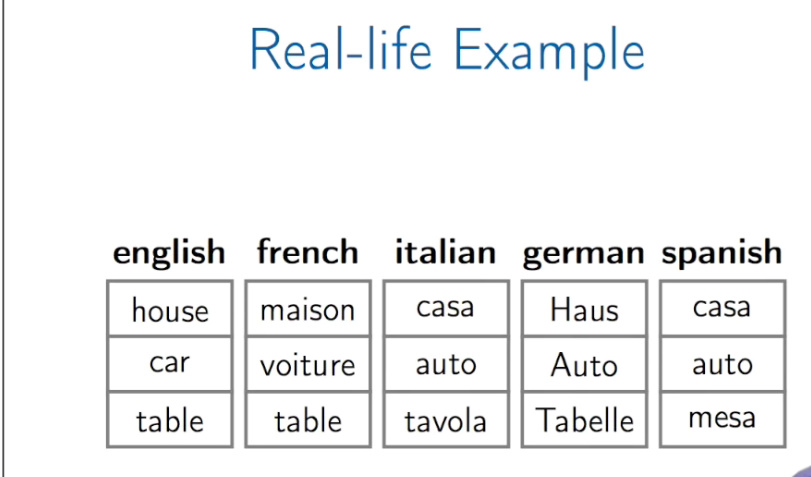
**Divide and Conquer**

* **divide and conquer** – divizeaza si invinge. Provine din antichitate, cand se spunea ca e mai usor sa invingi inamicii divizati in grupuri.
* **divide and conquer alghoritms** separa o problema in una sau mai multe care pot fi rezolvate separat.
* Ca si algoritmul greedy, Divide and Conquer nu poate rezolva orice problema.

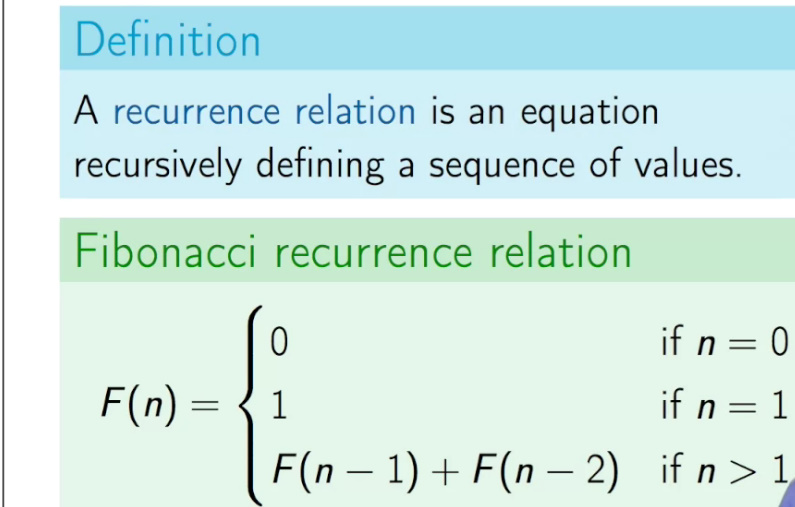
**Cerinte**:

* Toate subproblemele trebuie sa fie de acelasi tip pentru a putea folosi acest algoritm!
* Subproblemele trebuie sa nu se suprapuna, adica sa nu se intersecteze.
* Solutiile la subprobleme trebuie sa poata fi combinate intr-o solutie finala pentru problema generala.
* Deci problema trebuie divizata in subprobleme ce sunt independente si apoi ele la fel se rezolva independent.
* Odata ce am rezolvat subproblemele, combinam solutiile in una singura pentru problema originala.
* Asa cum fiecare subproblema e de acelasi tip ca celelalte, putem sa rezolvam fiecare recursiv, folosind aceeasi strategie.

**Linear Search**

* Se bazeaza ca daca vrem sa gasim un element dintr-o lista, se ia fiecare element din lista rand pe rand pana se gaseste cel dorit.
* 
* De ex, aici avem un traducator. Daca userul vrea sa traduca din engleza in franceza cuvantul car, Linear Search se bazeaza pe faptul ca in lista la english se va cauta acest cuvant, si odata ce va fi gasit, se va memora pozitia si apoi se va returna cuvantul de pe aceeasi pozitie din lista french.
* Daca ar fi zeci de mii de cuvinte, Linear Search ar lua foarte mult timp.

