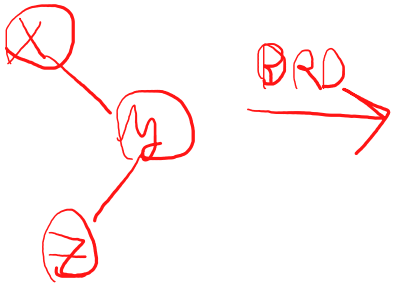
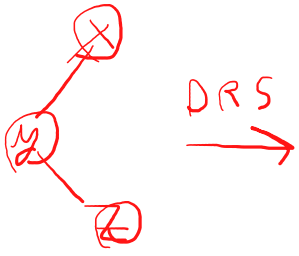
A. Deduceți timpii mediu si defavorabil pentru următorul subalgoritm. Justificați rezultatul.

```
Funcția F(n, i) este {:Intreg}
|       {pre: n, i:Intreg}
|       dacă n=1 atunci
|       |       F←1
|       |  altfel
|       |       pentru j←1,n executa i←i+1 sfpentru
|       |       m ← n div 2
|       |       dacă i mod 2 =0 atunci
|       |       |       F← F(m, i) - i
|       |       |  altfel
|       |       |       F← F(m, i) + i
|       |       Sfdacă
|       Sfdacă
SfF
```

$$T(n) = \begin{cases} 1, & n = 1 \\ n + T(\frac{n}{2}), & \text{altfel} \end{cases}$$

B. Ilustrați pe un exemplu concret operația de simplă rotație spre dreapta într-un arbore AVL.Justificati





C. O TD cu coliziuni rezolvate prin adresare deschisă şi verificare pătratica are 1024 locaşii. Care este numărul maxim de intrări care pot fi plasate în tabelă? Justificati

a) 256 b) 1023 c) 512 d) 1024 e) oricât

- Locaţia *i* nu este liberă ⇒ avem coliziune
  - dacă *primLiber* = *m* (tabela este plină) ⇒ redimensionare: mărim *m*, ceea ce presupune redispersarea elementelor (*rehashing*)
  - memorăm cheia *c* la *primLiber*
  - ultimul nod din lista înlănţuită care începe de la locaţia *i* este legat de *primLiber*
  - se actualizează primul liber

C. Un vector $x_1, \ldots, x_n$ de numere întregi cuprinse în intervalul [10, 2000] pot fi sortate crescător folosind BucketSort în:

a) O(n) b) θ(n) c) θ(n^2)

Justificati

=> Complexitatea celui de-al doilea pas al algoritmului este Θ(N+n) <=> Θ(max{N,n}) *(a se observa similaritatea cu problema 4 din seminarul 2)*

=> Complexitatea algoritmului BucketSort este Θ(N+n) <=> Θ(max{N,n})

Dacă N ϵ O(n) => Complexitatea algoritmului BucketSort este Θ(n), deci **liniară**.

BucketSort are complexitate **θ(n)** deoarece, pentru un vector de numere întregi bine distribuite într-un interval cunoscut ([10, 2000]), elementele pot fi plasate uniform în bucket-uri și sortate eficient, ducând la timp de execuție liniar.