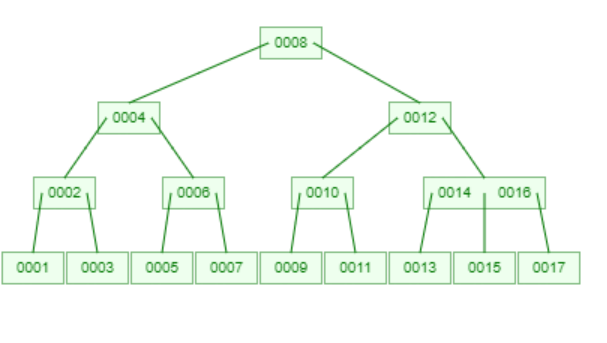
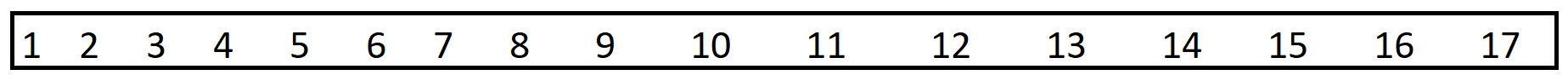
Rezolvari Restanta SD

1. **Ce înălţime poate să aibă un B-arbore cu 17 elemente? Justificaţi! Desenaţi arborele de înălţime minimă şi cel de h maximă.**





**2. Câte noduri poate avea un heap de înălţime 4. Un heap de inălţime 0 are 1 nod, rădăcina. Un heap de înălţime 1 are 2 sau 3 noduri...**

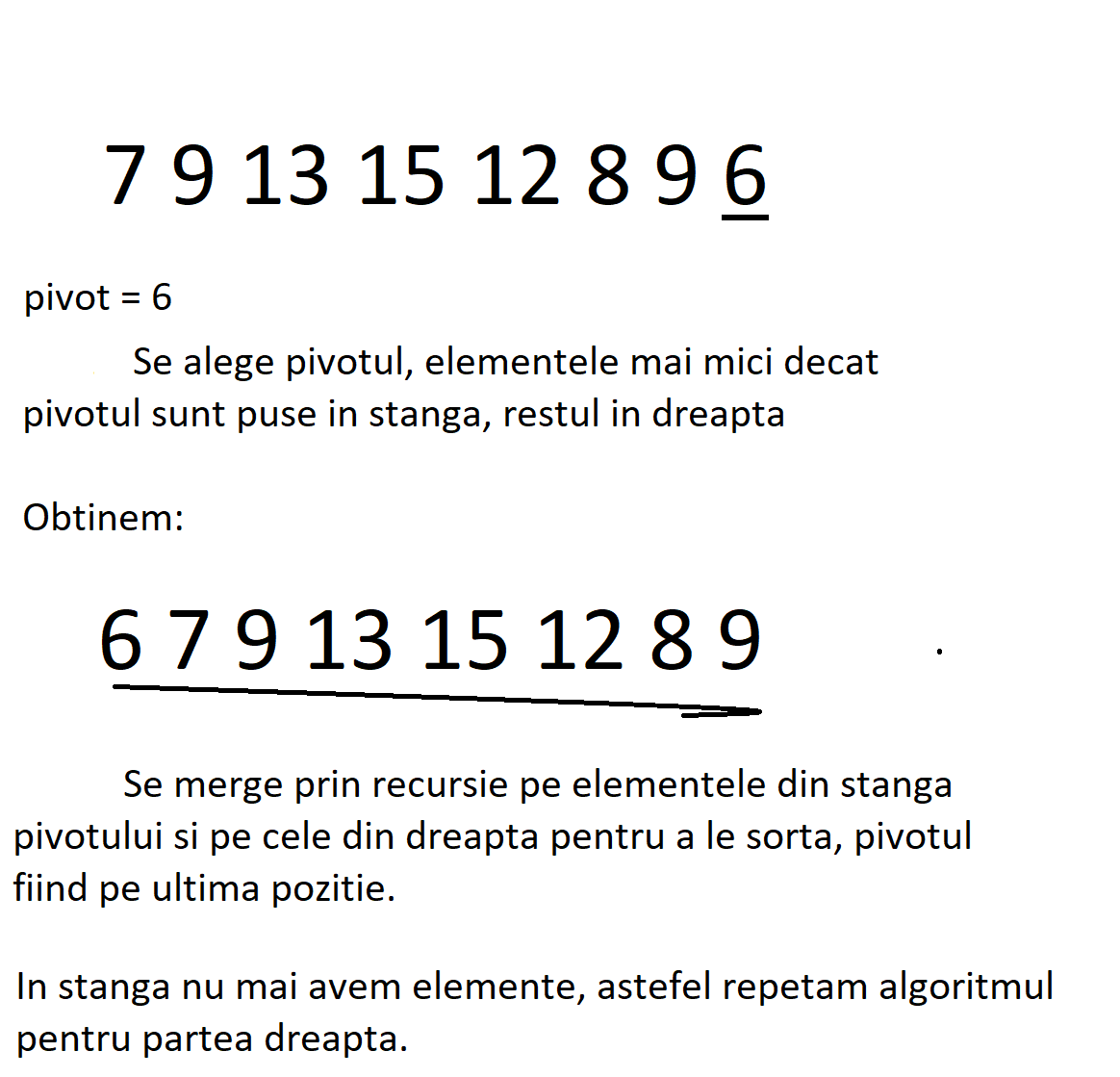
Un heap poate avea minim si maxim noduri.

