Day 2

Juniors Summer Camp 2023

Πρόγραμμα

- Πολυπλοκότητες
- STL vector
- Sorting
- Prefix sum
- Problem Solving

Πολυπλοκότητες

Πολυπλοκότητα

- Το πόσο «γρήγορος» είναι ένας αλγόριθμος μετριέται με βάση την πολυπλοκότητα του.
- Ο αλγόριθμος ανάλογα με τα δεδομένα εισόδου μπορεί να διαφέρει στο πόσο γρήγορα θα τρέξει (πόσους υπολογισμούς θα κάνει), για αυτό μετράμε το άνω όριο της πολυπλοκότητας , δηλαδή την χειρότερη πιθανή περίπτωση
- Η πολυπλοκότητα συμβολίζεται με το γράμμα O(…) και μέσα στην παρένθεση γράφουμε την πολυπλοκότητα σε συναρτήσει κάποιων μεταβλητών
- Με ένα «αργό» κώδικα **μπορούμε να πάρουμε κάποιους πόντους** σε ένα πρόβλημα

O(1)

- Πάρα πολύ γρήγορος κώδικας που δεν έχει βρόχους (επαναλήψεις)
- Είναι απλές εντολές, πράξεις

```
int main(){
int a,b;
cin>>a>>b;
cout<<a+b<<endl;
}
```

O(N)

- Γραμμική πολυπλοκότητα
- Η πολυπλοκότητα καθορίζετε από το πόσες επαναλήψεις θα κάνει το for/while

```
int main(){
int n;
cin>>n;
for(int i=1;i<=n;i++){
    cout<<i<<" ";
}
}
```

$O(N^2)$

- Όταν υπάρχει βρόγχος μέσα σε βρόγχο με τις ίδιες επαναλήψεις n τότε η πολυπλοκότητα είναι $O(N^2)$

```
int main(){
int n;
cin>>n;
for(int i=0;i<n;i++){
    for(int j=0;j<n;j++){
        ///code
    }
}
```

O(N*M)

- Όταν υπάρχει βρόγχος μέσα σε βρόγχο με τις διαφορετικές επαναλήψεις n και m τότε η πολυπλοκότητα είναι O(n*m)

```
int main(){
int n,m;
cin>>n>>m;
for(int i=0;i<n;i++){
    for(int j=0;j<m;j++){
        ///code
    }
}
```

Άλλες πολυπλοκότητες

- $O(\sqrt{n})$ (πχ έλεγχος πρώτου αριθμού)
- O(logn) (πχ binary search)
- O(n*logn) (πχ quick sort)
- O(n!)

Και συνδυασμός αυτών πχ O(n*m*log(n^2))

