SDA 103

A. subalgoritm alg(n, i, j) i=j=0 apel extern
  Dacă n = 1 atunci
    alg = 1
  altfel
    Dacă i = n atunci
      alg ← 1 + alg(n-1, 0, 0)
    altfel
      Dacă j = n atunci
        alg ← 1 + alg(n, i+1, 0)
      altfel
        alg ← 1 + alg(n, i, j+1)
      sfârșitdacă
    sfârșitdacă
  sfârșitdacă
sfârșit subalgoritm

$$f(n,i,j) = \begin{cases} 1, & n=1 \\ 1+T(n-1,0,0), & i=n \\ 1+T(n-1,i+1,0) & j=n \\ 1+T(n-1,i,j+1) & \text{altfel} \end{cases}$$

$T(n) = n^2 + T(n-1)$

⇓ recursiv

$T(n) = 1^2 + 2^2 + \dots + n^2 \in \Theta(n^3)$

$\uparrow$

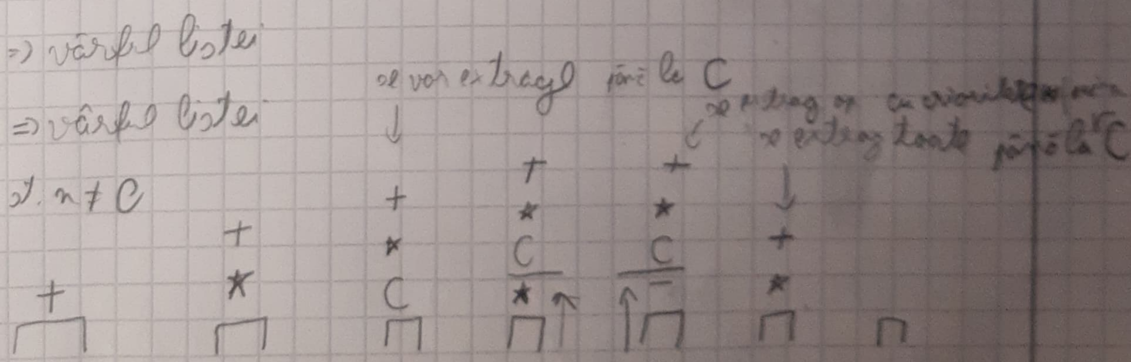
$\frac{n(n+1)(2n+6)}{6}$ (IDK)

B. Da, deoarece se satisface prop. de ansamblu $\forall i \in \{1, \dots, [n/2]\}$ (are vecin stâng și/sau drept)

$a_i \leq a_{2i} \wedge a_i \leq a_{2i+1}$, deoarece $\forall i, j \in \overline{1,n}$ $a_i < a_j \Leftrightarrow i < j$ (ord. crescător)

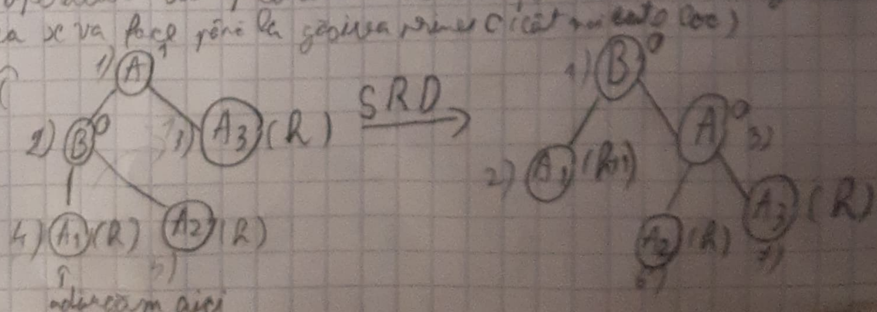
C. (c) accesare ⇒ vârful listei
  ștergere ⇒ vârful listei
  adăugare ⇒ vârful listei
  vidă ⇔ vf. = 0

C2. Stiva:



(d) "C" e un operator cu prioritate minimă, dar care nu extrage elemente (→ extragerea se va face până la găsirea primei C (cât mai este loc))

D. Situație:



Container:

v:]Element[], elementele

n: Întreg, dimensiunea

max: Întreg capacitatea maximă

1/8

subalgoritm(ab)

Pre: ab: un arbore binar de căutare

Post: ab: noul arbore binar de căutare, rotit la stânga de rădăcină

pentru i←1, n+1 execută
 v[i] ← NIL {inițializăm toate el. cu NIL} {v→noul ansamblu}
sfârșit pentru
creează(c) {coadă cu indici}
adaugă(c,4) {A_1}; m←ab.n
cât timp ¬vidă(c) execută
 șterge(c, i)
 v[i-2] ← ab.v[i] {toate el. din noul ansamblu ∈ A_1, vor avea indicii ...}
 Dacă 2·i ≤ ab.n ∧ ab.v[2·i] ≠ NIL atunci
  adaugă(c, 2·i)
 sfârșit dacă
 Dacă 2·i+1 ≤ ab.n ∧ ab.v[2·i+1] ≠ NIL atunci
  adaugă(c, 2·i+1)
 sfârșit dacă
sfârșit cât timp
adaugă(c, 5) {A_2}
cât timp ¬vidă(c) execută
 șterge(c, i); m ← max(m, i+1)
 v[i+1] ← ab.v[i] {indicii cresc cu o unitate}
 Dacă 2·i ≤ ab.n ∧ ab.v[2·i] ≠ NIL atunci
  adaugă(c, 2·i)
 sfârșit dacă
 Dacă 2·i+1 ≤ ab.n ∧ ab.v[2·i+1] ≠ NIL atunci
  adaugă(c, 2·i+1)
 sfârșit dacă

```
sfârșit cât timp
adaugă(C, 3)
cât timp ¬vidă(C) execută
    șterge(C, i); n ← max(m, i+4)
    v[i+4] ← ab.v[i] { indici crescuți cu 4 unități }
    Dacă 2·i ≤ ab.n ∧ a.v[2·i] ≠ NIL atunci
        adaugă(C, 2·i)
    sfârșit dacă
    Dacă 2·i+1 ≤ ab.n ∧ a.v[2·i+1] ≠ NIL atunci
        adaugă(C, 2·i+1)
    sfârșit dacă
sfârșit cât timp
Dacă m > ab.max atunci
    redim(ab) { ar putea să nu încapă unele elemente }
sfârșit dacă
Pentru i ← 1, m execută
    ab.v[i] ← v[i]
sfârșit pentru
sfârșit subalgoritm
```