**Relatie de Recurenta**

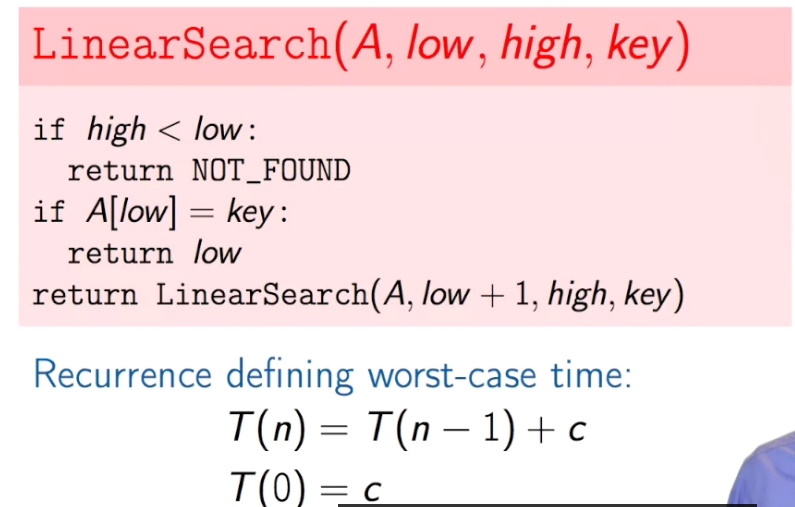


**Relatie de recurenta** – ecuatie care defineste recursiv o secventa de valori.

* Orice problema de recursivitate are nevoie de o solutie de baza, la care sa opreasca recusivitatea.
* T(n) – worst case --> O(n), doar ca se refera la relatii de recurenta, adica de recursie
* Cel mai rau caz in cautarea prin lista e cand cuvantul nu e gasit.

**Complexitatea**

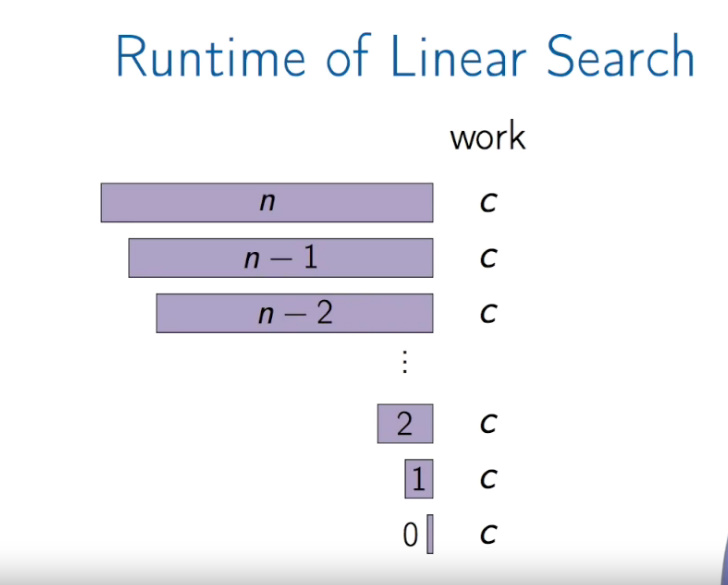
Fie algoritmul recursiv de cautare:



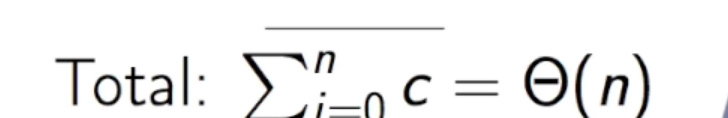
c – constanta de volum de lucru, si se refera la acele if-uri, care mereu iau acelasi nr de instructiuni si returnarea rezultatelor unde au fost chemate

T(n-1) – se bazeaza pe numarul de returnuri, adica de recursii, dar acesta e n-1, asa cum primul element deja s-a calculat in elementele din c, gen daca avem 100 de numere, in return de jos se va foloi intai 99, apoi 98 si tot asa,

In cel mai bun caz, T(0) = c, cand array e empty



Insa:

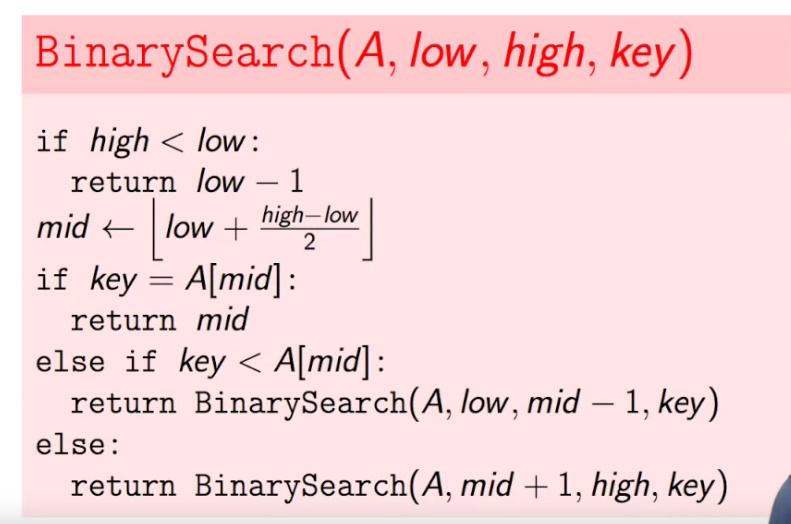


E Teta, deoarece rata de crestere a instructiunilor constante, c, e oricum mereu identica, deci de aia e teta de n, unde n arata de cate ori aceste c se repeta

* Adesea, algoritmii recursivi sunt trasformati in iterativi, asa cum recursia e foarte lenta.

**Binary Search**

* Consta in divizarea in jumatate a listei unde cautam elementul si deciderea daca divizam din nou sau nu partea stanga sau dreapta a listei, incepand cu elementul mijlociu.
* Un exemplu de Binary Search e dictionarul, unde cuvintele sunt sortate dupa literele pe care le au.



A – array sortat

low – punctul de start

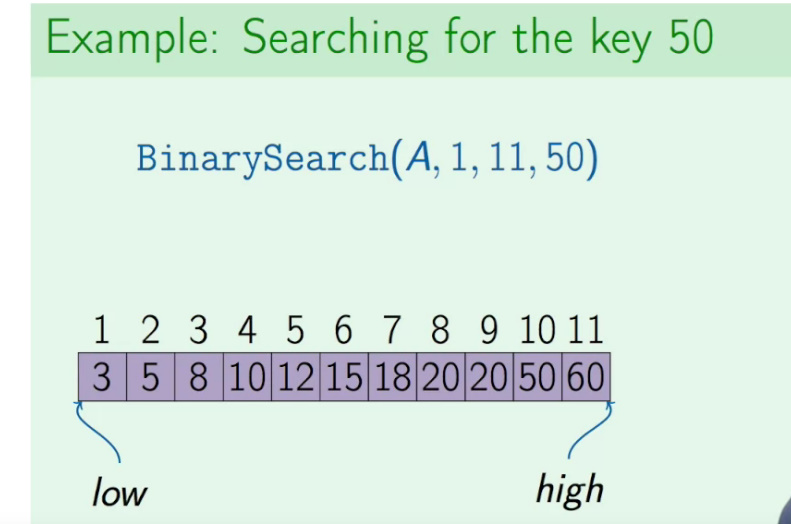
high – punctul final

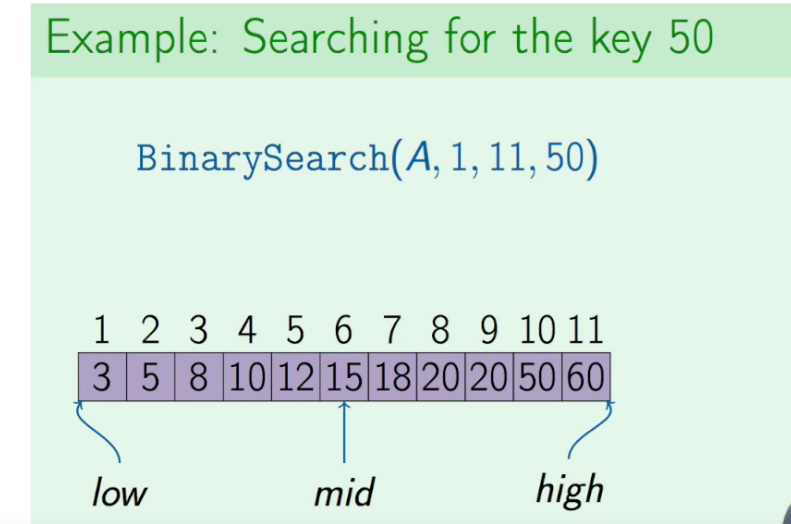
key – elementul ce il cautam.