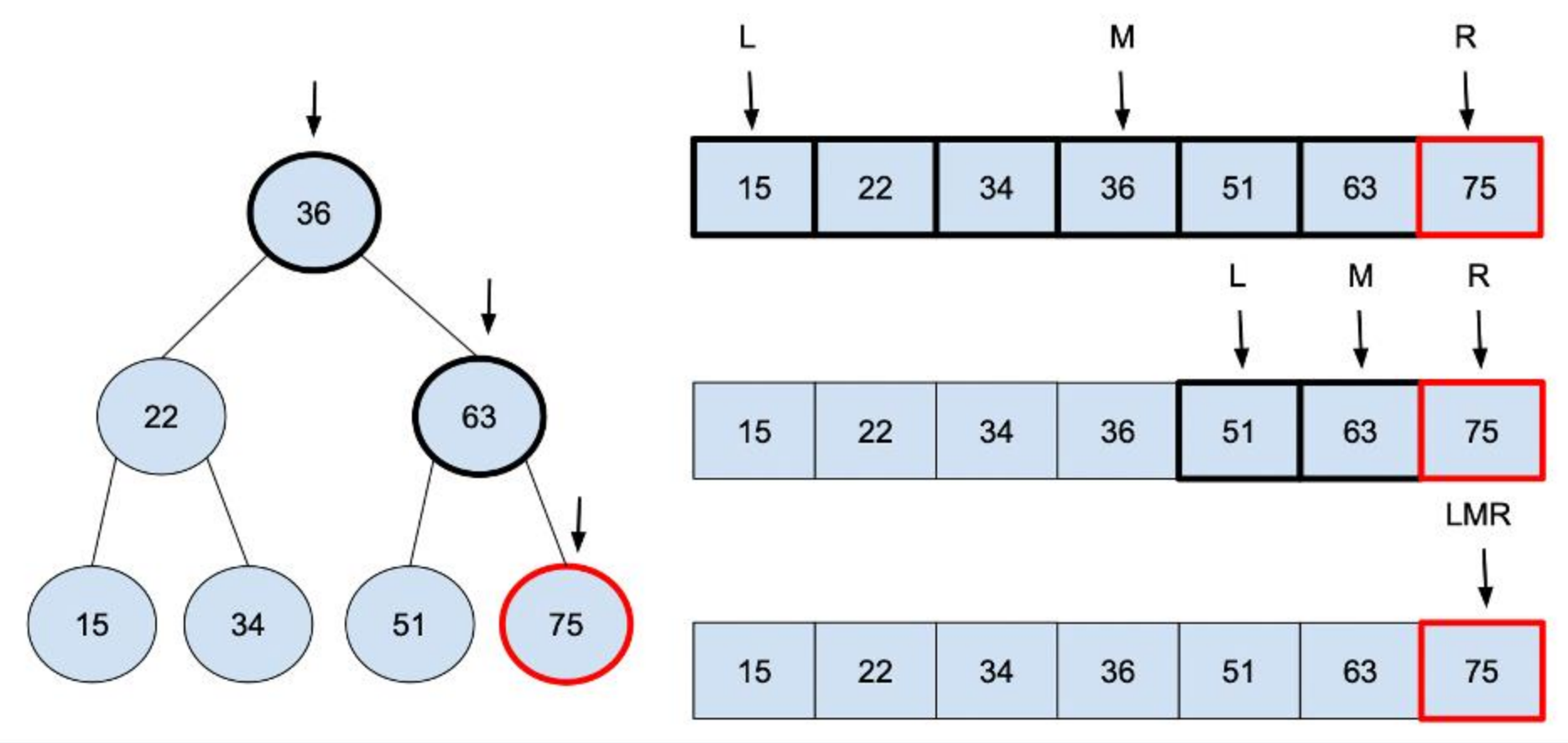
$O(\sqrt{N})$

- Η συνάρτηση ελέγχει εάν ο αριθμός είναι πρώτος με πολυπλοκότητα $O(\sqrt{N})$

```
int protos(int a){
    int ch=1;    ///protos
    for(int i=2;i<=sqrt(a);i++){
        if(a%i==0){
            ch=0;      ///oxi protos
        }
    }
    return ch;
}
```

Λογαριθμική πολυπλοκότητα O(logn)

- Παράδειγμα



Τάξη Μεγέθους

- Όταν μετράμε πολυπλοκότητα δεν λαμβάνουμε υπόψη τις σταθερές αλλά την τάξη μεγέθους

Για παράδειγμα

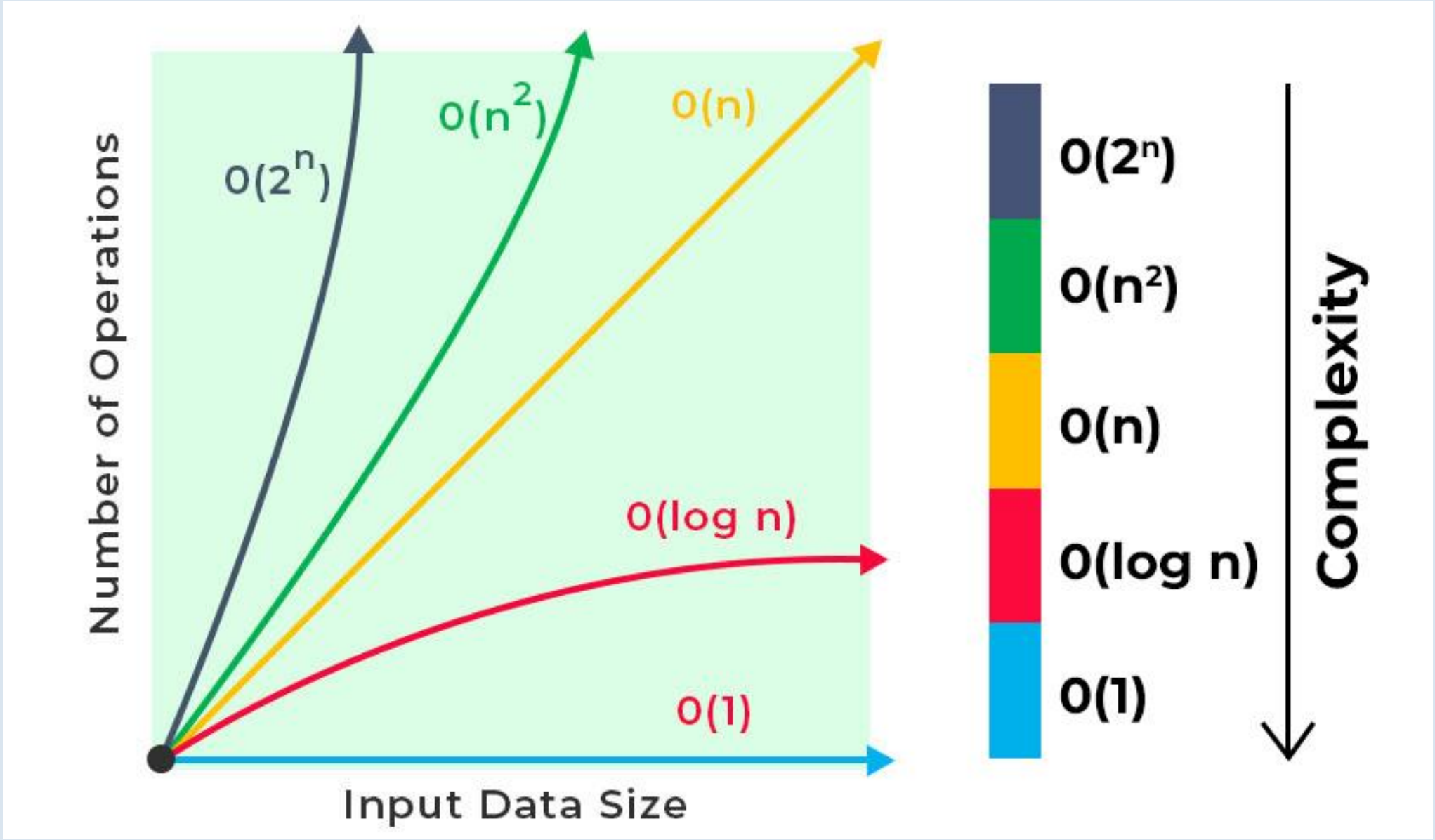
- O(3n) = O(n)
- O(2n + 6) = O(n)
- O(n*(n-1) + 16) = O(n^2)

