A.

```
Subalgoritm rec(n) este
    Dacă n==0 atunci
        rec <- 0
    altfel
        suma<-0
        Pentru i=1,n execută
            Pentru j=1,n execută
                Suma <- Suma + j
            Sf Pentru
        Sf Pentru
        rec <- rec(n-1)
    Sf Dacă
Sf Subalgoritm
```

complexitatea acestui algoritm este $O(n^3)$.

$rec(0) = 1$

$rec(1) = 1^2 + rec(0)$

$rec(2) = 2^2 + 1^2 + 1$

$rec(3) = 3^2 + 2^2 + 1^2 + 1$

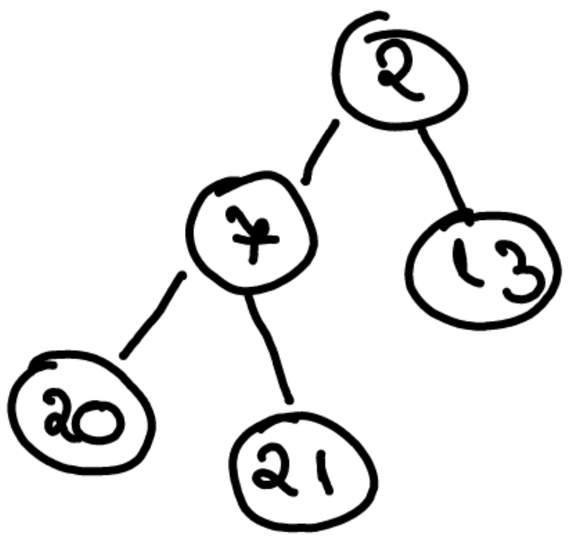
$\vdots$

$rec(n) = n^2 + (n-1)^2 + \dots + 2^2 + 1 + 1$

$= \frac{n(n+1)(2n+1)}{6} \simeq O(n^3)$

B. Un vector în care elementele se succed în ordine crescătoare poate fi considerat un ansamblu, relația R fiind $\leq$. Astfel, vom avea un MIN HEAP: primul element este cel mai mic, iar fiecare element de după acesta este mai mic decât fiii și descendenții săi.

Spre exemplu, pt. vectorul: 2, 7, 13, 20, 21 avem:



care este un ansamblu (MIN HEAP)

Putem mai riguros: $a_i \leq a_{2i} \wedge a_i \leq a_{2i+1}$

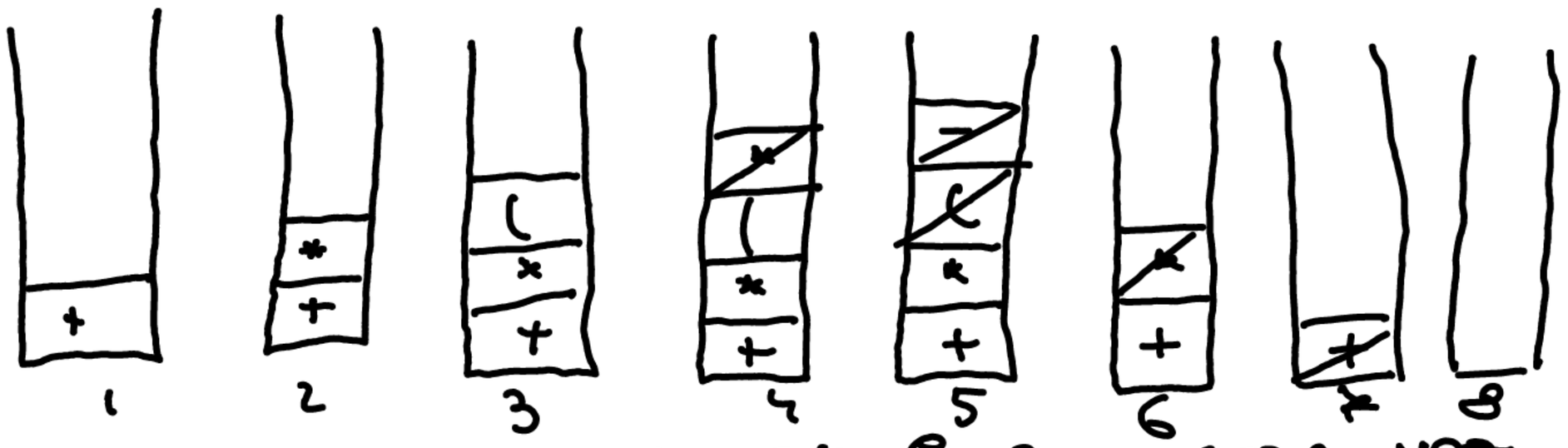
C1. e) niciuna dintre op. menționate

Pentru că, în cazul implementării unei stive, vom ține minte și ultimul element (vf. stivei) => ștergerea, adăugarea și accesarea vor avea timp constant, iar op. vidă se poate face și ea de vf. stivei (în cazul în care e Nil -> true) dar se poate pierde evidențierea dimensiunii; în orice modalitate, timpul ar fi tot constant.