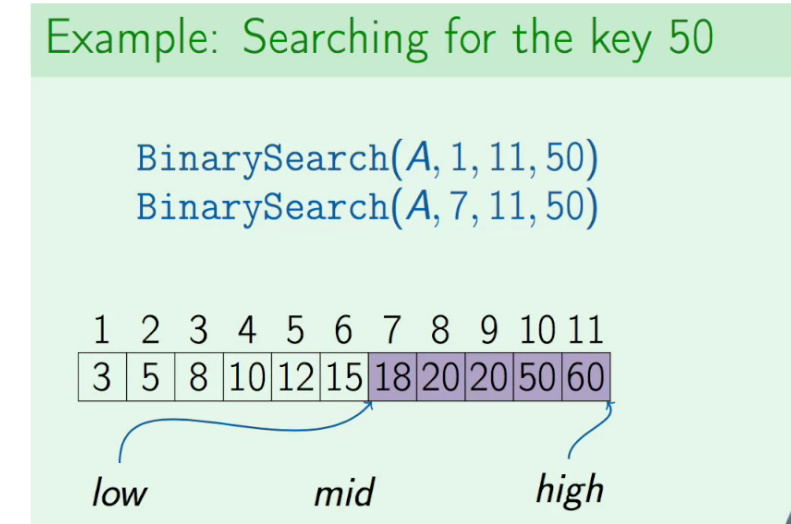
Algoritmul dat se bazeaza pe aceea ca nu am cautat elementul in tot array(care trebuie sa fie sortat), dar mereu impartim array in jumatate, si daca elementul cautat e mai mare ca elementul din mijloc, il cautam in dreapta lui, dar daca e mai mic, atunci in stanga lui. Dupa care iar divizam intervalul avut de la low la high in 2 si tot asa cautam.

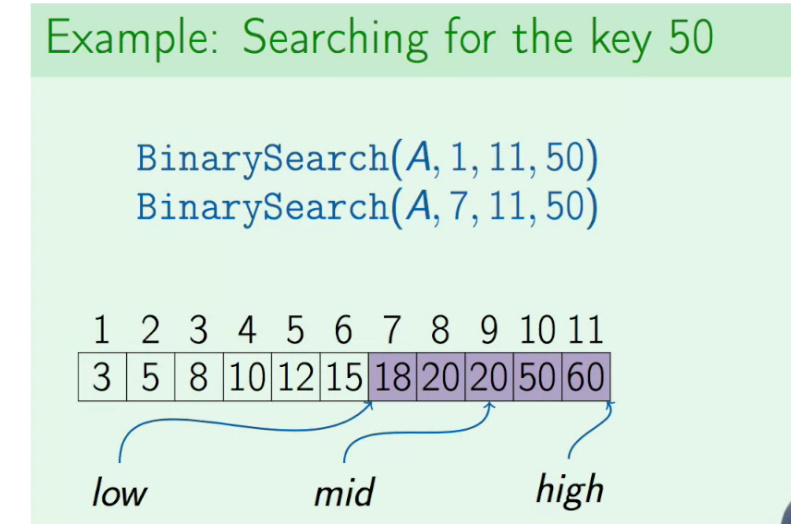
Cand deja am trecut de limita de inceput, adica high<low, returnam low – 1

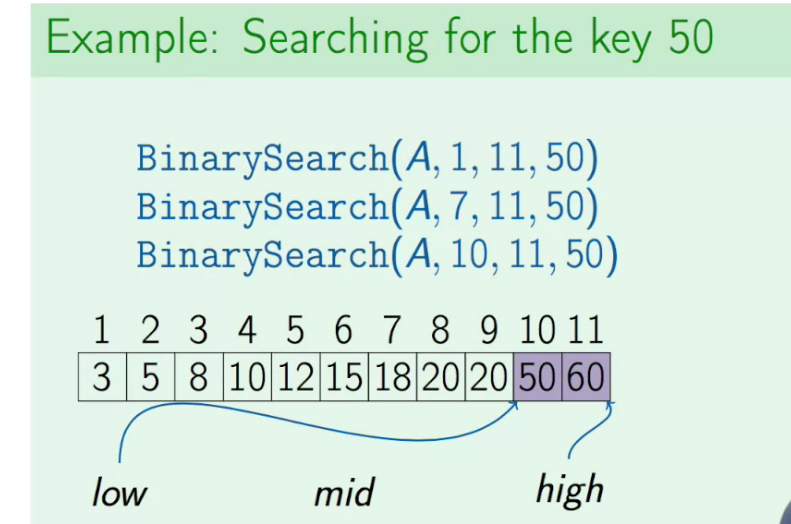
Daca intervalul nu se imparte exact, merge cu o pozitie inainte fata de pozitia sa obtinuta cand se imparte la 2.