Φάσεις

- Όταν μετράμε πολυπλοκότητα σε ένα πρόγραμμα που σε κάποια μέρη διαφέρει η πολυπλοκότητα του από κάποια άλλα τότε λαμβάνουμε υπόψη την μεγαλύτερη πολυπλοκότητα από όλες της φάσεις

Για παράδειγμα

- O(3n) , O(3*n^2) , O(n*logn) = O(n^2)
- O(2n + 6) , O(1) , O(n^3 * logn) = O(n^3 * logn)
- O(logn) , O(2*logn) , O(1) = O(logn)


Number of Operations
O(2^n)
O(n^2)
O(n)
O(log n)
O(1)
Input Data Size
O(2^n)
O(n^2)
O(n)
O(log n)
O(1)
Complexity

Ανω όριο πολυπλοκότητας O(f)

N►	10	20	30	50	100
1	1	1	1	1	1
log2	3,322	4,322	4,907	5,644	6,644
n	10	20	30	50	100
n*log2(n)	33,220	86,440	147,210	282,200	664,400
n^2	100	400	900	2500	10000
n^3	1000	8000	27000	125000	1000000
2^n	1024	1048576	1073741824	1,1259E+15	1,2677E+30
n!	3628800	2,4329E+18	2,6525E+32	3,0414E+64	9,333E+157

Πολυπλοκότητα ως βοήθεια

- Όλα πρόβλημα στους διαγωνισμούς αναφέρουν τα όρια των μεταβλητών σε κάθε υποπρόβλημα (subtask)
- Αυτά τα όρια μπορούν να μας βοηθήσουν να βρούμε την λύση του προβλήματος ή πια τεχνική επίλυσης θα χρησιμοποιήσουμε

input size	required time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

Ταξινόμηση

Ταξινόμηση

- Πολλές φορές θα χρειαστεί να ταξινομήσουμε τα δεδομένα μας βάσει ενός κριτηρίου