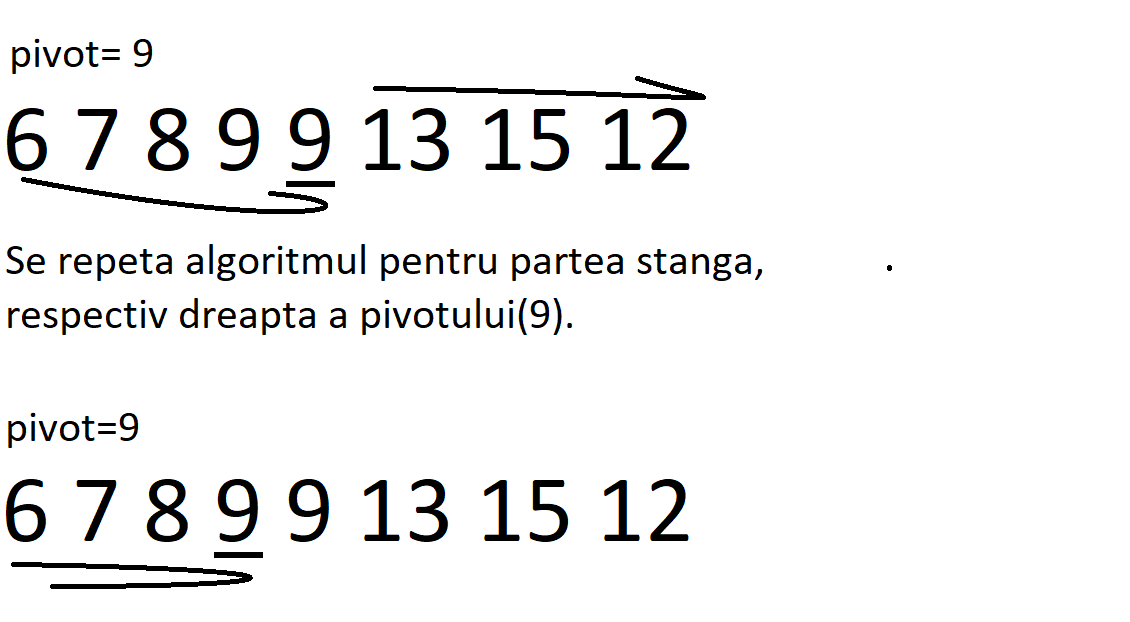
Deci, pentru h=4, Numărul de noduri este :

