A.

```
Subalgoritm ceva (n)}
   {pre : n e Intreg
    post : se returneaza 0
   }
    i<-n
    CâtTimp i>1    executa
    j<-n
    cât timp j>1 executa
          j<- [n/2]
    SRcâtTimp
     i<- [n/2]
SRcâtTimp

ceva <-0
SRSubalgoritm
```

$$T(n) = \sum_{i=1}^{\log_2 n} \sum_{j=1}^{\log_2 n} 1 = \sum_{i=1}^{\log_2 n} \log_2 n = \log_2 n \cdot \log_2 n = \log_2^2(n)$$

Sau

$$\left.\begin{aligned} &T(2^k) = T(2^{k-1}) + 1 \\ &T(2^{k-1}) = T(2^{k-2}) + 1 \\ &\vdots \\ &T(1) = 1 \end{aligned}\right\} \quad T(n) = \sum_{i=1}^{\log_2} \text{---} \cdots \text{---}$$

$$\oplus \quad T(2^k) = \underbrace{1+1+\cdots+1}_{k \text{ ori}} = k = \log_2 n$$

B. $m = 11$

$d_1(c) = c \bmod m = c \bmod 11$

$d_2(c) = 1 + (c \bmod (m-1)) = 1 + (c \bmod 10)$

$d(c,i) = (d_1(c) + i * d_2(c)) \bmod 11$

0	1	2	3	4	5	6	7	8	9	10
c: 22	-1	17	15	4	-1	29	59	88	31	10

se adaugă 10 : $(10 + 0) \% 11 = 10$

Se adaugă 22 : $(0 + 0) \% 11 = 0$

Se adaugă 31 : $(9 + 0) \% 11 = 9$

Se adaugă 4 : $(4 + 0) \% 11 = 4$

se adaugă 15 : $(4 + 0) \% 11 = 4$ x

$(4 + 1 \cdot 5) \% 11 = 9$ x

$(4 + 2 \cdot 5) \% 11 = 14 \% 11 = 3$

se adaugă 28 : $(6 + 0) \% 11 = 6$

se adaugă 17 : $(6 + 0) \% 11 = 6$ x

$(6 + 1 \cdot 7) \% 11 = 13 \% 11 = 2$

se adaugă 88 : $(0 + 0) \% 11 = 0$ x

$(0 + 1 \cdot 8) \% 11 = 8$

se adaugă 59 : $(4 + 0) \% 11 = 0$ x

$(4 + 1 \cdot 9) \% 11 = 13 \% 11 = 2$ x

$(4 + 2 \cdot 8) \% 11 = 22 \% 11 = 0$ x

$(4 + 3 \cdot 9) \% 11 = (4 + 27) \% 11 = 31 \% 11 = 9$ x

$(4 + 4 \cdot 3) \% 11 = (4 \cdot 10) \% 11 = 40 \% 11 = 7$

C1.

a) verificări prop. de ansamblu + are structura de ansamblu

-descriere:

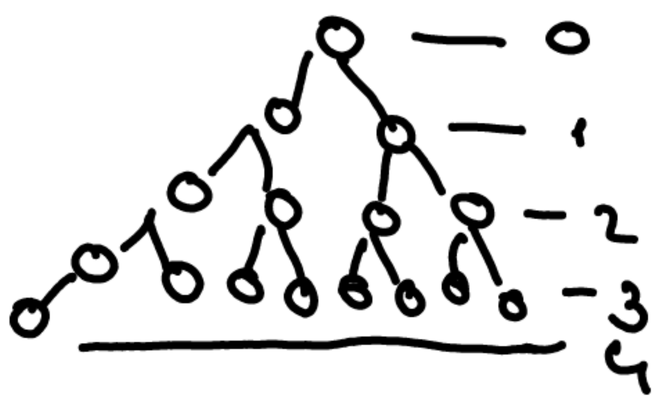
- are primele 2 niveluri pline, iar ultimul este completat de la stânga la dreapta
- are relația : $\leq$ = (MIN HEAP)

e) este ansamblu ↑

C2. nr. minim de noduri într-un arbore binar aproape plin, de adâncime 4 (5 niv)

primele 4 niv. sunt pline:

- 0 -> 1 nod
- 1 -> 2 noduri
- 2 -> 4 noduri
- 3 -> 8 noduri



ultimul nivel conține un singur nod (se cere nr. min)

$1 + 2 + 4 + 8 + 1 = 10 + 6 = 16$ -c)

D.