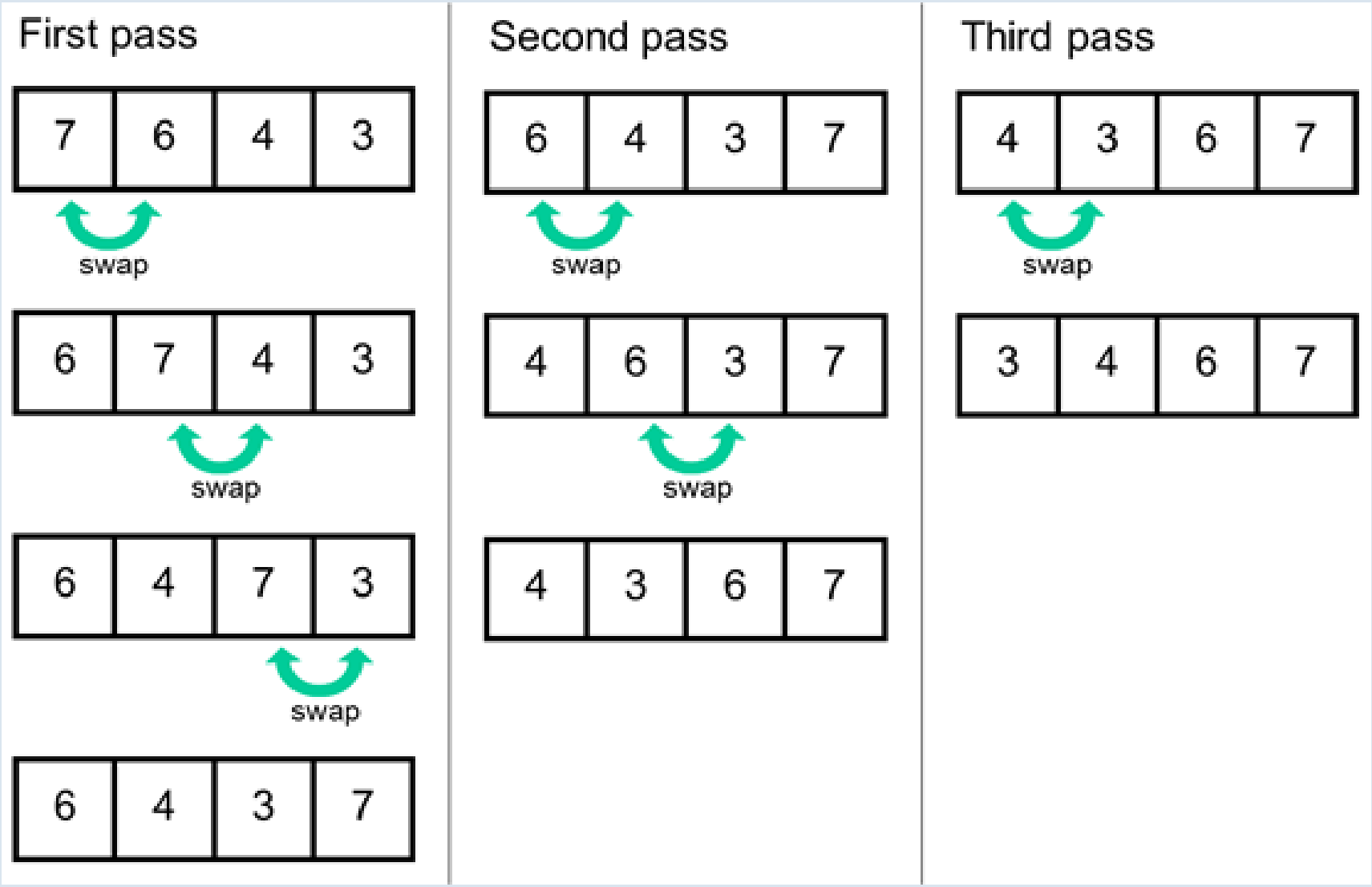
- Υπάρχουν αρκετοί αλγόριθμοι ταξινόμησης όπως bubble sort, merge sort, quick sort, insertion sort …

- Θα δούμε το bubble sort και quick sort

Bubble sort

- Το Bubble sort ή ταξινόμηση φυσαλίδας συγκρίνει δύο δύο τους αριθμούς και εάν δεν είναι στην σωστή σειρά, τότε τους αλλάζει μεταξύ τους μέχρι όλα τα στοιχεία να είναι ταξινομημένα

- Ποια είναι η πολυπλοκότητα;



Πολυπλοκότητα O(n^2)

Υλοποίηση

```
///bubble sort

for(int i=0;i<n;i++){
    for(int j=i;j<n;j++){
        if(a[i]>a[j]){
            swap(a[i],a[j]);
        }
    }
}
```

Quick Sort

- Το quick sort είναι ένας γρηγορότερος αλγόριθμος ταξινόμησης που είναι έτοιμα υλοποιημένος στην c++ στην βιβλιοθήκη algorithm

- Το quick sort ταξινομεί τα δεδομένα σε αύξουσα σειρά από μόνο του, εκτός εάν ορίσουμε εμείς κάτι διαφορετικό

- Quick sort σε πίνακα: sort(a,a+n);
- Βιβλιοθήκη: #include<iostream>
- Πολυπλοκότητα: O(n*logn)

Παράδειγμα

```
1  #include<iostream>
2  #include<algorithm>
3  using namespace std;
4  int main(){
5
6      int a[5];
7      for(int i=0;i<5;i++){
8          cin>>a[i];
9      }
10
11     sort(a,a+5);
12
13     for(int i=0;i<5;i++){
14         cout<<a[i]<<" ";
15     }
16
17 }
```

«Δικό μας» Quick Sort

- Αν θέλω το quick sort να ταξινομεί με δικό μου τρόπο (κριτήριο), τότε μπορώ να υλοποιήσω δική μου συνάρτηση που με βάσει αυτή θα ταξινομήσει τα στοιχεία σε γρήγορο χρόνο