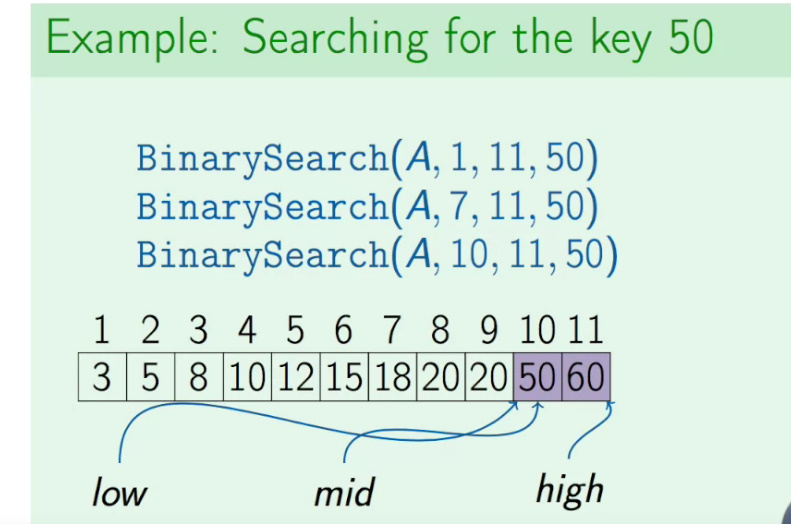






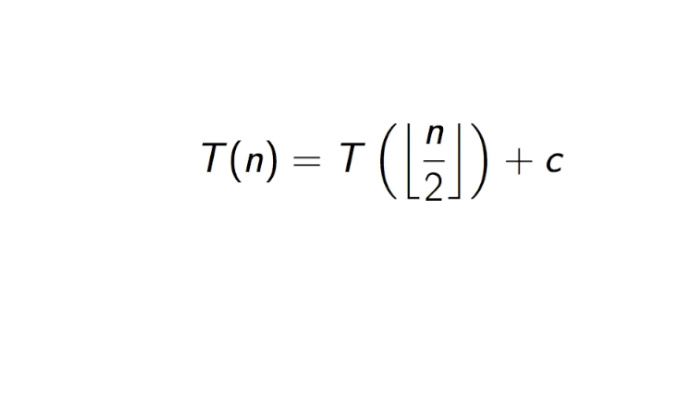




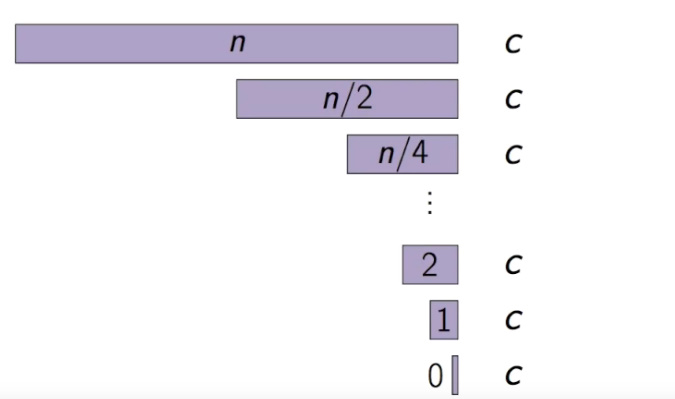


Aici am descompus problema generala in subprobleme ce sunt de acelasi tip si a caror solutii le putem include in cea finala.

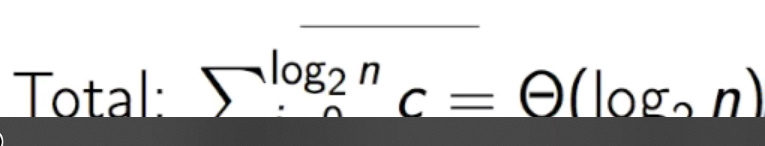
**Binary Search Runtime**

Aici avem nevoie sa aflam relatia de recurenta, asa cum avem recursie.