Nod
e: TElement
st: ↑Nod
dr: ↑Nod

Arbore:
r: ↑Nod (adresa rădăcinii arborelui)

SubalgoritmDRD (a) este
{pre: a ∈ Arbore, a -> neechilibrat + conține cel puțin 3 noduri
post: a' este un arbore echilibrat
}

A ← a.r
B ← [a.r].st
C ← [[a.r].st].dr
[A].st ← [C].dr
[B].dr ← [C].st
[C].st ← B
[C].dr ← A

{ s-ar recalcula inaltimile pt toate nodurile }

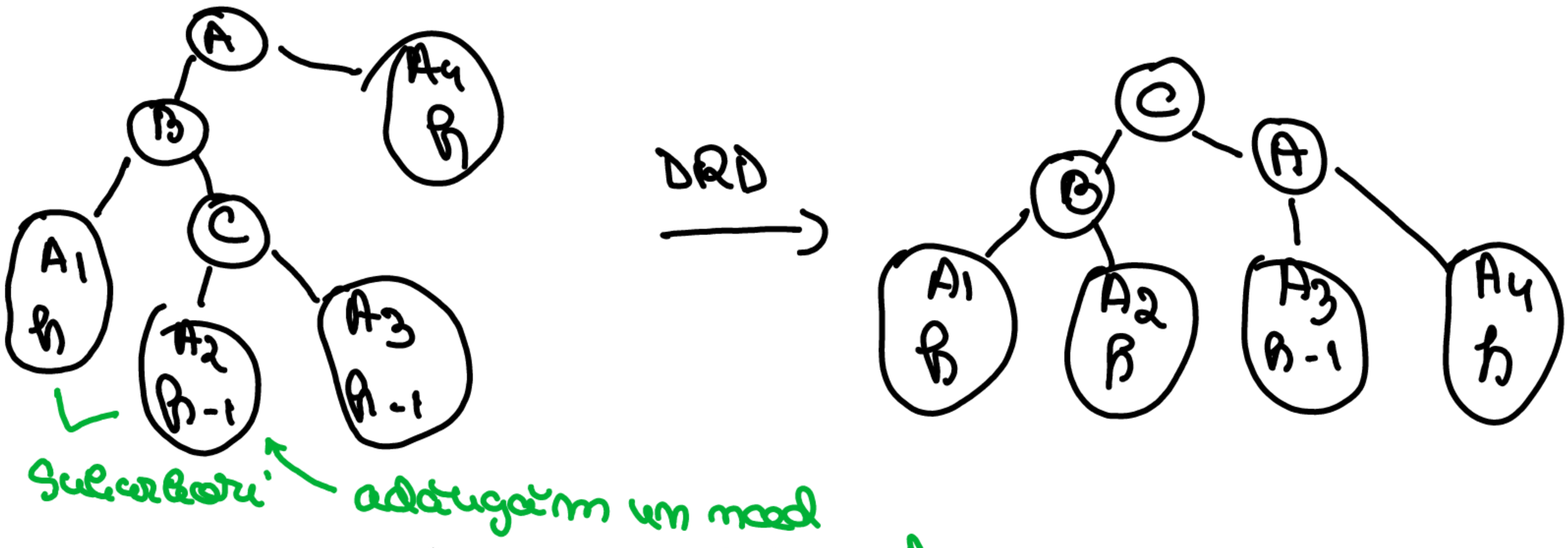
DRD ← a
SRSubalgoritm

Complexitate:
- de timp: Θ(1)
- de sp. aditional: Θ(1)

Ideea:
- se retine in fiecare nod inaltimea
- nodurile ce isi vor schimba inaltimea sunt: A, B, C

B.h = max(A1.h, A2.h) + 1
A.h = max(A3.h, A4.h) + 1
C.h = max(B.h, A.h) + 1

Situație în care avem nevoie de DRD:



Subarbori' ← adăugăm un nod

Factorii de echilibrare: aici => A2 va avea înălțimea h

INIȚIAL:	DUPĂ ADĂUGARE:	DUPĂ ROTIRE:
C: 0	C: 1	C: 0 => echilibrat
B: 0	B: -1	B: 0
A: 1	A: 2 => dezechilibrat	A: -1

Cum factorul de echilibrare al lui A este 2 => este necesară o rot. spre dreapta (avem o înălțime prea mare în subarborele stâng); totuși, nu va fi suficientă o singură rot. => dublă rotație

Schimbări:

C -> devine rădăcină

descendentul st. al lui C devine B (B<C)

descendentul dr. al lui C devine A (C<A)

descendentul dr. al lui B devine A_2

descendentul st. al lui A devine A_3