A. Cost Rau = Cost defau = Cost mediu

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2^{i-1}) + n^2 \cdot (i \% 2 + 1) \end{cases}$$

Notăm $n = 2^k$

$T(2^k, par) = 2T(2^{k-1}, impar) + 2^{2k}$

$T(2^{k-1}, impar) = 2T(2^{k-2}, par) + 1 \quad |\cdot 2$

$T(2^{k-2}, par) = 2T(2^{k-3}, impar) + 2^{2k-4} \quad |\cdot 2^2$

$T(2^{k-3}, impar) = 2T(2^{k-4}, par) + 1 \quad |\cdot 2^3$

$T(2^{k-4}, par) = 2T(2^{k-5}, impar) + 2^{2k-8} \quad |\cdot 2^4$

$\vdots$

$T(2^2, impar) = 2T(2) + 1 \quad |\cdot 2^{k-2}$

$T(2, par) = 2T(1) + 1 \quad |\cdot 2^{k-1}$

$T(1) = 1$

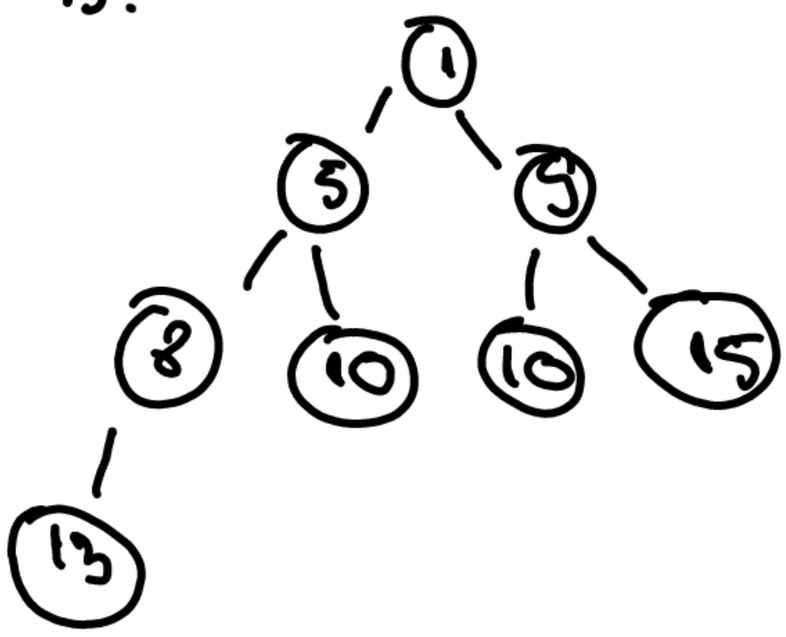
$T(n, par) = 2^{2k} + 2 + 2^{2k-2} + 2^3 + \quad |\cdot 2^k$

$2^{2k-4} + 2^5 + 2^{2k-6} + 2^7 + \dots + 2^{k-2} + 2^{k+1} =$

$= 2 \cdot \frac{2^{\frac{k}{2}} - 1}{2-1} + 2^{k+1} \cdot \frac{2^{\frac{k}{2}} - 1}{2-1} =$

$= (2^{\frac{k}{2}} - 1)(2 + 2^{k+1}) = (\sqrt{n} - 1)(2 + 2n) = O(n\sqrt{n})$

B.



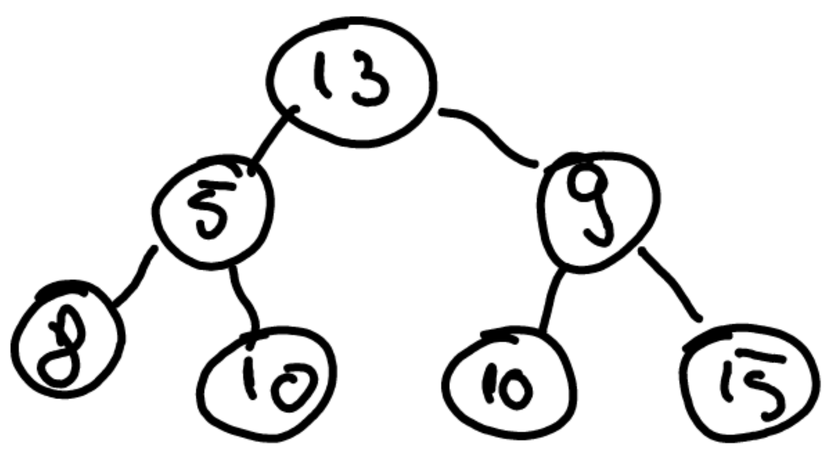
→ Ansamblul inițial

Acest ansamblu este un MIN HEAP (elementele mai mici au prioritate).

