Model Examen – Seria 14
Rezolvari

1. Exprimarea în notația Θ

- $\log(\sqrt{n}) = \frac{1}{2}\log(n)$. Scapam de constanta si obtinem raspunsul: $\Theta(\log n)$.
- $(n + 2^{200})^{500} = \Theta(n^{500})$, deoarece cel mai mare termen din paranteza este n^{500}.

2. Intersecție $o(f(n)) \cap \omega(f(n))$

Deoarece $o(f(n))$ reprezinta functiile care cresc strict mai lent decat f(n) si $\omega(f(n))$ reprezinta functiile care cresc strict mai rapid, intersectia este nula: $o(f(n)) \cap \omega(f(n)) = \emptyset$.

3. Cate muchii are un arbore binar complet cu n noduri?

Fiind un arbore, implicit are **n-1** muchii.

4. Operatii pe un min-heap

Inseram valorile **40, 22, 2, 18, 19, 5, 3**, iar apoi scoatem radacina.

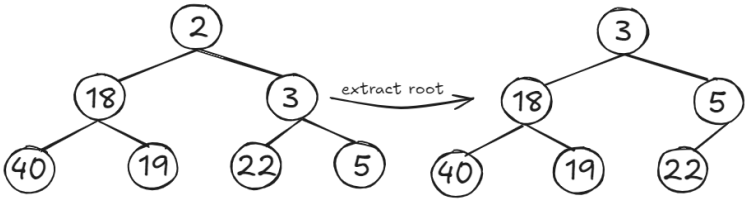

Figure 1: Min-Heap

5. Operatii pe un BST

Inseram valorile **15, 22, 2, 18, 19, 40, 30, 16, 50** si stergem valoarea **22**.

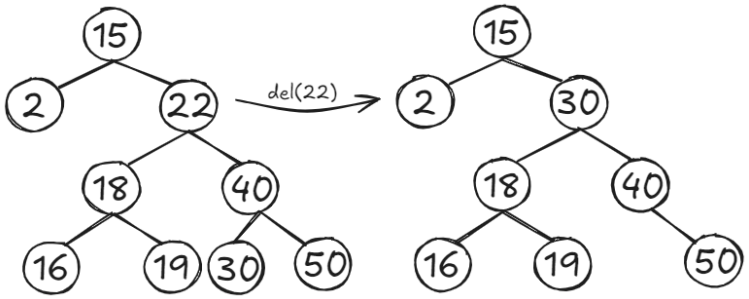

Figure 2: Binary Search Tree

6. Arbore Huffman

Cerinta: desenati arborele Huffman optim pentru $a : 32, c : 1, d : 15, e : 10, k : 22, o : 2, r : 5, t : 13$.

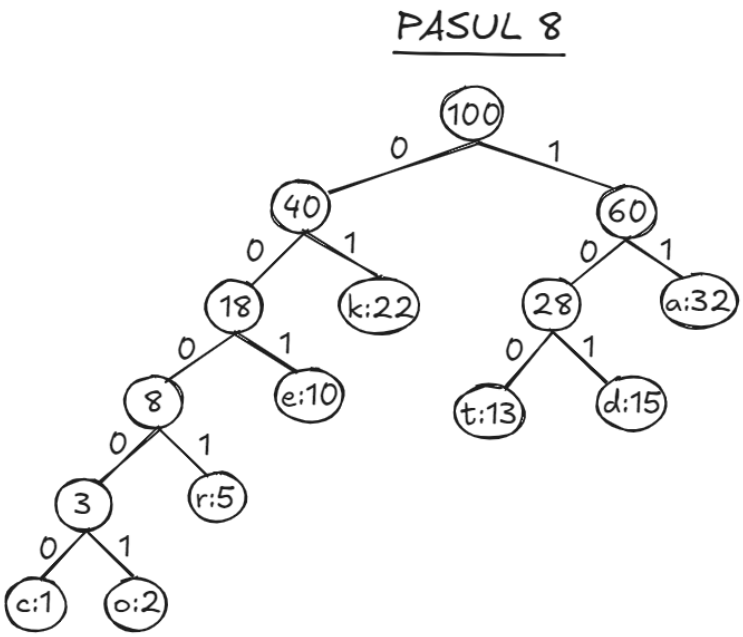

Figure 3: Huffman Coding

7. Demonstrati ca orice algoritm de sortare bazat pe comparatii are timp de rulare $\Omega(n \log n)$

Doua demonstratii complete pot fi gasite in **Tutoriat 3 - Limite inferioare pentru sortare**.

8. Rezolvati recurenta: $T(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + \log n$

Fiecare apel $T(x)$ va fi impartit in doua apeluri recursive $T(\frac{x}{4})$ si $T(\frac{3x}{4})$. De asemenea, fiecare apel cu dimensiunea x va avea complexitate $\log(x)$.

Daca facem arborele recursiv, radacina va fi n, care va fi impartita in $\frac{n}{4}$ si $\frac{3n}{4}$; acestea vor fi si ele impartite, la randul lor, in $(\frac{n}{16}, \frac{3n}{16})$ si $(\frac{3n}{16}, \frac{9n}{16})$, si asa mai departe.