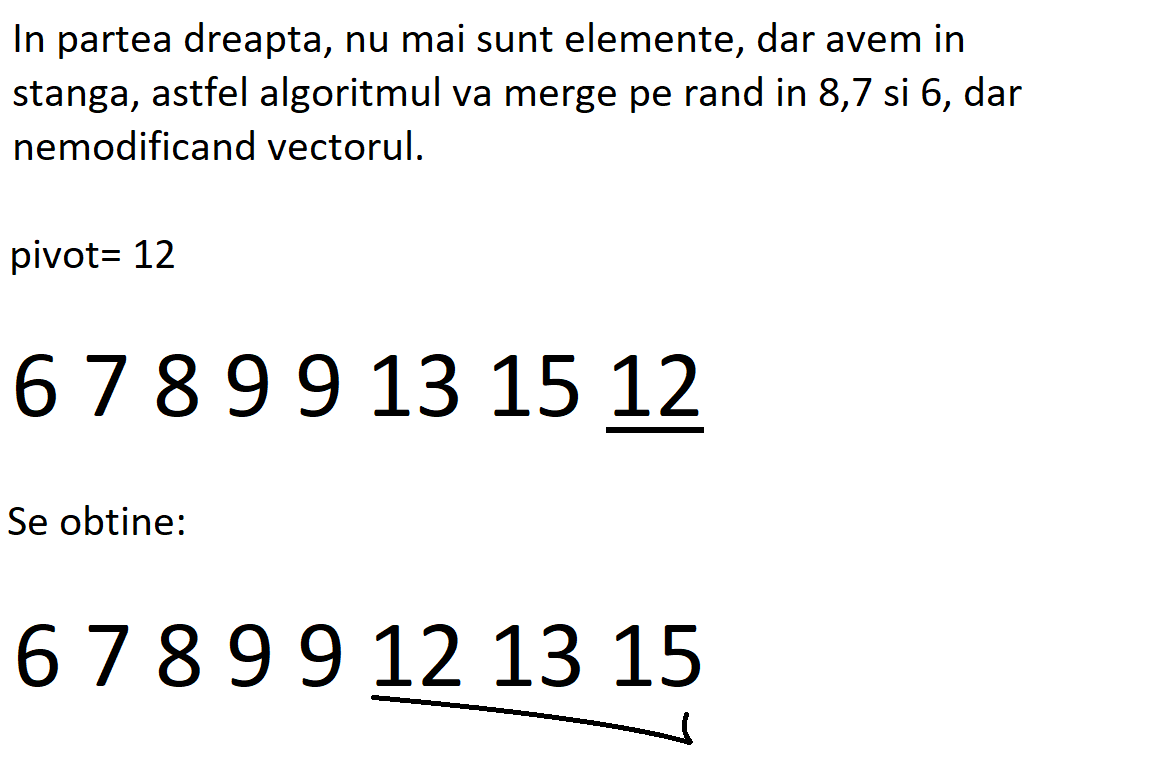
16<= n <=31

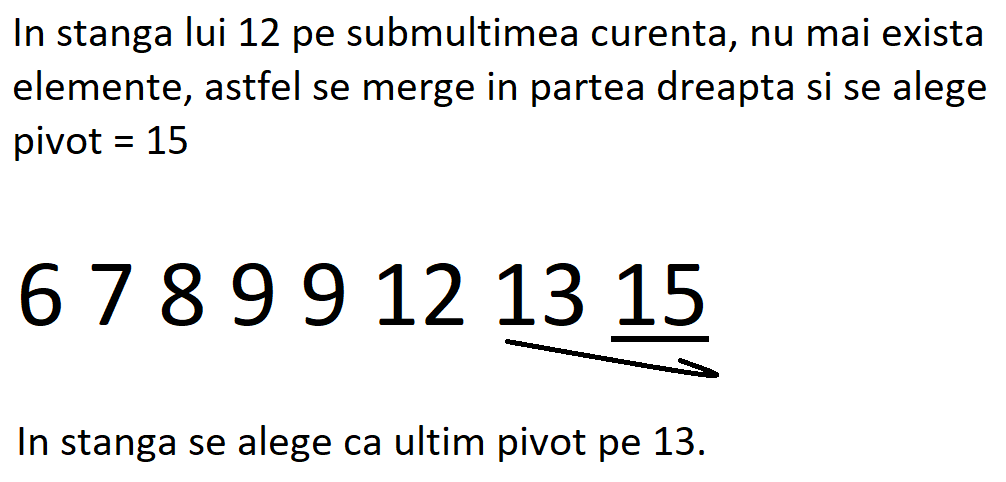
**3. Exemplificaţi cum funcţionează quicksort(pivot ales ca ultim element din secvenţă) pe vectorul**