- sort(a,a+n,f) , όπου f δική μου συνάρτηση τύπου bool που περιέχει το κριτήριο ταξινόμησης μου

- Γράψτε ένα πρόγραμμα που ταξινομεί 5 νούμερα σε φθίνουσα σειρά

Ταξινόμηση φθίνουσα

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
bool f(int a,int b){
    return a > b;
}
int main(){

    int a[5];
    for(int i=0;i<5;i++){
        cin>>a[i];
    }

    sort(a,a+5,f);

    for(int i=0;i<5;i++){
        cout<<a[i]<<" ";
    }

}
```

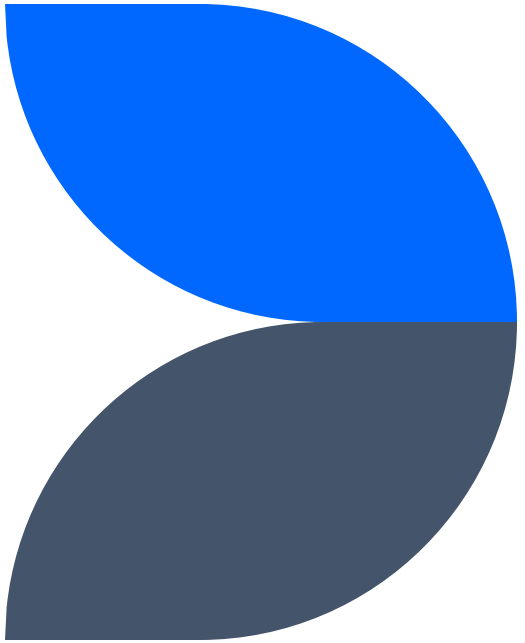
«Δικό μας» Quick Sort με struct

- Για να μπορέσω να χρησιμοποιήσω το quick sort με στοιχεία που είναι δικού μου τύπου (struct), πρέπει να υλοποιήσω συνάρτηση σύγκρισης f που παίρνει παραμέτρους δύο struct και κάνει την σύγκριση όπως θέλω εγώ

- Να γράψετε ένα πρόγραμμα που δέχεται το όνομα, επίθετο και ηλικία 5 ανθρώπων και τους τυπώνει σε αύξουσα σειρά με βάση την ηλικία τους. Σε περίπτωση που δύο άτομα έχουν την ίδιο ηλικία να ταξινομούνται με βάση το επίθετο τους.

```cpp
#include<iostream>
#include<algorithm>
using namespace std;

struct athropos{
string onoma;
string epitheto;
int ilikia;
};

bool f(athropos a,athropos b){
    return a.ilikia < b.ilikia || a.ilikia == b.ilikia && a.epitheto < b.epitheto;
}

int main(){

    athropos a[5];
    for(int i=0;i<5;i++){
        cin>>a[i].onoma>>a[i].epitheto>>a[i].ilikia;
    }

    sort(a,a+5,f);

    for(int i=0;i<5;i++){
        cout<<a[i].onoma<<" "<<a[i].epitheto<<" "<<a[i].ilikia<<endl;
    }

