Tema 1

Ex1.

Construcția unui arbore binar de căutare se realizează prin inserția succesivă a elementelor ce îl alcătuiesc.

Considerăm f(n) complexitatea construirii arborelui binar de căutare cu n elemente.

1) Primul pas este studierea complexității medii. Fie timpul necesar inserării în arbore T[n]=T[n/2]+1
   $T[n] \in \theta(\log n)$
   Se va aplica teorema Master: a=1, b=2, g(n)=1
   Deci trebuie verificat faptul că $g(n) \in \theta(n^{\log_b a})$ $\Leftrightarrow$ $g(n) \in \theta(n^{\log_2 1})$ $\Leftrightarrow$ $g(n)\ \theta(n^0)$ $\Leftrightarrow$ $g(n) \in \theta(1)$ ceea ce este evident adevărat.
   În acest caz, timpul mediu de inserare este $\theta(\log n)$. Pentru construirea unui arbore binar de căutare cu n elemente este nevoie de inserări. Astfel timpul mediu pentru această operațSie este $\theta(n \log n)$ => $f(n) \in \theta(n \log n)$ => $f(n) \in \Omega(n \log n)$.

2) Vom lua în considerare si cazul în care numere ce alcătuiesc arborele sunt în ordine crescătoare, acesta fiind cel mai nefavorabil dintre cazuri. Astfel timpul necesar inserării în arbore va fi $T[n] = T[n-1] + 1$.
   Trebuie demonstrat faptul că $T[n] \in O(n)$ $\Leftrightarrow$ există c, n0 > 0 astfel încât oricare ar fi $n \geq n0$ avem $T[n] \leq c \cdot n$.

   Presupunem că $T[n-1] \leq c \cdot (n-1)$.

   $$T[n] = T[n-1] + 1 \leq c \cdot (n-1) + 1 = c \cdot n - c + 1 \leq c \cdot n \ \forall\ c \geq 1$$

   De aici rezultă faptul că timpul necesar inserării în arborele binar de cautare este $O(n)$.
   Deci $f(n) \in O(n^2)$.
   Mai trebuie să demonstrăm că $f(n) \in \Omega(n \log n)$, $\Leftrightarrow$ există c, n0 >0 astfel încât oricare ar fi n > n0 avem $f(n) \geq c \cdot n \cdot \log n$. Alegem c = 1, n0 = 1 =>$n^2 \geq n \log n$ adevarat $\forall$ $n \geq n0$. Asadar $F(n) \in \Omega(n \log n)$.

Din 1) si 2) avem faptul ca $f(n) \in \Omega(n \log n)$

Ex2.

Dacă f(n) = Θ(g(n)) si g(n) = Θ(h(n)) => f(n)= Θ(h(n))

a) f(n) ∈ Θ(g(n)) dacă există c1, c2, n0 (constante) >0 astfel încât oricare n>n0 avem
c1·g(n) ≤ f(n) ≤ c2·g(n) (1)
b) g(n) ∈ Θ(h(n)) dacă există c'1, c'2, n'0 (constante) >0 astfel încât oricare n'>n0 avem
c'1·h(n) ≤ g(n) ≤ c'2·h(n) (2)

➔ dacă înmulțim expresia (2) cu c1 avem:
c1·c'1·h(n) ≤ c1·g(n) ≤ f(n) (*1)
➔ dacă înmulțim expresia (2) cu c2 avem:
f(n) ≤ c2·g(n) ≤ c2·c'2·h(n) (*2)

➔ Din (*1) si (*2) avem: c1·c'1·h(n) ≤ c1·g(n) ≤ f(n) ≤ c2·g(n) ≤ c2·c'2·h(n) (*)

c) f(n) ∈ Θ(h(n)) dacă există c''1, c''2, n''0 (constante) >0 astfel incât oricare n''>n0 avem
c''1·h(n) ≤ f(n) ≤ c''2·h(n) (1)
Din (*) avem: c''1=c1·c'1>0, c''2=c2·c''2>0 si n''=max(n, n')

Ex3.

log n = o($\sqrt{n}$)

f(n) ∈o(g(n)) daca oricare ar fi c>0, exista n0 astfel încât oricare ar fi n≥n0 avem f(n)<c·g(n) ⇔

$\log n < c \cdot \sqrt{n}$

Considerăm m ca fiind baza logaritmului: $\log_m n < c \cdot \sqrt{n} \Leftrightarrow \frac{\log_m n}{\sqrt{n}} < c$

Vom folosi limite pentru a demonstra inegalitatea de mai sus astfel: fie limita $\lim_{n\to\infty} \frac{\log_m n}{\sqrt{n}}$.

Folosind regula lui L'Hospital avem:

$$\lim_{n\to\infty} \frac{\frac{1}{n \ln m}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2\sqrt{n}}{n \ln m} = \lim_{n\to\infty} \frac{2}{\sqrt{n} \ln m} = 0 < c \; \forall \, c > 0$$

⇨ $\frac{\log_a n}{\sqrt{n}} < c \; \forall \, c > 0$ =>$\log_a n < c\sqrt{n} \; \forall \, c > 0$

Prin calculele anterioare rezulta că log n = o($\sqrt{n}$).

