A. timpul mediu și cel defavorabile

Subalgoritmul nu are cazuri favorabile/defavorabile =>

=> timpul mediu = timpul defavorabile =

timpul favorabile

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + 1, & n \neq 1 \end{cases}$$

Notăm $n = 2^k \Rightarrow k = \log_2 n$

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

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

$\vdots$

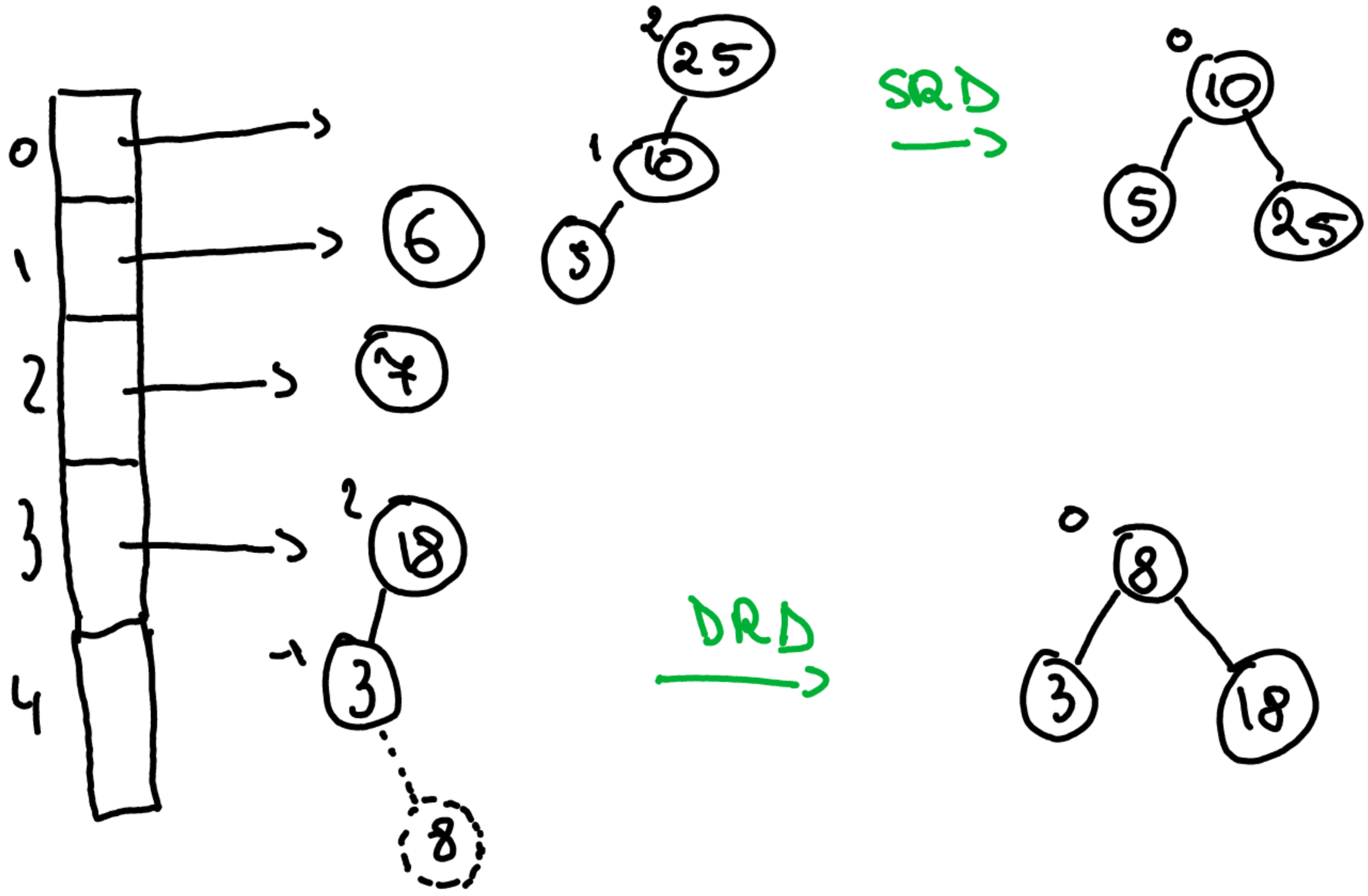
$T(1) = 1 \quad |\cdot 2^k$

$+) \; T(2^k) = 1 + 2 + \dots + 2^k = 1 \cdot \frac{2^{k+1} - 1}{2-1} = 2^{k+1} - 1 \Rightarrow T(n) \in \Theta(n)$

$\Theta(2^{\log_2 n}) = \Theta(n)$

B. $d(c) = c \bmod 5$

coliziunile sunt ref. prin înlănțuire, realizând arbori AVL pt. memorarea coliziunilor



A fost nevoie de op. de: dublă rot. dreapta și simplă rot. dreapta pt. a echilibra cei doi arbori.

C1. Timpul defavorabil pt. HeapSort este $O(n \log_2 n)$, deoarece putem aplica operația de ștergere de n ori, ea având timpul defav. $O(\log_2 n)$, pentru a ne reface elementele sortate.

+ se putea menționa, dacă alg. nu este în place, și că, în cazul construcției unui arbore -> O(n)

C2. a) Atât adăugarea, cât și ștergerea se fac în timp liniar

- pentru că, adăugând un nr. nou, el va deveni vârful stivei => toate elem. din stivă trebuie mutate cu o poz. spre dreapta pt. a elibera poz. 1
- în cazul ștergerii, toate elementele vor trebui mutate cu o poziție spre stânga, pentru a avea în continuare, vârful pe poz. 0

b.

unde nu se menționa, putem să ținem direct nivelul

```
Nod:
    e: TElement
    st: ↑Nod
    dr: ↑Nod

Arbore:
    r: ↑Nod (rădăcina arborelui)

Coada:
    c: ↑Nod[]  → vect. de pointeri nod
    n: întreg[]
```

Subalgoritm parcurgere-BFS (a, e, e') este

```
{pre: a ∈ Arbore, e ∈ TElement, e' ∈ TElement
post: se returnează 'da' sau 'nu'
    'nu' -> dacă cele 2 elemente nu se află pe același nivel
    'da' -> în caz contrar
}
```

creeaza (c) { se creeaza o coada, c ∈ Coada }

nivel_dat ← -1 ; raspuns ← "nu"

Daca a.r ≠ NIL atunci:

adauga (c.e, a.r)

SR Daca adauga (c.m, 0)

cât Timp ? vida (c.e) executa

{ sterge (c.e, p) ; sterge (c.m, pr)

Daca [p].e = e atunci:

Daca nivel_dat = -1 atunci:

nivel_dat ← pr

altfel

Daca nivel_dat = pn atunci:

raspuns ← "da"

SF Daca

@ STOP

SR Daca

SF Daca }

{ Daca [p].e = e' atunci:

Daca nivel_dat = -1 atunci:

nivel_dat ← pr

altfel

Daca nivel_dat = pn atunci:

raspuns ← "da"

SF Daca

@ STOP

SR Daca

SF Daca }

Daca [p].st ≠ NIL atunci:

adauga (c.e, [p].st)

adauga (c.m, pn+1)

SF Daca

Dacă [p].dr ≠ NiL atunci

adauga (c.e, [p].dr)

adauga (c.n, p.n+1)

SfDacă

SfCâtTimp

parcurge- BFS <- răspuns

SfSubalgoritm

complexitate : O(n) -> liniară (timp)

complexitate spațiu adițional O(n) -> pt. coadă