}
```

Vector



Vector

- Τα vectors είναι δομές δεδομένων με τις ιδιότητες ενός δυναμικού πίνακα χωρίς προκαθορισμένο μέγεθος.

- Έχουμε πρόσβαση σε όλα τα στοιχεία ενός vector αλλά έχει και την δυνατότητα να αυξάνει δυναμικά το μέγεθος της δομής κάθε φορά που εισέρχεται ή αφαιρείται κάποιο αντικείμενο

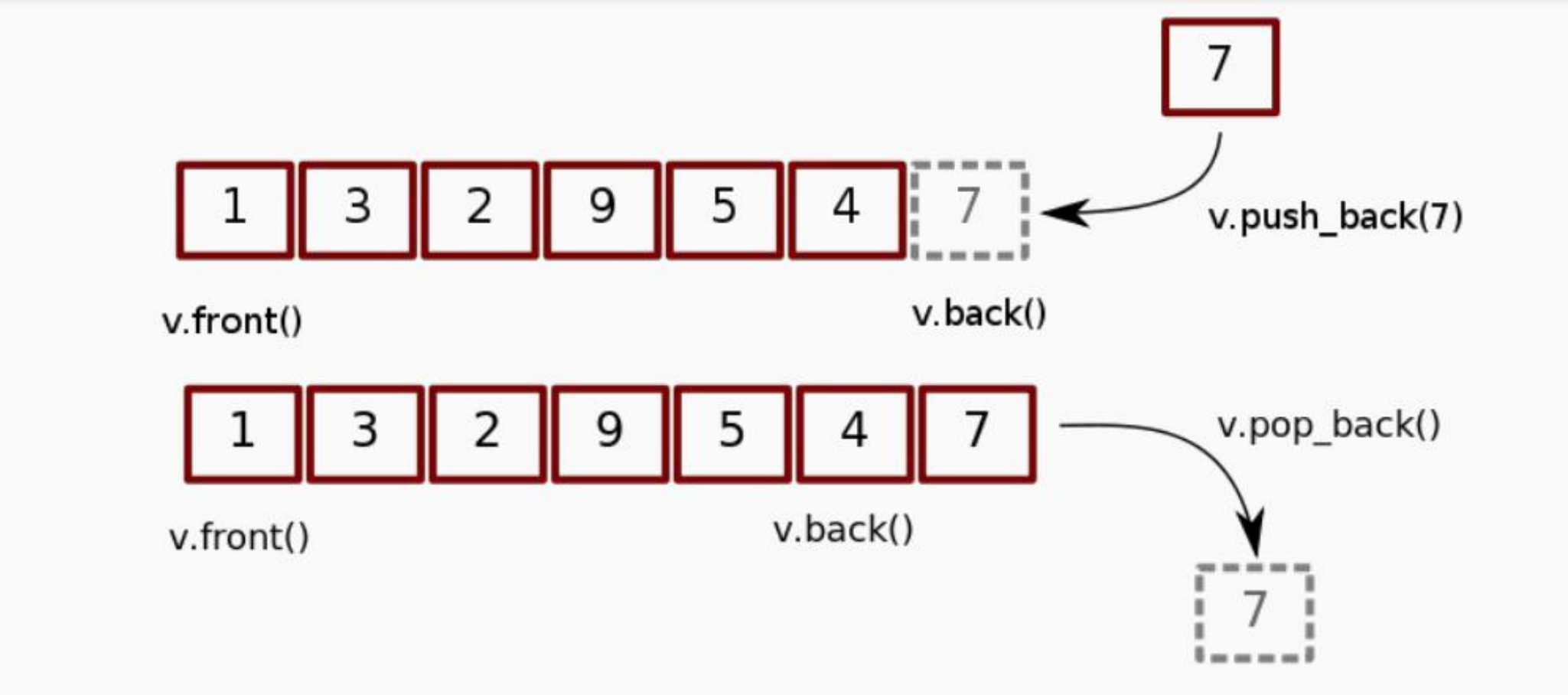
- Βιβλιοθήκη: #include<vector>
- Δήλωση: vector <type> name; (πχ vector < int > a;)

Βασικές συναρτήσεις

insert	Εισάγει στοιχεία μέσα στο vector, όχι απαραίτητα στο τέλος της δομής
push_back	Εισάγει ένα στοιχείο στο τέλος της και εκχωρεί αυτόματα μνήμη
pop_back	Διαγράφει το τελευταίο στοιχείο του vector
erase	Σβήνει ένα ή μια σειρά στοιχείων από το vector
clear	Διαγράφονται όλα τα στοιχεία του vector
front	Παραπέμπει στο πρώτο στοιχείο του vector
back	Παραπέμπει στο τελευταίο στοιχείο του vector
