A. Subalgoritm iterativ de complexitate $O(n^3)$

Subalgoritm găsește (V , n , e) este

{ pre: V este un vector de dimensiune n^3
n , e ∈ ℕ
post: se returnează 1 dacă s-a găsit e în vector și 0 , în caz contrar
}

Pentru i = 1, n^3 execută
Dacă v[i] = e atunci
găsește ← 1
SF Dacă
SF Pentru
găsește ← 0
SF Subalgoritm

Caz favorabil -> când elementul căutat este pe prima poziție: $\Theta(1)$

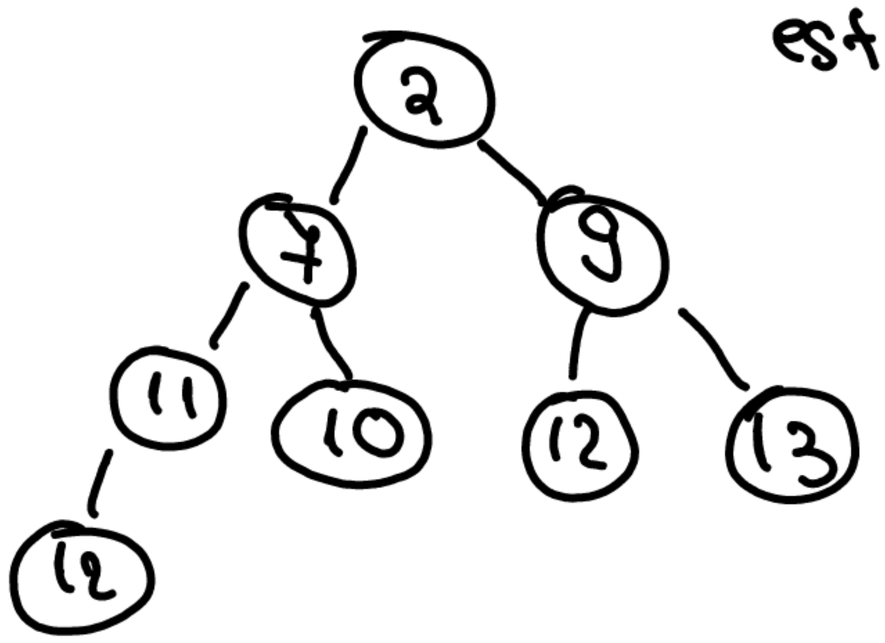
Caz defavorabil -> când elementul nu se găsește în șir $\Theta(n^3)$

Caz mediu -> $\frac{1+2+\dots+n^3}{n^3} = \frac{(n^3+1)\cdot n^3}{n3} \in \Theta(n^3)$

=> Complexitatea este $O(n^3)$

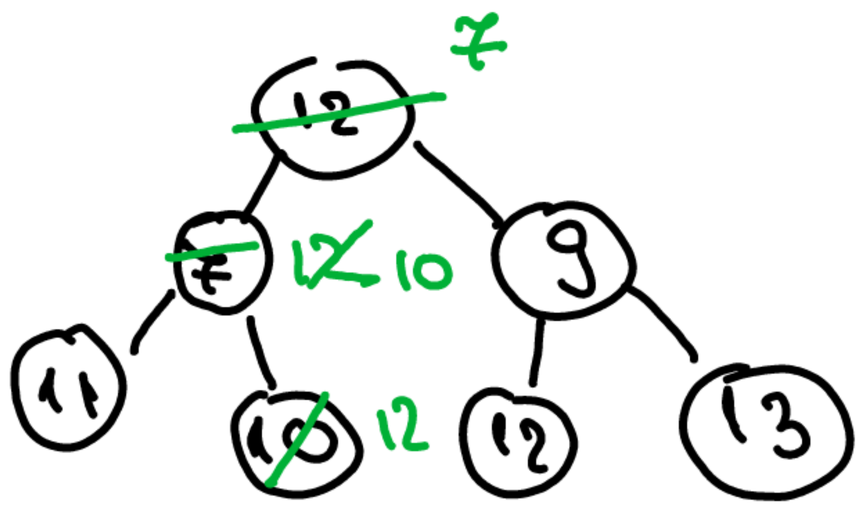
B.

