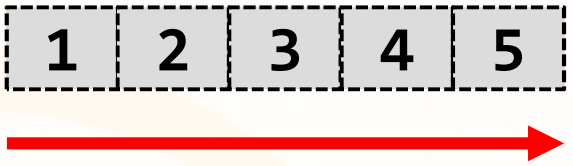
Γραμμική vs. Δυαδική Διερεύνηση

Δεδομένα Εισόδου: Πίνακας X με n στοιχεία, ταξινομημένος από το μικρότερο στο μεγαλύτερο, και ακέραιος k.

Στόχος: Να εξακριβώσουμε αν το k είναι στοιχείο του X.

Γραμμική Διερεύνηση: εξερευνούμε τον πίνακα από τα αριστερά στα δεξιά.

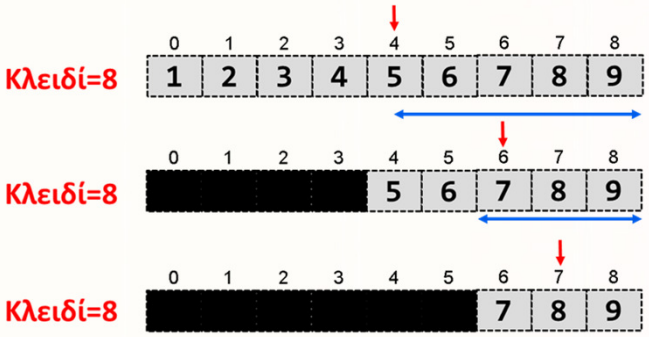
```
int linear( int X[], int n, int k){
  int i=0;
  while ( i < n ) {
    if (X[i] == k) return i;
    if (X[i] > k)  return -1;
    i++;
  }
  return -1;
}
```



Χείριστη περίπτωση: **O(n)** (ο βρόχος εκτελείται n φορές)

Δυαδική Διερεύνηση

Δυαδική Διερεύνηση: βρίσκουμε το μέσο του πίνακα και αποφασίζουμε αν το k ανήκει στο δεξιό ή αριστερό μισό. Επαναλαμβάνουμε την ίδια διαδικασία στο μισό που μας ενδιαφέρει:




```
int binary (int X[], int n, int key) {
  int low=0;
  int high = n - 1;
  int mid;
  while( low >= high) {
    mid = (low + high) / 2;
    if( key < X[mid])
      high = mid-1;
    else if (key > X[mid])
      low = mid+1;
    else {
      return mid; break;
    }
  }
  return -1;
}
```

Υλοποίηση

Υλοποιήστε τις ακόλουθες συναρτήσεις:

- `int linearSearch(int x[], int length, int key)`
  Εκτελεί γραμμικήδυαδική αναζήτηση για εύρεση του κλειδιού **key** σε ένα πίνακα **x** με μέγεθος **length** **(αντιγραφή από διάλεξη 16)**

- `int binarySearch(int x[], int length, int key)`
  Εκτελεί δυαδική αναζήτηση για εύρεση του κλειδιού **key** σε ένα πίνακα **x** με μέγεθος **length** **(αντιγραφή από διάλεξη 16)**

- `int ternarySearch(int x[], int length, int key)`
  Εκτελεί τριαδική αναζήτηση για εύρεση του κλειδιού **key** σε ένα πίνακα **x** με μέγεθος **length**

- Προσθέστε μία παράμετρο (int steps) για καταμέτρηση των βημάτων στην κάθε μία συνάρτηση.
- Επίσης καταμετρήστε τον χρόνο με την βιβλιοθήκη time.h

Έλεγχος

Για έλεγχο του προγράμματός σας υλοποιήστε την μέθοδο main ακριβώς όπως φαίνεται πιο κάτω:

```
//Πειραματιστείτε με διάφορα SIZE
#define SIZE 64000
int main(int argc, char* argv[]) {
   int input[SIZE];
   for(int i=0; i<SIZE; i++){
   input[i]=i;
   }
   srand(time(NULL));
   int random=rand();
   linearSearch(input, SIZE, random);
   binarySearch(input, SIZE, random);
   ternarySearch(input, SIZE, random);
}
```