size	Επιστρέφει τον αριθμό των στοιχείων που υπάρχουν στο vector
empty	Επιστρέφει true όταν το vector δεν έχει κανένα στοιχείο

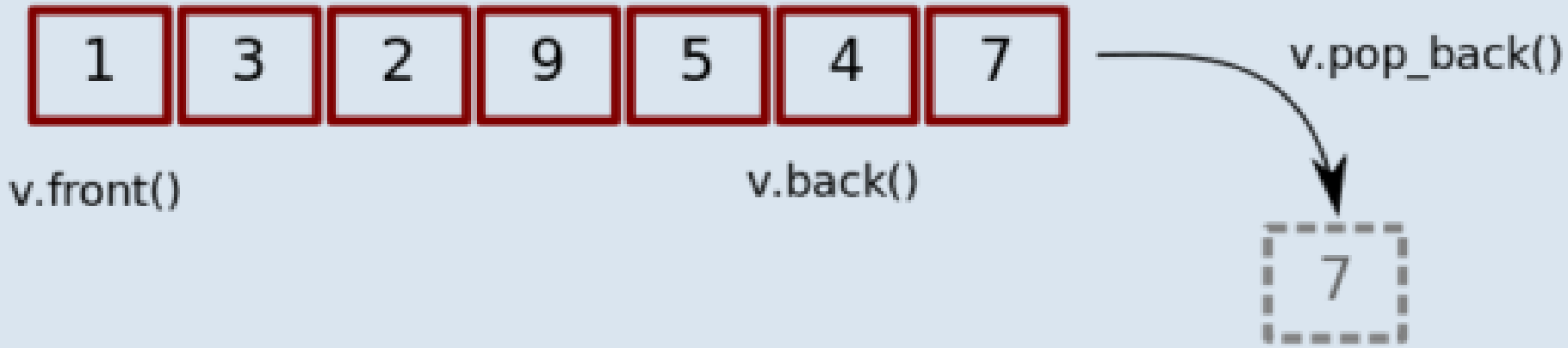
https://cplusplus.com/reference/vector/vector/

push_back / pop_back



Πολυπλοκότητα: O(1)

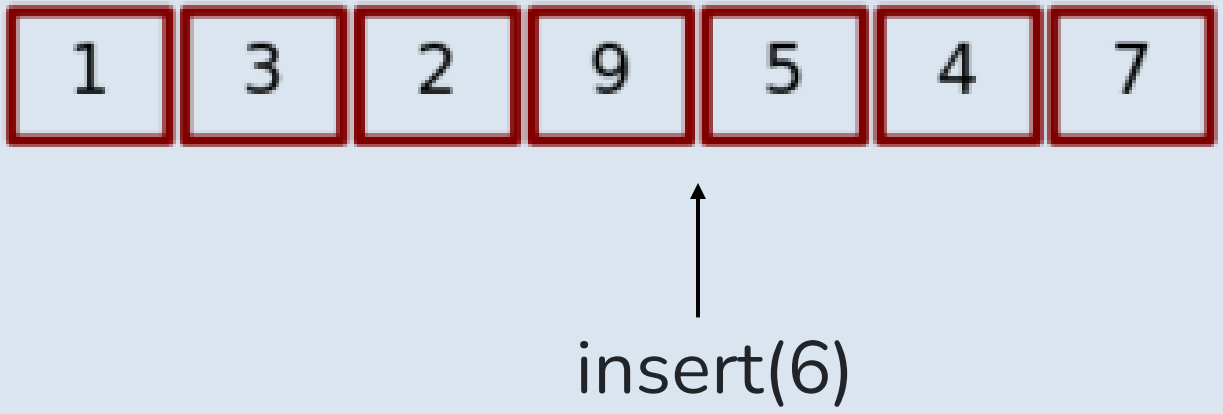
front / back



Πολυπλοκότητα: O(1)

insert

- Η συνάρτηση insert σε αντίθεση με το push_back, έχει γραμμική πολυπλοκότητα ανάλογη με το μέγεθος του vector την στιγμή της εισαγωγής



Πολυπλοκότητα: O(n)

Ταξινόμηση vector

- Όπως και στους πίνακες μπορούμε να χρησιμοποιήσουμε το έτοιμο quick sort σε ένα vector

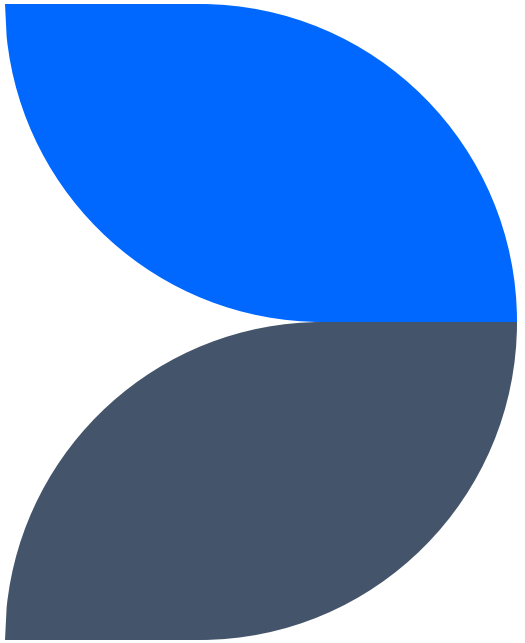
- sort(a.begin() , a.end());

- Πολυπλοκότητα: O(n*logn)

- Γράψτε πρόγραμμα που διαβάσει αριθμούς μέχρι να τελειώσουν τα δεδομένα εισόδου και τα τυπώνει ταξινομημένα