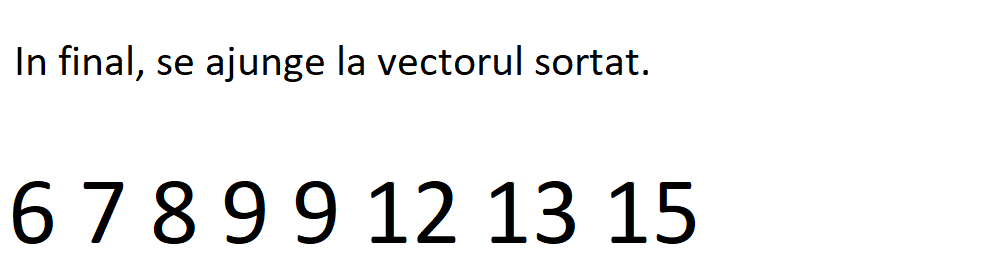
**7 9 13 15 12 8 9 6**



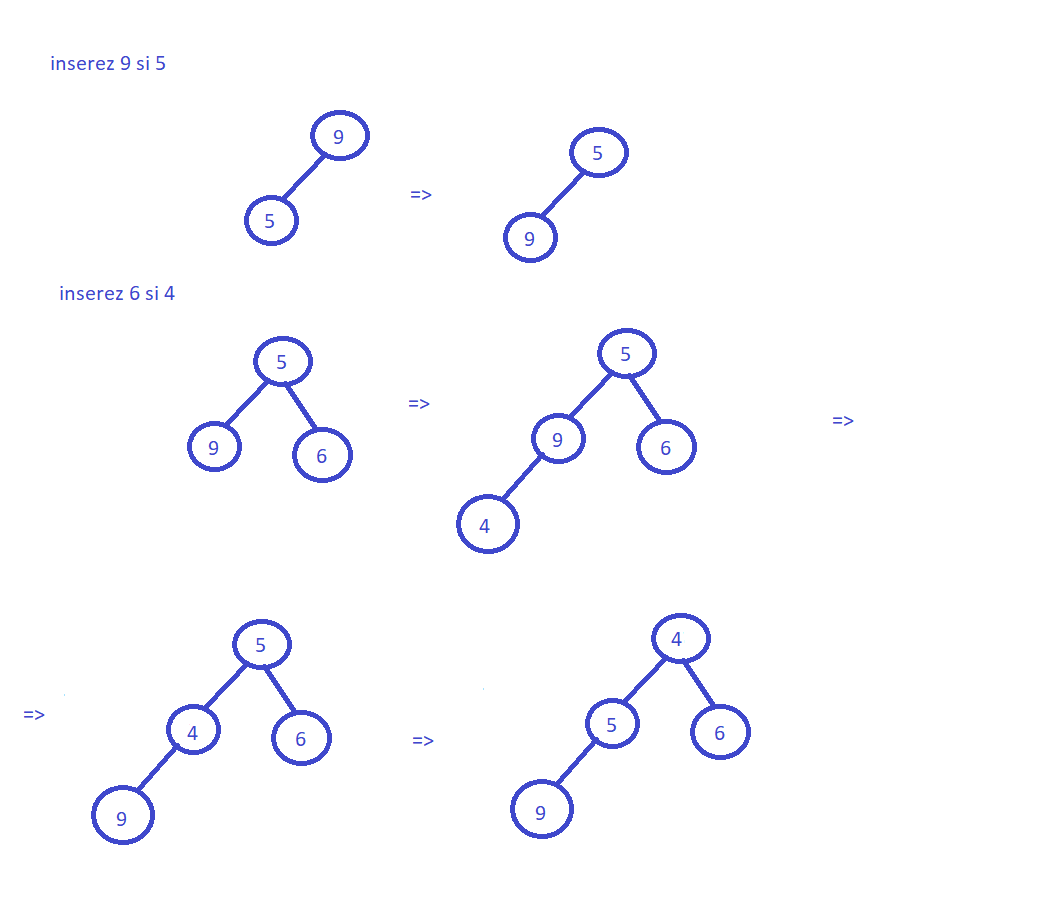


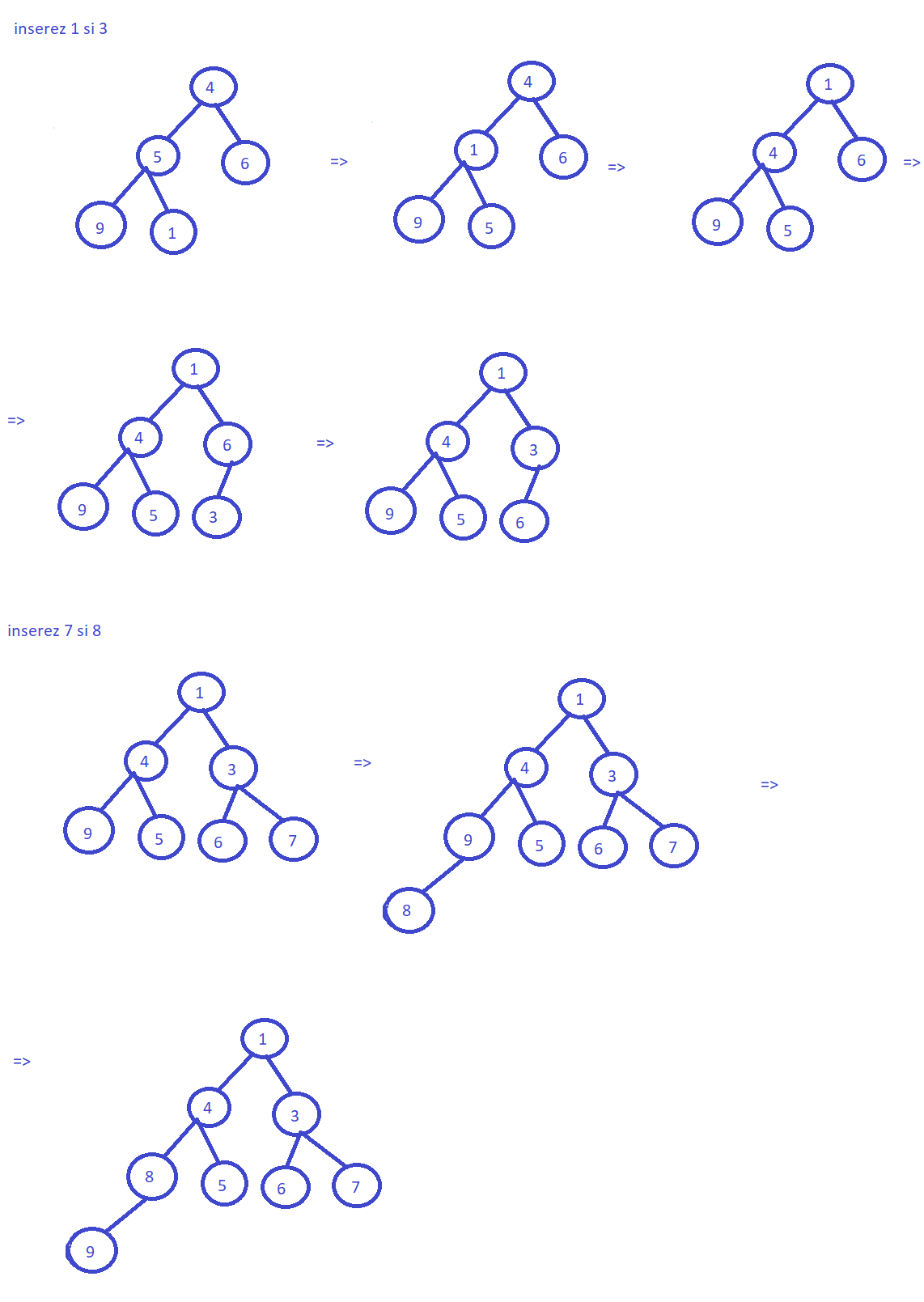






**4. Inseraţi pe rând într-un min-heap valorile 9 5 6 4 1 3 7 8**





**5. Dacă vrem să sortăm 1.000.000 numere reale mai mici egale cu 50.000.000.000 ce algoritm ar fi bine sa folosim ? De ce ?**

Pentru a sorta eficient 1000000 de numere reale mai mici sau egale cu 50000000000 putem folosi algoritmul Quick Sort obtinand astfel o complexitate O(n\*log n). Putem alege pivotul cu orice metoda, dar cel mai eficient este folosind mediana in 3. Nu se poate folosi Radix Sort pentru ca este vorba de numere reale.