Λύσεις

```
int linearSearch(int X[], int length, int key){
  for(int steps=0, i=0; i<length; i++,steps++){
    if(X[i]==key) {
      printf("linearSearch steps:%d\n", steps);
      return i;
    }
  }
  printf("linearSearch steps:%d", length+1);
  return -1;
}
```

Λύσεις (συν.)

```
int binarySearch(int X[], int length, int key) {
  int low=0;  int high = length - 1; int mid; int steps=0;
  while( high >= low) {
    mid = low + (high-low) / 2;
    if( key < X[mid])
      high = mid-1;
    else if (key > X[mid])
      low = mid+1;
    else {
      printf("binarySearch steps:%d\n", steps);
      return mid;
    }
    steps++;
  }
  printf("binarySearch steps:%d\n", steps);
  return -1;
}
```

Λύσεις (συν.)

```
int ternarySearch(int X[], int length, int key) {
  int low=0; int high=length-1; int mid_low; int mid_high;
  int steps=0;
  while( high >= low) {
    mid_low = low + (high-low)/3;
    mid_high = low + (high-low)/3 *2;
    if (key == X[mid_low]){
      printf("ternarySearch steps:%d\n", steps);
      return mid_low;
    }
    else if (key == X[mid_high]) {
      printf("ternarySearch steps:%d\n", steps);
      return mid_high;
    }
    else if (key == X[high]) {
      printf("ternarySearch steps:%d\n", steps);
      return high;
    }
    …
```

Λύσεις (συν.)

```
    …
    else if (key == X[low]) {
      printf("ternarySearch steps:%d\n", steps);
      return low;
     }
     else if( key < X[mid_low])
       high = mid_low-1;
     else if( key > X[mid_high])
       low = mid_high+1;
     else {
       low = mid_low   + 1;
       high = mid_high - 1;
     }
     steps++;
   }
   printf("ternarySearch steps:%d\n", steps);
   return -1;
}
```

Ανάλυση Πολυπλοκότητας ternarySearch

Έχουμε την αναδρομική εξίσωση

$T(n) = T(n/3) + 4$, για κάθε $n>1$

$T(1) = 1$

Τότε, αντικαθιστώντας το $T(n/3)$ με την τιμή του παίρνουμε

$T(n) = T(n/3) + 4$ // Εκτέλεση 1

$= (T(n/9) + 4) + 4$ // Εκτέλεση 2

$= T(n/9) + 2x4$ // Πράξεις

$= T(n/27) + 3x4$ // Εκτέλεση 3

$= T(n/3^k) + k.4$ // $k=\log_3 n$ ➔ $3^k=n$

$= T(1) + (\log_3 n).4$ // $n=3^k$, $T(n/3^k)=T(1)=1$

$= 4\log_3 n$ // Πράξεις

$\in \mathbf{\Theta(\log_3 n)}$

Παράδειγμα: Αντικατάσταση

T(n) = T(n/3) + 4

Master Theorem: Έστω ότι η f είναι μία αύξουσα που ικανοποιεί τη λειτουργία της επανάληψης:

$$f(n) = af(n/b) + cn^d$$

όταν το $n = b^k$, όπου k είναι ένας θετικός ακέραιος, $a \geq 1$, b > 1, και c, d είναι πραγματικοί αριθμοί, $c \geq 0, d \geq 0$. Τότε ισχύει:

$$f(n) is \begin{cases} O(n^d) & if\ a < b^d \\ O(n^d log_b(n)) & if\ a = b^d \\ O(n^{log_b(a)}) & if\ a > b^d \end{cases}$$

a=1, b=3, c=4, d=0 $b^d = 3^0 = 1$ → $a = b^d$

$$f(n) is \begin{cases} O(n^d log_b(n)) & if\ a = b^d \end{cases}$$

$T(n) \in O(n^0 log_3(n))$ → $T(n) \in O(log_3(n))$