```cpp
int main(){

    vector < int > a;

    int t;
    while(cin>>t){
        a.push_back(t);
    }

    sort(a.begin(),a.end());

    for(int i=0;i<a.size();i++){
        cout<<a[i]<<" ";
    }


}
```

Range Sum Queries



Range Sum Queries

- **Πρόβλημα**: Δίνονται n ακέραιοι αριθμοί και Q ερωτήματα. Για κάθε ερώτημα δίνονται δύο αριθμοί a, b. Να τυπώσετε για κάθε ερώτημα το άθροισμα μεταξύ των θέσεων a, b συμπεριλαμβανομένων.

```
4 3
5 3 1 7
1 2    //8
3 3    //1
2 4    //11
```

Απλή προσέγγιση

- Για κάθε a, b εκτελώ for και υπολογίζω το άθροισμα

- Πολυπλοκότητα: O(n*q) .....μέτριο

```cpp
#include<iostream>
using namespace std;
int main(){

int n,q;
cin>>n>>q;

int a[n];
for(int i=0;i<n;i++){
    cin>>a[i];
}

while(q--){
    int o,oo;
    cin>>o>>oo;
    int s=0;
    for(int i=o-1;i<oo;i++){
        s+=a[i];
    }
    cout<<s<<endl;
}


}
```

Μία άλλη προσέγγιση

- Προϋπολογίζω για κάθε a, b το άθροισμα και για κάθε ερώτημα έχω έτοιμη την απάντηση

- Πολυπλοκότητα: O(n^2) .....πάλι μέτριο αλλά μπορεί να συμφέρει εάν είναι πάρα πολλά τα ερωτήματα (queries)

```cpp
#include<iostream>
using namespace std;
int main(){

int n,q;
cin>>n>>q;

int a[n];
for(int i=0;i<n;i++){
    cin>>a[i];
}

int p[n][n];

for(int i=0;i<n;i++){
    int s=0;
    for(int j=i;j<n;j++){
        s+=a[j];
        p[i][j]=s;
    }
}

while(q--){
    int o,oo;
    cin>>o>>oo;
    cout<<p[o-1][oo-1]<<endl;
}
}
```

Prefix sum

- Με το prefix sum μπορούμε με ένα προϋπολογισμό πολυπλοκότητας O(n) να βρίσκουμε το άθροισμα ενός διαστήματος σε O(1) **μόνο** όταν ο πίνακας είναι στατικός, δηλαδή οι τιμές του δεν αλλάζουν κατά την διάρκεια των ερωτημάτων.

- Προϋπολογισμός: Κάθε τιμή στον πίνακα θα αποθηκεύει το άθροισμα όλων των τιμών μέχρι την συγκεκριμένη θέση. Δηλαδή η τιμή στην θέση k θα είναι ίση με το άθροισμα από 0 μέχρι k.

Προϋπολογισμός

Για παράδειγμα υποθέστε τον επόμενο πίνακα:

0	1	2	3	4	5	6	7
1	3	4	8	6	1	4	2

Ο αντίστοιχος πίνακας προθεμάτων θα είναι ο επόμενος:

0	1	2	3	4	5	6	7
1	4	8	16	22	23	27	29

Απάντηση στα ερωτήματα

- Για κάθε ερώτημα a, b η απάντηση δίνετε από το
  sum (a, b) = sum(0, b) – sum(0, a – 1) , σε χρόνο O(1)

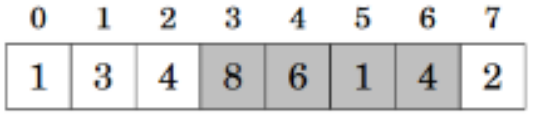
- Πολυπλοκότητα: O(n) ….πολύ καλύτερο

- Για περισσότερες πληροφορίες:

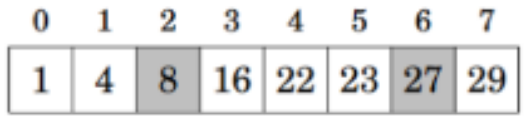
https://drive.google.com/file/d/1oaPRVSfRBzuWtLpXPHEUGQ1erHMJz_sE/view

Παράδειγμα

Για παράδειγμα, σκεφτείτε το διάστημα [3, 6]:



Σε αυτή την περίπτωση sum(3,6)=8+6+1+4=19. Αυτό το άθροισμα μπορεί να υπολογιστεί από δυο τιμές στον πίνακα προθεμάτων



Έτσι το άθροισμα είναι sum(3,6)=sum(0,6)-sum(0,2)=27-8=19.

```cpp
#include<iostream>
using namespace std;
int main(){

int n,q;
cin>>n>>q;

int a[n];
for(int i=0;i<n;i++){
    cin>>a[i];
}

int p[n+2]={};

for(int i=0;i<n;i++){
    p[i+1]=p[i]+a[i];
}

while(q--){
    int a,b;
    cin>>a>>b;
    cout<<p[b]-p[a-1]<<endl;
}


}
```

Problem Solving

Ασκήσεις

(sort and mod)

- https://www.hackerrank.com/contests/bubbles-bubbles/challenges/10-3

(prefix sum)

- https://cses.fi/problemset/task/1646

Περισσότερες ασκήσεις

- (easy alone)
- https://www.hackerrank.com/contests/bubbles-bubbles/challenges/6-12/problem

- (medium alone)
- https://www.hackerrank.com/contests/bubbles-bubbles/challenges/challenge-307/problem

- (hard alone)
- https://www.hackerrank.com/contests/bubbles-bubbles/challenges/7-6-30/problem

Τέλος Day 2

Juniors Summer Camp 2023