Ansamblul inițial -> se observă că acesta este un MIN HEAP

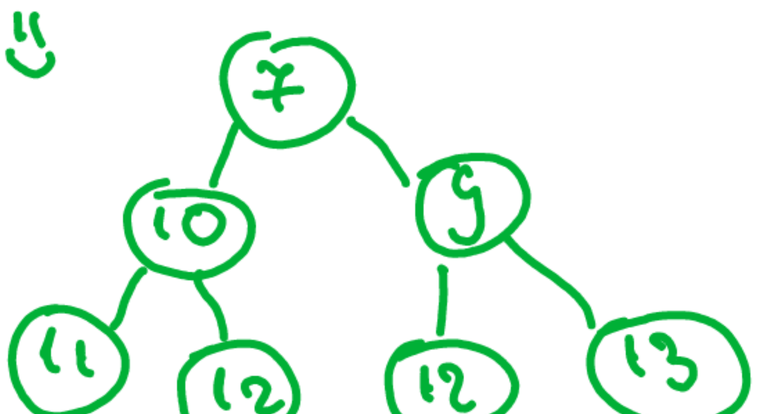


Prima operație de ștergere

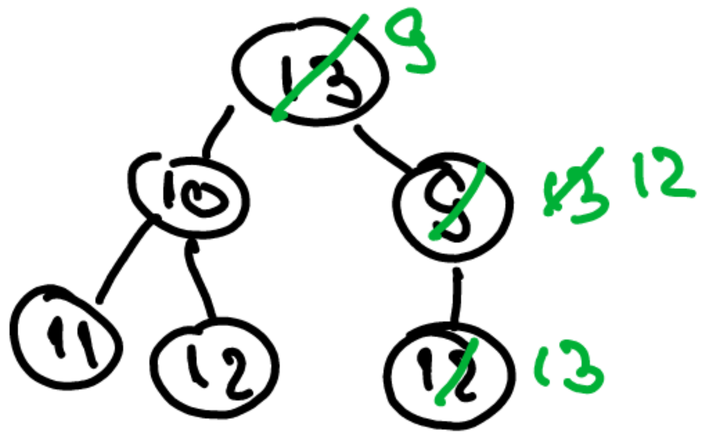
- se elimină rădăcina: 2 -> în acest înlocuită de ultimul element: 12



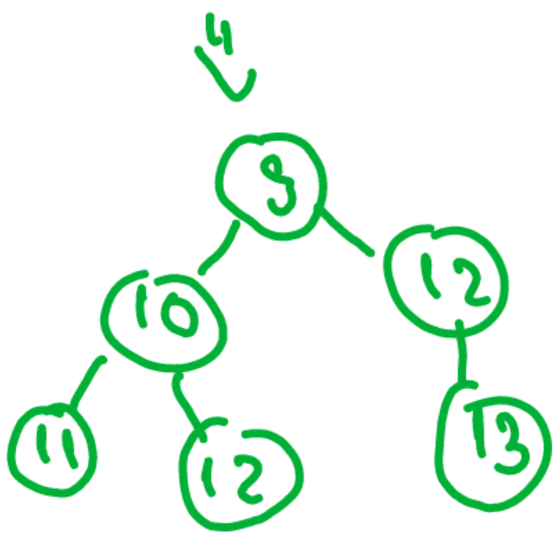
- pentru a restabili proprietatea de ansamblu, va fi necesar să-l coborâm pe 12
  - 7 < 12 ⇒ coborâm pe 12
  - 10 < 12 ⇒ coborâm pe 12



