A.

Cost favorabil = Cost mediu = Cost defavorabil =>

timpul mediu = timpul defav.

Notăm $n = 2^k$

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

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

$\vdots$

$T(1) = 1$

$T(2^k) = \underbrace{1 + 1 + \dots + 1}_{k \text{ ori}}$

$T(n) = \log_2 n \Rightarrow \Theta(\log_2 n)$

B. ștergere din AVL care necesită SRD

- se șterge nodul 10.5 => factorul de echilibrare al rădăcinii va ajunge la 2 => este necesară o rotire spre dreapta; cu exactitate, a SRD (subarborele stâng a lui 7 are o înălțime mai mare față de subarborele drept)

C1.

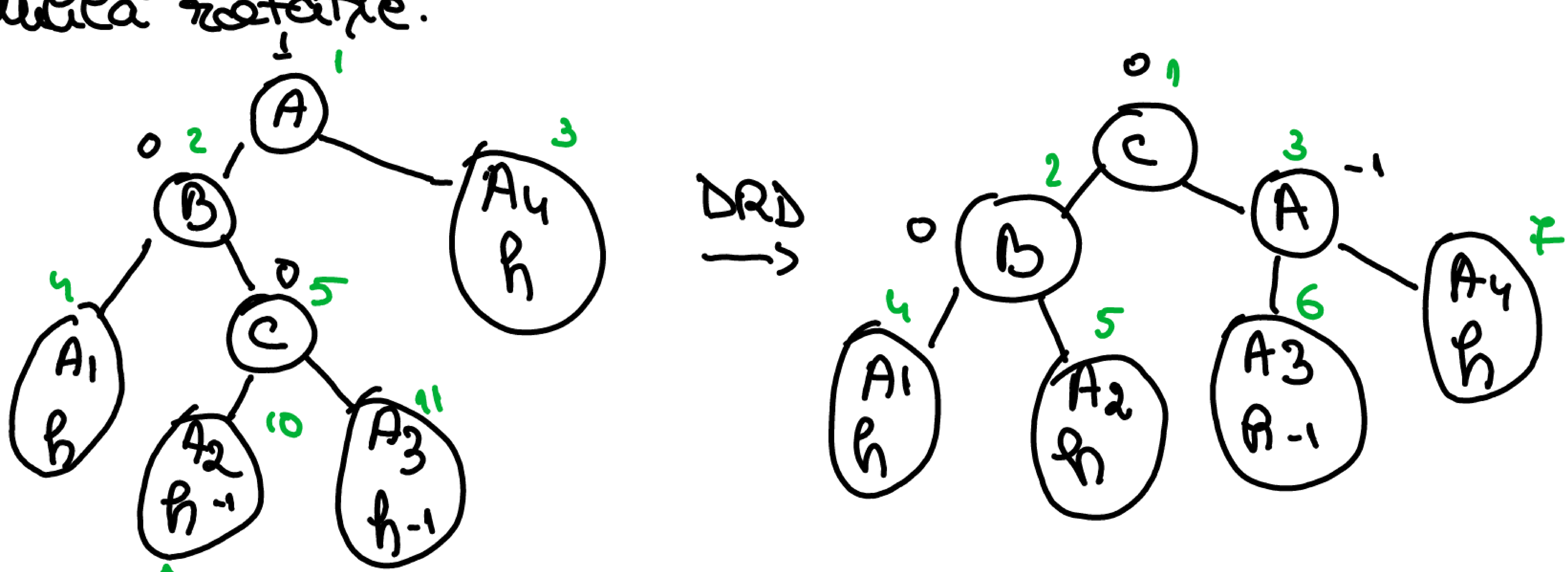
Pentru sortarea vectorului se pot folosi: Merge Sort, precum și HeapSort deoarece ele funcționează, în același mod, atât pe nr. întregi, cât și reale.

↑ ar fi mers și radix sort (ex. la seminar pe nr. reale)

C2. Valoarea cea mai mare într-un ansamblu construit cu relația „≥" va fi primul element din vector, acesta trebuind să îndeplinească rel. de „≥" cu toți descendenții săi. Mai mult, un astfel de ansamblu este un max heap, iar răspunsul corect este a) data[0].

D. DRD

O rotație spre dreapta se utilizează atunci când factorul de echilibrare al rădăcinii arborelui este 2. Totuși, există cazuri când o singură rotație spre dreapta nu este suficientă => vom implementa o dublă rotație.



↑ se adaugă un element aici

- adăugarea unui nou element în A_2 va determina creșterea înălțimii la h => factorul de echilibru al lui A va avea valoarea 2

- în acest caz, o SRD nu ar rezolva problema => vom apela la DRD

Pașii unei DRD:

- C va deveni rădăcină
- A va deveni descendentul drept al lui C (poz: 3)
- B va deveni descendentul stâng al lui C (poz: 2)
- A_1 și A_4 își păstrează părinții, dar își modifică pozițiile: A_1 - poz: 4, A_4 - poz: 7
- A_2 va deveni descendentul drept al lui B (poz: 5)
- A_3 va deveni descendentul stâng al lui A (poz: 6)

vector
  e: TElement []
  n: întreg (dimensiunea vectorului)

Functia copiere (a, v, poz1, poz2) este
  { pre: a ∈ Vector ; v ∈ Vector
      poz1 ∈ întreg, poz2 ∈ întreg
   post: v va conține o parte din elementele
      din a, așezate pe poz. corespunzătoare
  }

  v.e[poz2] ← a.e[poz1]

creează (c) {se creează o coadă}

adauga (c, {poz1, poz2}) {se stochează cele 2 {poziții {before, after} }

câtTimp ¬ vida (c) execută

sterge (c, {begin, end})
{se accesează indicii}

Dacă begin * 2 ≤ a.n atunci ← poate ar fi util să a[begin*2] ≠ NIL fie

v.e[end*2] ← a.e[begin*2]
adauga (c, {begin*2, end*2})
SfDacă

Dacă begin*2+1 ≤ a.n atunci

v.e[end*2+1] ← a.e[begin*2+1]
adauga (c, {begin*2+1, end*2+1})
SfDacă
SfCâtTimp
SfFunctia

Subalgoritm DRD (a) este
{pre: a ∈ Vector și este dezechilibrat (care nevoie de DRD)
post: a este echilibrat
}
← și inițializarea vectorului v cu NIL
Dacă a.n ≥ 5 atunci

v.e[1] ← a.e[5]
v.e[2] ← a.e[2]
v.e[3] ← a.e[1]

{se mută A1}

copiere (a, v, 4, 4)

{se mută A2}

copiere (a, v, 10, 5)

{se mută A3}

copiere (a, v, 11, 6)

{se mută A4}

copiere (a, v, 3, 7)

pentru i = 1, ~~a~~ n execută (nr. random n)

a.e[i] ← v.e[i]

sf pentru

sf dacă

sf subalgoritm

complexitate :
- de timp : $\Theta(n)$ (se mută toate elementele)
- de spațiu : $\Theta(n)$ (adițional)