Ștergerea din ansamblu se face mereu de la nivelul rădăcinii:

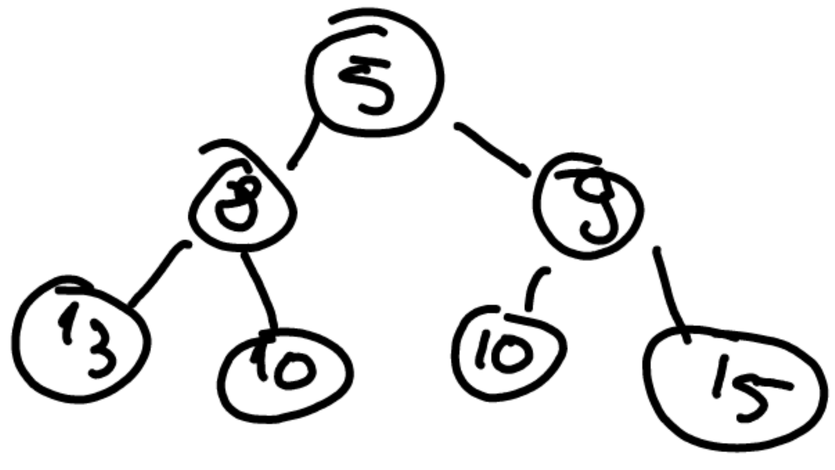
- Ștergem rădăcina 1
- o înlocuim cu ultimul elem. din ansamblu (13)
- coborâm pe 13 (pe cel mai prioritar descendent) până când cond. de ansamblu este din nou îndeplinită

$a_i < a_{2i}$ și $a_i < a_{2i+1}$



$5 < 13$ => coborâre

$8 < 13$ => coborâre =>



c1. d) data[10]

capacitatea maximă a stivei este de 42; chiar dacă are 10 elemente stocate, asta nu înseamnă că nu mai dispune de sp. liber.

De asemenea, noul element se adaugă în primul dacă acel capăt este considerat vârful stivei (ceea ce ar fi și mai eficient din punct de vedere timp -> adăugare + ștergere în $\Theta(1)$)

c2. Op. de adaugă (c) poate cauza depășire superioară, în cazul în care dorim adăugarea unui element într-o coadă plină.

Op. de ștergere poate cauza numai depășire superioară, când se dorește ștergerea dintr-o coadă vidă, iar op. vidă nu poate produce depășire.

3. Op. de adăugare în ABC -> reprez. înlănțuit, cu înlănțuire pe talpă:

Arbore

e : TElement[]

st : Întreg[]

dr : Întreg[]

r : Întreg (rădăcina arborelui)

next-free : Întreg (poz. primului element liber)

Subalgoritm adaugare (a, e) este

{pre: a ∈ Arbore
e ∈ TElement, e ≠ NULLTElement

post: a' ← a(+)e
}

Dacă next-free = -1 atunci
@ realocare spațiu
sfDacă

Dacă a.r = -1 atunci:
a.r ← a.next-free
a.next-free ← a.st[next-free]
a.e[r] ← e

{e devine noua rădăcină}

altfel

{trebuie să îl adăugăm pe e undeva în arbore}

poz ← a.r
poze ← -1
cât timp poz ≠ -1 ∧ a.e[poz] ≠ NULLTElement ∧ poze = -1 execută
Dacă e < a.e[poz] atunci:
Dacă a.st[poz] ≠ -1 atunci:
poze ← poz
poz ← a.st[poz]
{mergem și căutăm un loc în subarborele stâng}
altfel
poz ← a.dr[poz]
{mergem și căutăm un loc în subarborele drept}
sfDacă

{inserăm că am găsit loc}
Dacă poze = -1 atunci:
poze ← a.next-free
a.next-free ← a.st[next-free]
a.e[poze] ← e
altfel
a.st[poze] ← a.next-free
a.next-free ← a.st[next-free]
a.e[poze] ← e
sfDacă

altfel
{inserăm că am găsit loc}
poze ← a.next-free
a.next-free ← a.st[next-free]
a.e[poze] ← e
sfDacă

altfel
{inserăm că am găsit loc}
poze ← a.next-free
a.next-free ← a.st[next-free]
a.e[poze] ← e
sfDacă

altfel

Dacă a.dr[poz] ≠ -1 atunci
{trebuie continuată căutarea în subarborele drept}
poz ← a.dr[poz]
altfel
{am găsit loc + update pt. prima poz. liberă}
poze ← a.next free
a.next-free ← a.st[next-free]
a.e[poze] ← e
sf Dacă
sfCât Timp
sf Dacă
sfSubalgoritm

complexitate:
- de timp: O(n), considerând că realocarea se amortizează în O(1)
- de spațiu adițional: Θ(1)