Hai sa calculam complexitatea operatiilor de pe nivelul 2: $\log(\frac{n}{16})+\log(\frac{3n}{16})+\log(\frac{3n}{16})+\log(\frac{9n}{16}) = \log(\frac{n}{16} * \frac{3n}{16} * \frac{3n}{16} * \frac{9n}{16}) = \log(\frac{3^4}{2^{16}} * n^4) = \log(n^4)+\log(\frac{3^4}{2^{16}}) = 4*\log(n)+\log(\frac{3^4}{2^{16}}) \in \Theta(\log(n))$ (scapam de constante). Asadar, putem deduce ca formula generala pentru complexitatea unui nivel din arborele recursiv este: $\log(\frac{3^x}{4^y} * n^z) \in \Theta(\log(n))$.

Stiind complexitatea operatiilor unui nivel, trebuie sa determinam cate nivele sunt in arbore. Adancimea arborelui va fi data de ramura cu $T(\frac{3n}{4})$, deoarece $\frac{3n}{4}$ scade mai lent decat $\frac{n}{4}$ (evident). Noi trebuie sa ajungem de la dimensiunea n la dimensiunea 1, iar la fiecare pas micsoram dimensiunea cu $\frac{3}{4} \rightarrow n*(\frac{3}{4})^k = 1 \rightarrow \log(n*(\frac{3}{4})^k) = \log(1) \rightarrow \log(n)+\log((\frac{3}{4})^k) = 0 \rightarrow \log(n)+k*\log(\frac{3}{4}) = 0 \rightarrow k = -\frac{\log(n)}{\log(\frac{3}{4})} = \frac{\log(n)}{\log(\frac{4}{3})} \in \Theta(\log(n))$.

Avand $\Theta(\log(n))$ nivele, fiecare nivel avand complexitatea $\Theta(\log(n))$, putem afirma ca recurenta $T(n) = T(\frac{n}{4}) + T(\frac{3n}{4}) + \log n \in \Theta(\log^2(n))$.

9. Demonstrati ca: $\log n \in o(\sqrt{n})$

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{1}{n} * \frac{1}{\frac{1}{2} * n^{-\frac{1}{2}}} = 2 * \lim_{n\to\infty} \frac{n^{\frac{1}{2}}}{n} = 2 * \lim_{n\to\infty} n^{-\frac{1}{2}} = 0 \Rightarrow \log n \in o(\sqrt{n})$$

10. Verifica daca un arbore binar este BST

10.1. Cerinta

Se da un arbore binar cu **n** noduri in urmatorul format: se specifica radacina, iar pentru fiecare nod se dau fiul stang si fiul drept, daca acestia exista. De asemenea, fiecarui nod ii este asociat un numar intreg. Sa se decida daca acest arbore este un **arbore binar de cautare**. Pentru punctaj maxim, gasiti o solutie care sa ruleze in timp **O(n)**.

10.2. Rezolvare si pseudocod

Daca arborele respectiv ar fi un arbore binar de cautare, asta ar insemna ca **parcurgerea in inordine** ar trebui sa genereze o lista sortata de numere. Asadar, realizam o parcurgere in inordine, in timp ce retinem elementul precedent intr-o variabila auxiliara. Mereu vom compara valoarea elementului curent cu valoarea elementului anterior; daca vreodata gasim 2 elemente adiacente care nu sunt sortate, atunci returnam **false**.

Fiind doar o parcurgere a nodurilor, algoritmul are timp de rulare **O(n)** (deoarece va trece prin fiecare nod).

Algorithm 1: Verificare daca un arbore binar este BST

Input: Un arbore binar (rădăcină `root`)
Output: `true` dacă este BST, altfel `false`

```
1 Function inorder(Node* root, int& prev):
2     if root == null then
3         return true
4     if not inorder(root → left, prev) then
5         return false
6     if prev ≥ root → data then
7         return false
8     prev ← root → data
9     return inorder(root → right, prev)
```

11. Mediana a 2 vectori sortati

11.1. Cerinta

Se dau doi vectori **X** si **Y**, fiecare cu **n** numere sortate. Prezentati un algoritm care sa gaseasca mediana celor $2 * n$ elemente. Pentru punctaj maxim, gasiti o solutie care sa ruleze in timp $O(logn)$.

Mediana unei multimi sortate de **n** elemente este elementul de pe pozitia $[\frac{n}{2}]$ cand **n este impar** si media celor 2 elemente din mijloc cand **n este par**. De exemplu, pentru sirul **3, 1, 7, 6, 9**, mediana este **4**, iar pentru sirul **10, 3, 15, 1, 8, 6**, mediana este $\frac{6+8}{2} = 7$.

11.2. Rezolvare si pseudocod

O prima observatie importanta: lungimea totala este $2 * n$; astfel, o sa avem mereu lungime para, iar mediana o sa fie media celor 2 elemente din mijlocul sirului sortat.