**6. Cât ne costă să găsim al doilea cel mai mic element dintr-un heap ? Cum il găsim ?**

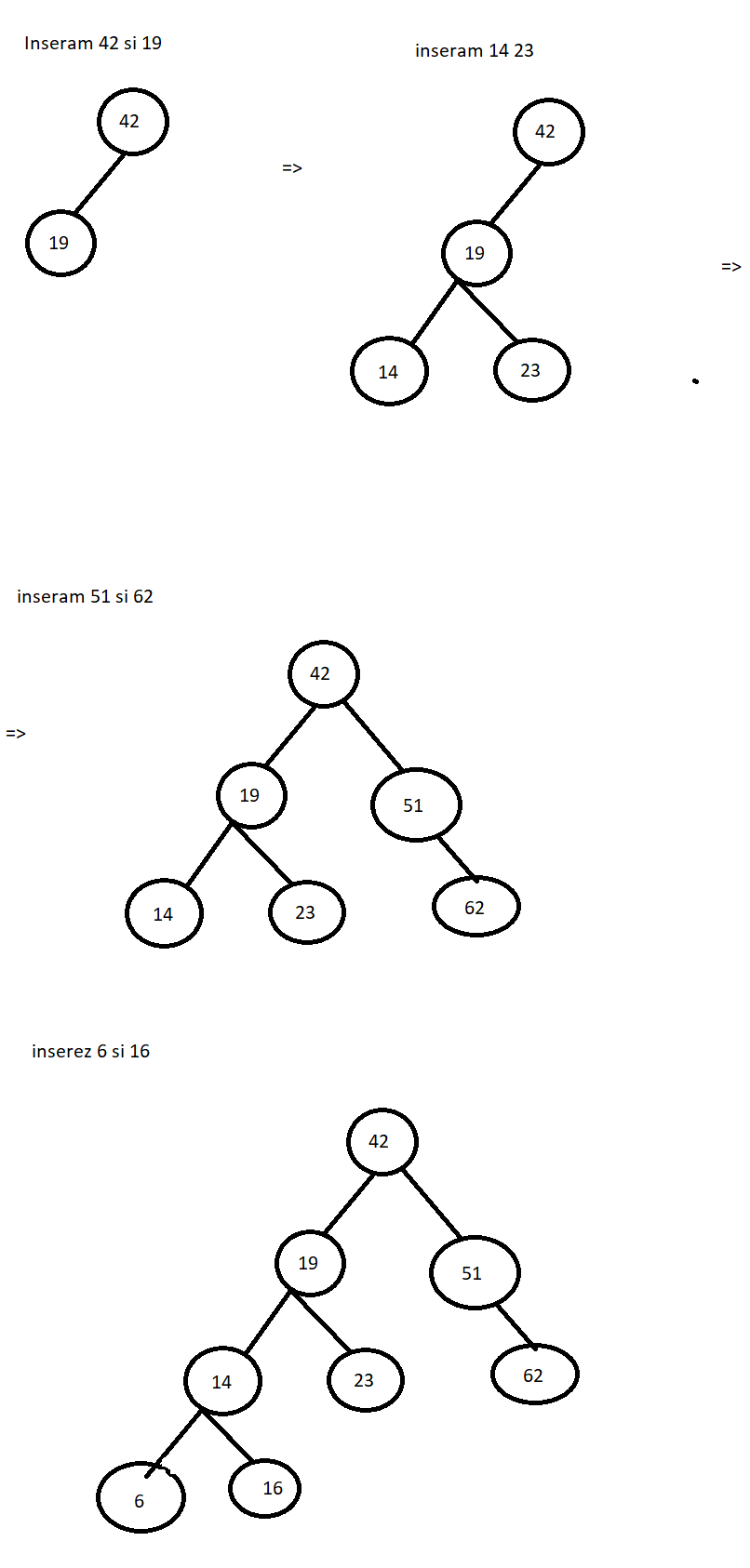
Intr-un heap de minim aflam al doilea cel mai mic element in O(1) deoarece acesta este unul dintre cei doi fii ai radacinii.

**7. Inserati intr-un hash valorile 11, 21, 31, 42, 81, 167, 291, 344, 817 folosind functia de dispersie x%10. Puteti alege cum rezolvati coliziunile, motivati decizia.**

