A.

```
Subalgoritm rec(n) este
    { pre: n ∈ întreg
      post: se returnează valoarea 0
    }
    Dacă n <= 1 atunci
        rec <- 0
    altfel
        j <- [n/2]
        rec(j)
    sfDacă
sfSubalgoritm
```

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

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

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

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

$\vdots$

$T(1) = 1$

$(+)$ $T(2^k) = \underbrace{1+1+1+\dots+1}_{\text{de } k \text{ ori}} \Rightarrow T(2^k) = k \Rightarrow T(n) = \log_2 n \in$

$\Theta(\log_2 n)$

$\downarrow$

caz fav = caz mediu = caz defav.

B.

d(c) = nr. zecimal coresp. biților $b_3b_2b_1b_0$ ai cheii c
Coliziunile sunt rezolvate prin liste întrepătrunse
Gestiunea sp. liber : stânga -> dreapta

i:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
c:	19	34	18	3	20	18	-1	23	8	-1	-1	11	-1	-1	-1	-1
urm:	-1	5	1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

primLiber : $\cancel{0}\ \cancel{1}\ \cancel{4}\ 5$

$23 = 10111_{(2)} \Rightarrow d(23) = 0111_{(2)} = 7$
$11 < 15 \Rightarrow d(11) = 11$
$8 < 15 \Rightarrow d(8) = 8$
$18 = 10010_{(2)} \Rightarrow d(18) = 0010_{(2)} = 2$
$3 < 15 \Rightarrow d(3) = 3$
$19 = 10011_{(2)} \Rightarrow d(19) = 0011_{(2)} = 3$ (ocupat)
=> se ocupă prima poz liberă + se leagă de poz. 3
$34 = 100010_{(2)} \Rightarrow d(34) = 0010_{(2)} = 2$ (ocupat)
=> se ocupă prima poz. liberă + se leagă de poz. 2
$20 = 10100_{(2)} \Rightarrow d(20) = 0100_{(2)} = 4$
$18 = 10010_{(2)} \Rightarrow d(18) = 0010_{(2)} = 2$ (ocupat)
=> se ocupă prima poz. liberă + se leagă de poz 1

C1. a)

Adăugarea se va realiza la final (într-o reprez. care nu ține cont de frecvența elementelor) => $\Theta(1)$ => și în caz defav. vom avea un timp constant de execuție

Totuşi, în orice reprezentare a colecţiei, operaţiile de ştergere şi de numărAparinţii conţin operaţie de căutare, care în caz defavorabil are complexitatea liniară $\Theta(n)$.

C2. e)

Atâta timp cât se ia în considerare opţiunea de rehashing, se pot stoca oricâte elemente. Dacă această opţiune nu este permisă, atunci s-ar putea stoca maxim 512 intrări.

D.

CP3

```
e: TElement[ ]
n: Intreg (nr. de elemente)
r: TRelaţie
```

```
Subalgoritm şterge (c) este
  { pre : c ∈ CP3
    post: c' = c - e, unde e este al 3-lea cel mai
          prioritar. e ∈ TElement
          @ excepţie dacă c nu are 3 elemente
  }
```

```
    Dacă c.n < 3 atunci
        @ excepție "Nu avem 3 elem. în coadă"
    altfel
        a1 <- ștergere (c)
        a2 <- ștergere (c)
        ștergere(c)
        { se șterg primele 3 elem. prioritare }
        { urmează ca primele 2 cele mai prioritare,
        stocate în a1 și a2 să fie readăugate }
        adaugă (c, a1)
        adaugă (c, a2)
    SfDacă
SfSubalgoritm
```

```
Subalgoritm ștergere (c) este        trebuie introduse
                                     pre și post condiții
    a <- c.e[1]
    c.e[1] <- c.e[c.n]
    c.n <- c.n - 1
    { se coboară elementul dacă nu respectă relația }
    poz <- 1
    pozd <- 2
    Dacă c.r (c.e[3], c.e[poz]) atunci
        pozd <- 3
    SfDacă
```

```
        CâtTimp c.r(c.e[pozd],c.e[poz]) ∧ pozd ≤ c.n ex:
            aux ← c.e[pozd]
            c.e[pozd] ← c.e[poz]
            c.e[poz] ← aux
            poz ← pozd
            pozd ← pozd*2
            Dacă c.r(c.e[pozd+1],c.e[pozd]) ∧
                 pozd+1 ≤ c.n ∧ pozd ≤ c.n atunci
                 pozd ← pozd+1
            sfDacă
        sfCâtTimp

        ștergere ← a
sfSubalgoritm
```

+ câteva comentarii pt. ușurarea înțelegerii

```
Subalgoritm adaugă(c,e) este        + pre & post condiții
    c.n ← c.n+1
    c.e[c.n] ← e
    {acum ne pregătim de înălțare}
    {vom înălța doar dacă părintele nu se află în rel.cu}
    poz ← c.n
    pozp ← [poz/2]
    CâtTimp pozp ≥ 1 ∧ c.r(c.e[poz],c.e[pozp]) ex
        aux ← c.e[poz]
        c.e[poz] ← c.e[pozp]
        c.e[pozp] ← aux

        poz ← pozp
        pozp ← [poz/2]
    sfCâtTimp
sfSubalgoritm
```

Complexitate:

- timp: $O(h)$, unde h este înălțimea ansamblului
  $h = \log_2 n$
- spațiu adițional: $\Theta(1)$