C2.

$4 + 3 * (6 * 3 - 12)$

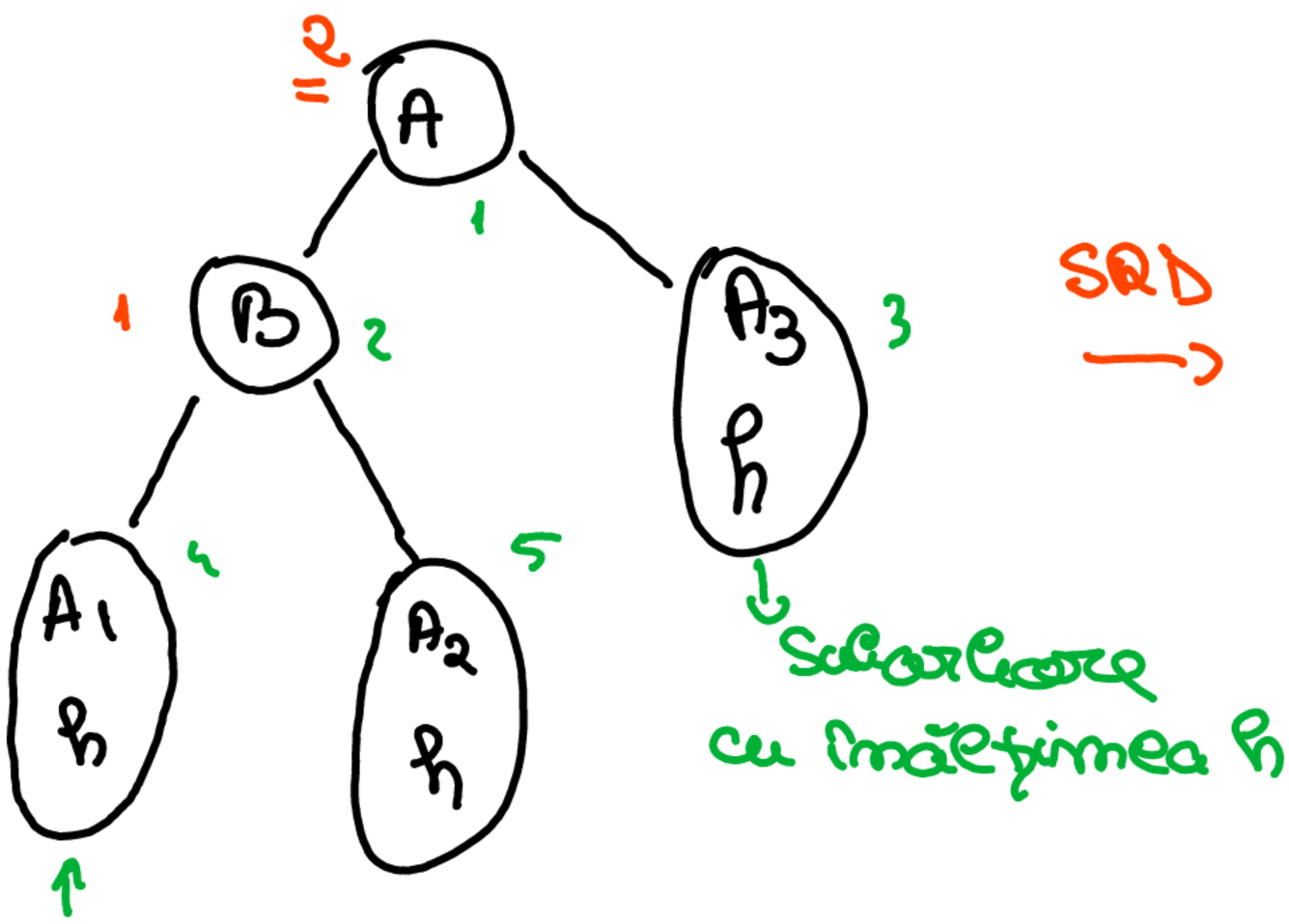
Evoluția stivei:

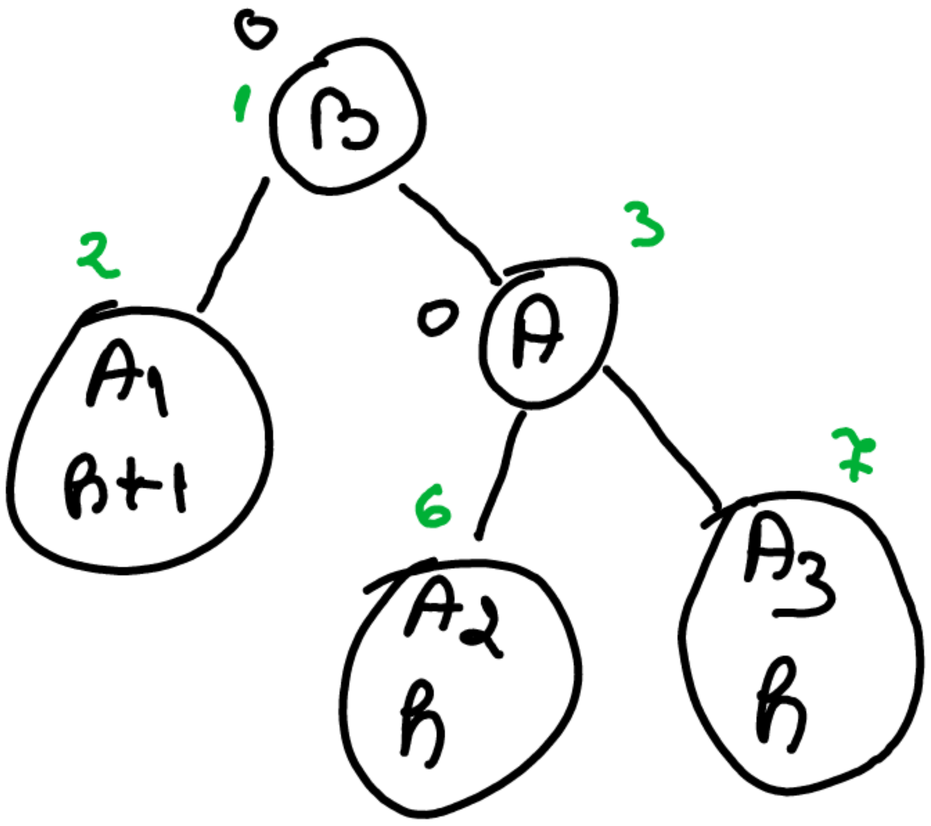


=> nr. maxim de simboluri care vor apărea în stivă la un moment dat este: 4 (în cazurile 4 și 5)

D.

Operația de rotație dreapta este utilizată pentru a rechilibra / echilibra un arbore a cărui rădăcină are factorul de echilibrare 2. Totuși, este necesară singura rotație la dreapta în următorul caz:





Astfel, singura rotație spre dreapta va modifica rădăcina arborelui pointer - o „rotire la dreapta" => în loc de A, arborele va avea rădăcina B.

Totuși, pentru a-și păstra proprietățile de ABC, mai sunt câteva schimbări de făcut:

- cum B era inițial în st. lui A => A > B => în noul arbore A va fi în dreapta lui B
- dar și A_2 este în dreapta lui B
  dar, dar, inițial, B fiind în st. lui A => A_2 este în st. lui A

=> B.dr → A (se schimbă) (B - indice 1, A - indice 3)
B.st → A_1 (rămâne A_1) (B - indice 1, A_1 - indice 2)
A.st → A_2 (se schimbă) (A - indice 3, A_2 - indice 6)
A.dr → A_3 (rămâne A_3) (A - indice 3, A_3 - indice 7)

Subalgoritm SRD (a) este
{pre: a este un vector de TElemente
post: a va fi echilibrat
}
Dacă dim(a) > 2 atunci
A ← a[1]
B ← a[2]
{ se copiază A_1, A_2, A_3 în vectori separați }

creează (v) } s-a creat vectorul v {

v[1] ← B

v[3] ← A

copiere (a, 4, 2, v)
copiere (a, 5, 6, v)
copiere (a, 3, 7, v)

} se copiază cam toate elementele

pentru i=1, n execută

a2[i] ← v2[i]

SfPentru

Sf Subalgoritm

Complexitate: - timp : $\Theta(n)$

- spațiu : $\Theta(n)$ (adițional)

unde n reprezintă nr. de elemente din vectori

Functia copiere (a, poz1, poz2, v) este

{ pre: a -> oricare

poz1, poz2 -> nr. intregi, reprez. pozitiile initiale si cea finala

post: v -> vector de TElemente
elem. din subs. a caror indici apartin

se afla ca poz1 este mutat ca poz2

}

v[poz2] ← a[poz1]

creeaza(c) {coada}

adauga(c, {poz1, poz2})

CatTimp {vida(c) executa

{parcurgere in latime a subarborelui}

sterge(c, {inainte, dupa})

Daca inainte*2 ≤ dim(a) atunci

v[dupa*2] ← a[inainte*2]

adauga(c, {inainte*2, dupa*2})

SF Daca

Daca inainte*2 + 1 ≤ dim(a) atunci

v[dupa*2+1] ← a[inainte*2+1]

adauga(c, {inainte*2+1, dupa*2+1})

SF Daca

{elem. s-au pozitionat corect in vectorul}

SF CatTimp

SF Functia

entete

O(n) : n dimensiunea subarborelui