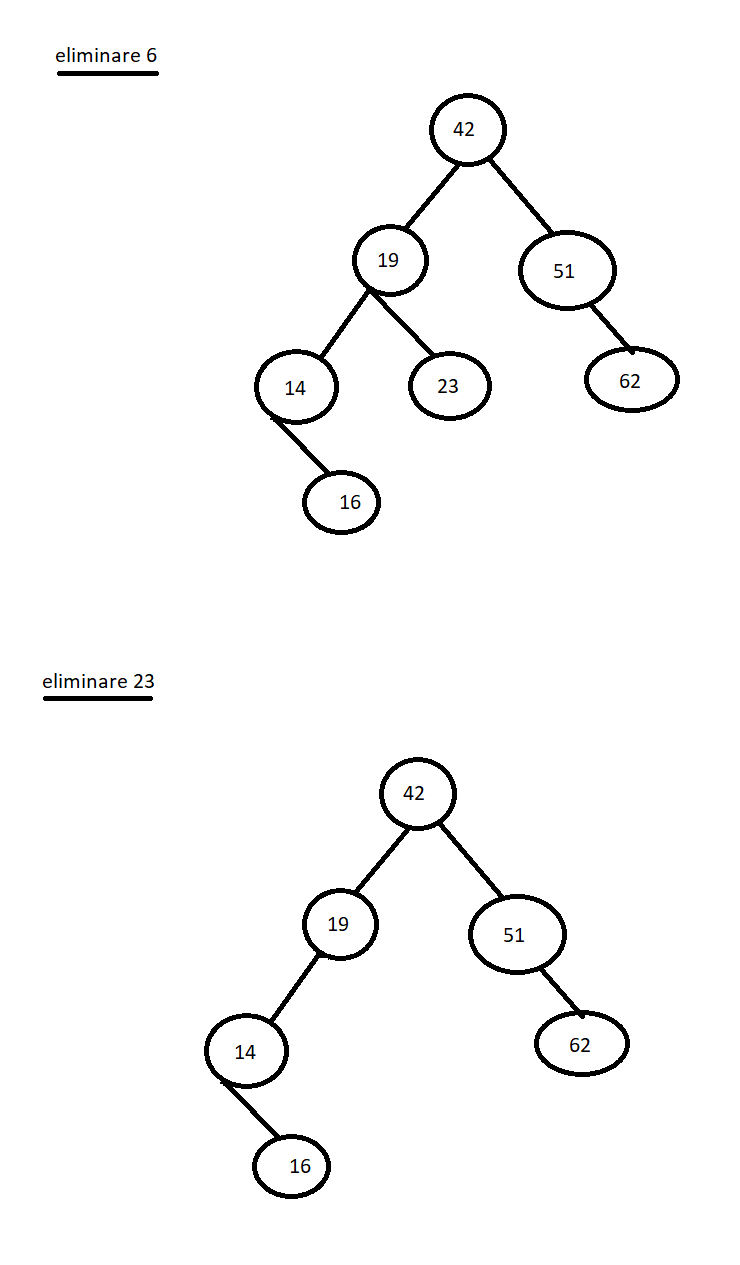
|  |
| --- |
| 0 |
| 1 11 -> 21 -> 31 -> 81 ->291 |
| 2 42 |
| 3 |
| 4 344 |
| 5 |
| 6 |
| 7 167 -> 817 |
| 8 |
| 9 |

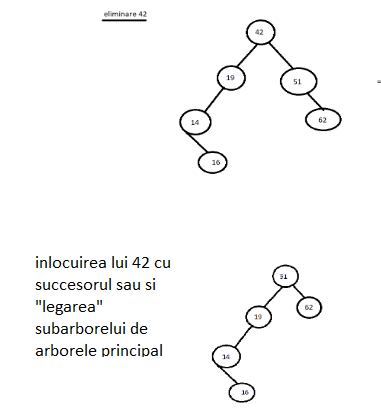
Coliziunile pot fi rezolvate prin implementarea unei liste inlantuite, in cadrul careia legam valorile ce se suprapun. Astfel, intr-un caz pur teoretic, toate cele n valori ar putea fi repartizate in aceeasi locatie, insa in cazuri practice lungimea medie a celui mai lung lant este de lg(n). Memorie O(n). Accesare rapida.