Deci asta e cel mai rau caz posibil.

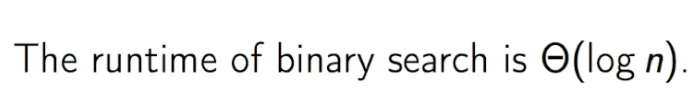


Deci avem O(n) = log2(n+1) Mai avem si +1, deoarece se mai adauga un pas cand deja low > high si acolo se opreste programul

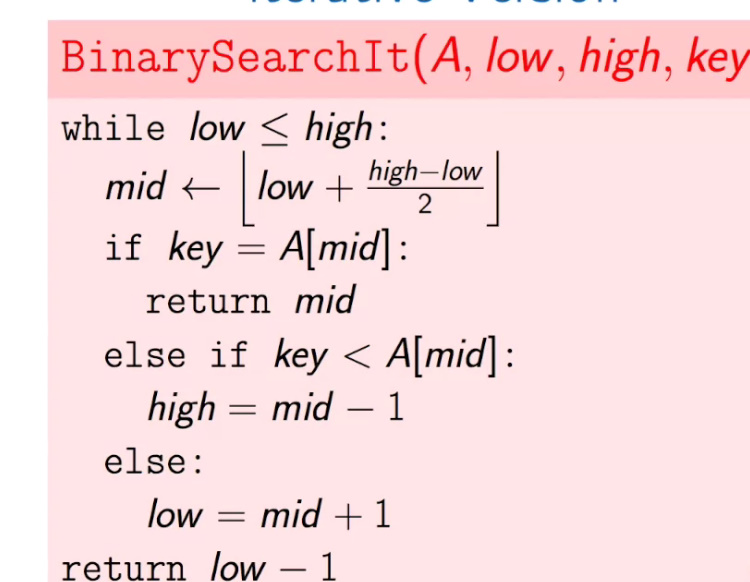


Scapam de 1, deoarece la acea iteratie unde se opreste programul, nu se face tot din c, adica doar un if e executat,si de aia si omitem acest +1

* Putem afla totusi:



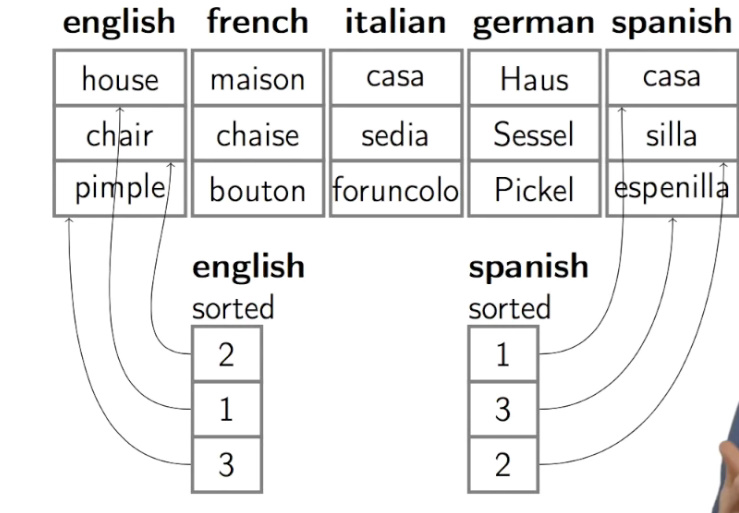
**Versiunea Iteratica**



**Problema cu translate**



* Nu putem sorta aici cuvintele dupa alfabet, caci fiecare limba ar sorta diferit si se pierde legatura de linie dintre ele, adica ordinea de inserare.
* Putem insa sa cream un array cu indexi, in care fiecare index corespunde unui pointer cu numarului liniei in care se gaseste cuvantul in dictionar ce are referinta la acel cuvant, si sa sortam array dupa ordinea pe care ar trebui sa o aiba cuvintele in dictionar.



De ex, “chair” e al 2 in dictionar, dar va avea un pointer la el ce va fi primul in array, asa cum chair ar trebui sa fie primul in dictionar. Fiecare limba are propriu array sortat dupa ordinea pe care cuvintele ar trebui sa le aiba in dictionar, insa in dictionar ele raman asa cum sunt.