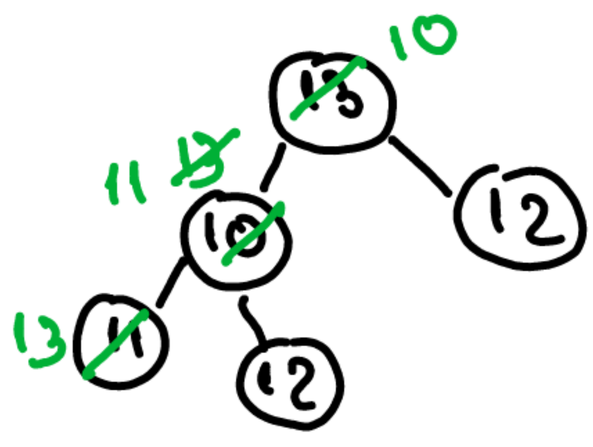
A doua operatie de stergere



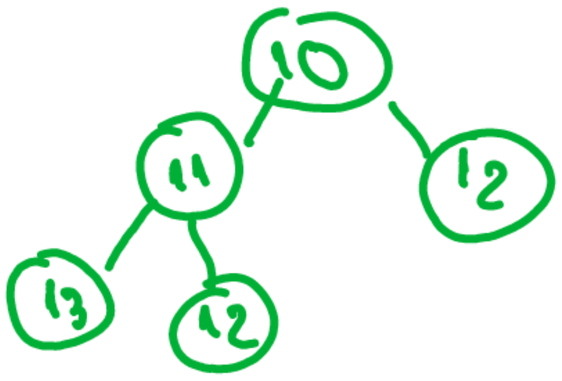
- 9 < 13 => coboram pe 13
- 12 < 13 => coboram pe 13



A treia operatie de stergere



- 10 < 13 => coboram pe 13
- 11 < 13 => coboram pe 13



c1. Elementele vectorului pot fi sortate crescător folosind BucketSort în $\Theta(n)$. Acest lucru se datorează faptului că, complexitatea generală a algoritmului de sortare este $\Theta(N+n)$, unde N reprezintă cheia maximă (elem. maxim - 1), iar n nr. de elemente (pt. că la final, se parcurg cele N existe dintr B și, implicit, toate cele n elem. ale vectorului inițial). În cazul de față, N are valoarea fixă de 1990 $\Rightarrow \Theta(n+1990) = \Theta(n)$.

c2. TD în care colecțiunile se rez. prin încadrare

Afirmația făcută este una falsă, deoarece factorul de încărcare al tabelei $\alpha = \frac{n}{m}$ ($n \rightarrow$ nr. de elemente, $m \rightarrow$ dimensiunea tabelei) ar presupune că elementele sunt împărțite în mod egal între cele m locații.

Totuși, în cazul defavorabil, timpul de execuție poate ajunge la $\Theta(m+n) \simeq \Theta(n) \simeq \Theta(m \cdot \frac{n}{m})$

Cum se consideră?

5.

Array :

e : TElement[]

n : Intreg (dimensiunea vect. de elemente)

Subalgoritm cod (a, e) este

{pre: a e Array

e e TElement

post: se va returna un cod dacă e se află

în vector sau excepție dacă nu

}

{ corpul în care array-ul e vid }

Dacă a.n = 0 atunci

@ excepție "Nu a fost găsit elementul"

SFDacă

String <- "1" { codul cerut va fi stocat în var string }

Dacă e = a.e[1] atunci

cod <- string

SFDacă

Pentru c execută { se parcurge c }

adauga (c, {1, String})

{ se adaugă în coadă o pereche de {indice, String} }

Cât Timp Nevida(c) execută

{ sterge (c, {indice, S})

Dacă indice * 2 ≤ a.n atunci

{ ne aflăm în subarborele stâng }

Dacă a.e[indice*2] = e atunci

cod ← @ concatenare s și "2"

SfDacă

adaugă(c, 2*indice, s+"2")

SfDacă

Dacă indice*2+1 ≤ a.n atunci

{ne aflăm în subarborele drept}

{verificăm să vedem dacă am găsit elementul}

Dacă a.e[indice*2+1] = e atunci

cod ← s+"0"

SfDacă

↑ poate ar trebui menționată cop.

adaugă(c, 2*indice+1, s+"0") cu @ concatenare

SfDacă

SfCâtTimp

{dacă s-a ajuns la frunză, înseamnă că nu am}

{găsit elementul}

@ excepție "Nu s-a găsit elementul"

SfSubalgoritm

Complexitate: de timp: O(n)

caz fav: găsim elem pe prima poz: Θ(1)

caz defav: nu găsim elem: Θ(n+1)

caz mediu: $\frac{1+2+\dots+n}{n} = \frac{(n+1)\cdot n}{n} \in \Theta(n)$

de spatiu : $\Theta(n)$
(adițional)