**8. Inserati intr-un arbore binar de cautare urmatoarele valori: 42 19 14 23 51 62 6 16**

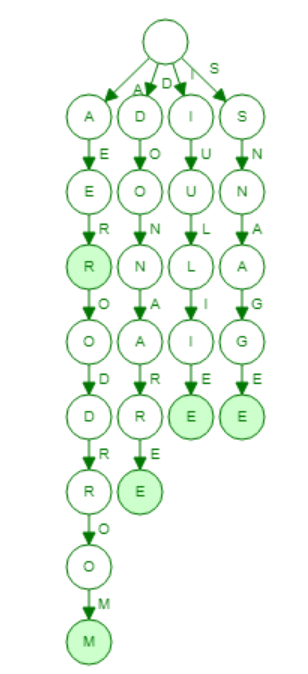


**9. Sterge-ti pe rand 6 23 42 14 din acel arbore.**





**10. Inserati intr-un trie cuvinetele: aer, aerodrom, iulie, donare, sange**



**11. Explicati ce face LA si aratati cum functioneaza pe un arbore cu macar 8 noduri.**

**Lowest ancestor** se referă la al k-lea părinte al nodului și este o sugestie a rmq-ului. Se țin minte părinții de grad 2,4,8,..2^n pentru fiecare nod, și se accesează pe sărite. Ex: al 13-lea parinte = parinte[parinte4[parinte8[x]]] (parintele celui de-al patrulea parinte celui de-al 8-lea parinte al lui x).

Obtinem astfel O(n log n) pe pre-procesare si O(log n) pe query. Tehnica se numeste Binary Lifting.

**12. Explicati cum putem raspunde cu RMQ in O(1) cu preprocesare O( n log n) la queriurile 1,7 respectiv 4,8 in vectorul 1, 2, 3, 4, 0, 5, -2, 9, -5, 11**

Folosim efectiv tehnica RMQ care are preprocesare de O(nlogn) calcuand intervale de 2 cate 2, fiecare linie calculandu-se folosind 2 elemente de pe randul anterior, astfel obtinem intervale care sunt puteri ale lui 2, si pentru a eficientiza calculul pe un interval incercam sa il impartim in bucati de lungimi puteri ale lui 2

query pe (1,7) il impartim in bucatile (1, 4), (3,7) care se iau din matricea obtinuta din preprocesarea RMQ-ului, prima bucata fiind 2^2 mergem pe linia 2, (3,7) este 2^2, deci linia 2 si comparam bucatelele astea precalculate in loc sa comparam toate elementele. Coloana pe care mergem este log(I) unde intervalul este (I,j).

query-ul (4,8) este direct 2^2 deci are rezultat direct.

**13. Se da un vector cu valori intregi. Eliminati dublurile.**

Unordered Map(hashing)

Putem rezolva problema folosind Unordered\_map, introducand pe rand valorile neduplicate in acest map, iar apoi cautand in mod secvential prin map daca mai exista acel element. Eficienta vine din modul de implementare a acestei structuri de date ( datele sunt stocate folosind un arbore binar si operatiile insert() si find() ne costa O(1) ) . Pe masura ce gasim elemente unice in map, putem modifica si vectorul initial pentru a contine valorile neduplicate.

Complexitate timp: O(n)

Complexitate spatiu: O(n)

<https://slaystudy.com/remove-duplicates-from-unsorted-array-c/#using-hashing>

**14. N studenti numerotati 1, 2, … n sunt intr-un cerc. La primul pas e eliminat primul, la al doilea pas sarim 1 student si il eliminam pe urmatorul ( adica al 3-lea), la al doilea pas sarim 2 elemente si eliminam urmatorul (al 6-lea daca am minim 6 studenti). Gasiti ordinea eliminarii studentilor. Pentru n=5 ordinea e 1, 3, 2, 5, 4**

Liste inlantuite

Cream o lista inlantuita cu n elemente ( de la 1 la n ), apoi traversam aceasta lista si pe masura ce eliminam fiecare nod cerut, il afisam.

Complexitate: O()

<https://tutorialspoint.dev/data-structure/linked-list/josephus-circle-using-circular-linked-list>

Pentru punctaj maxim se folosesc **arbori de intervale** (<https://infoarena.ro/problema/order> pentru testare )

**15. Se da o matrice cu n\*m elemente.**

**a) Aveti de raspuns cat mai eficient la p query-uri de forma: care este cmmdc-ul pe dreptunghiul x1, y1, x2, y2**

**b) Rezolvati si in cazul in care exista si update-uri intercalate cu query-uri. Intr-un update se modifica valoarea unui element din matrice.**

Solutia banala

for( l =x1; l <= x2; ++l)

for (L2 = I; L2 <= y2; L2++)

{

cmmdc = A[l][L2];

cmmdc = cmmdc(cmmdc, A[l][L2]);

}

O() - pentru un query , O(\*p) pentru p queries

Pentru b) se fac update-urile in matrice. O(1)

Complexitate finala: O() - pentru un query

O(\*p) - pentru mai multe queries