In primul rand, o rezolvare foarte simpla (dar lenta) ar fi sa punem toate elementele intr-un vector si sa sortam vectorul, returnand media celor 2 elemente din mijloc. Din cauza sortarii, timpul de rulare este $O(n \log n)$.

Putem imbunatati solutia: faptul ca avem doi vectori sortati ne duce cu gandul la **interclasare / merge sort**. In functie de cum realizam interclasarea, putem avea complexitate de spatiu $O(n)$ (daca ne folosim de un vector auxiliar) sau $O(1)$ (daca facem in-place), iar timpul de rulare devine $O(n)$ (deoarece doar trecem prin cele $2 * n$ elemente).

Ca sa ajungem in continuare la timp de rulare $O(logn)$, problema se complica in mod considerabil. Reamintim: faptul ca lungimea totala a sirului este de $2 * n$ elemente ne permite sa il partitionam in doua subsiruri de lungime n; astfel, mediana este media dintre ultimul element din prima partitie si primul element din a doua partitie. Intrebarea este, cum obtinem aceste elemente in mod eficient?

In mod foarte evident, ambele partitii o sa contina niste elemente din **X** si niste elemente din **Y**. De exemplu: pentru **n=10**, partitiile o sa aiba cate 5 elemente; daca prima partitie contine primele doua elemente din **X** si primele trei elemente din **Y**, atunci a doua partitie o sa contina ultimele trei elemente din **X** si ultimele doua elemente din **Y**. Notam prima partitie cu A si a doua partitie cu B; de asemenea, $A = X_1 \cup Y_1$ si $B = X_2 \cup Y_2$, unde X_1 reprezinta elementele din X care se afla in A, iar Y_1 reprezinta elementele din Y care se afla in A (definitii similare pentru X_2 si Y_2). Deoarece A contine elemente mai mici decat B (evident), rezulta ca elementele din X_1 trebuie sa fie mai mici decat elementele din Y_2, iar elementele din Y_1 trebuie sa fie mai mici decat elementele din X_2. Stim ca mediana este media dintre ultimul element din A si primul element din B; **ultimul element din A** este, de fapt, $max(X_1, Y_1)$, iar **primul element din B** este $min(X_2, Y_2)$. Totusi, ele fiind sortate (deoarece X si Y sunt sortate), $max(X_1, Y_1) = max(last\_element(X_1), last\_element(Y_1))$, iar $min(X_2, Y_2) = min(first\_element(X_2), first\_element(Y_2))$.

In concluzie, problema a fost redusa la urmatoarea sarcina: trebuie sa gasim un pivot ca sa impartim vectorul X in X_1 si X_2 si vectorul Y in Y_1 si Y_2, astfel incat sa avem $A = X_1 \cup Y_1$ si $B = X_2 \cup Y_2$, cu urmatoarele doua constrangeri:

- $last\_element(X_1) \leq first\_element(Y_2)$. Daca aceasta proprietate este incalcata, inseamna ca X_1 are prea multe elemente si trebuie sa mutam pivotul mai in spate.
- $last\_element(Y_1) \leq first\_element(X_2)$. Daca aceasta proprietate este incalcata, inseamna ca am inclus prea putine elemente din X_1 (sau prea multe elemente din X_2) si trebuie sa mutam pivotul mai in fata.

O alta observatie importanta: lucram cu un singur pivot (care va fi pur si simplu un numar intreg), care va determina cate si care elemente luam din toate cele 4 subpartitii. Pentru 10 elemente si $pivot = 1$: A va contine primul element din X si primele patru elemente din Y, iar B va contine ultimele patru elemente din X si ultimul element din Y.

Pentru a gasi pivotul in mod eficient si a avea timp de rulare $O(\log(n))$, ne vom folosi de **cautare binara**. Presupunem ca pivotul se afla fix la mijlocul lui X (adica $left = 0, right = n, pivot = \frac{left+right}{2}$) si verificam proprietatile: daca prima proprietate este incalcata, trebuie sa cautam pivotul mai in stanga, eliminand cel putin un element din X_1 ($right = pivot - 1$); daca a doua proprietate este incalcata, trebuie sa mai adaugam elemente in X_1 si mutam pivotul spre dreapta ($left = pivot + 1$).

Algorithm 2: Mediana a 2 vectori sortati de lungime egala

Input: Doi vectori sortati X si Y de lungime n
Output: Mediana celor $2n$ elemente

```
1  left ← 0
2  right ← n
3  while true do
4      i ← ⌊(left+right)/2⌋
5      j ← n − i
6      X1 ← INT_MIN if i = 0, else X[i − 1]
7      X2 ← INT_MAX if i = n, else X[i]
8      Y1 ← INT_MIN if j = 0, else Y[j − 1]
9      Y2 ← INT_MAX if j = n, else Y[j]
10     if X1 > Y2 then
11         right ← i − 1
12     else if Y1 > X2 then
13         left ← i + 1
14     else
15         return (max(X1, Y1) + min(X2, Y2))/2
```