Ex4.

citesc n

```
sum1=(1+n)*n/2;  //suma tuturor numerelor de la 1 la n
sum2=0   // suma numerelor introduce
pentru i=1,n executa
    citesc x
    Sum2 = sum2 + x
scrie sum2 – sum 1  //sum2 va avea in plus numarul care apare duplicat
```

Exemplu: n=5 v: 3 4 3 2 1 5

Elementul duplicat este: 3

Ex5.

```
mediana (v1[], n, v2[], m)
    daca n<=2 atunci
            iau mediana lui v2  // elementul din mijloc
            interclasez v1 cu mediana lui v2
            returnez mediana vectorului nou
    altfel
            x<- mediana lui v1
            x<- mediana lui v2
            daca x == y atunci
                    returneaza x


            altfel daca x < y atunci
                    renunt la prima jumatate a lui v1
```

renunt la a 2-a jumatate a lui v2

altfel

renunt la a 2-a jumatate a lui v1

renunt la prima jumatate a lui v2

mediana(v1_nou, v2_nou)

Fie T[n]=T[n/2]+1 complexitatea timp a algoritmului definit mai sus. Această recurență este corectă deoarece algoritmul este bazat pe metoda Divide et Impera.

In continuare voi utiliza Teorema Master astfel: f(n) ∈ $\theta\left(n^{log_b a}\right)$

->a=1;

->b=2;

->f(n)=1;

Înlocuim valorile lui a, b si f(n) și avem:

f(n) ∈ $\theta(n^{log_2 1})$ ⇔ f(n) ∈ $\theta(n^0)$ ⇔ f(n) ∈ $\theta(1)$ ⇔ f(1) ∈ $\theta(1)$ ceea ce este evident adevărat.

Din demonstrația de mai sus rezultă faptul că T[n] ∈ $\theta\left(n^{\log_b a} \cdot \log_b n\right)$ ⇔

T[n] ∈ $\theta\left(n^{\log_2 1} \cdot \log_2 n\right)$·T[n] ⇔ T[n] ∈ $\theta(n^0 \cdot \log_2 n)$ ⇔ T[n] ∈ $\theta(\log_2 n)$

⇨ T[n] ∈ O(logn) => complexitatea timp a algoritumului definit anterior este O(logn).

Ex6.

a. Se dă urmatorul exemplu:

Capacitatea mașinii: 32

Numărul de obiecte: 7

Dimesiunile obiectelor: 30, 28, 26, 24, 7, 4, 2

Procedându-se conform algoritmului greedy prezentat in cerință, primele 3 obiecte vor fi transportate individual (daca am aduna 30 cu 28, 28 cu 26 sau 26 cu 24 s-ar depăși capacitatea mașinii), al patrulea și al cincilea impreună și apoi ultimele 2. Astfel se efectuează 5 transporturi. Se poate observa că

primul si ultimul ar putea fi transportate împreună și la fel și al doilea cu penultimul. Astfel se efectuează doar 4 transporturi ceea ce este mai eficient decât rezultatul oferit de algoritmul greedy.

Prin demonstratia de mai sus am oferit un contraexemplu, așadar, din faptul ca algoritmul prezentat în cerință nu oferă de fiecare dată soluția optimă resulta ca acesta nu este optim.

b. Voi incerca in continuare sa gasesc o aranjare a obiectelor pentru care algoritmul greedy va face de doua ori mai multe drumuri decat daca obiectele ar fi aranjate altfel, astfel incat la fiecare drum spatiul din portbagaj sa fie utilizat cat mai optim.

Considerăm cel mai nefavorabilă aranjare a obiectelor astfel: pentru o capacitate n a mașinii avem n obiecte cu dimensiuni cuprinse între 1 și n, fiecare dimensiune apărând exact o singura dată. Aceste obiecte sunt asezate astfel: 1, n, 2, n-1 etc.

Spre exemplu pentru n=15 dimensiunile se vor afla în mulțimea {1, 2, ..., 15} și vor fi așezate dupa cum urmeaza: 1, 15, 2, 14, 3, 13, 4, 12, 5, 11, 6, 10, 7, 9, 8.

In acest caz particular, algoritmul greedy prezentat în cerință va efectua 15 drumuri, adică un drum pentru fiecare obiect în parte deoarece nu poate combina obiectele consecutive, suma lor depășind capacitatea mașinii.

Fie OPT soluția optimă. Aceasta s-ar atinge prin gruparea elementelor a căror suma este n. În exemplul de mai sus n este impar așa că se vor obtine $(n - 1)/2$ perechi întrucât numărul n nu poate fi grupat cu altcineva. În acest caz $OPT = (n - 1)/2 + 1$ drumuri. În cazul în care n este par se pot forma $(n - 2)/2$ perechi întrucat numerele n si n/2 nu se pot grupa cu nimeni. În acest caz $OPT = (n - 2)/2 + 2$ drumuri. Luând în considerare cele 2 cazuri prezentate anterior vor fi necesare cel mult $OPT = n/2 + 1$ drumuri.

Solutia generată de algoritmul greedy din cerință are n drumuri in cel mai nefavorabil caz, iar solutia prezentata mai sus ar avea $n/2 + 1$ drumuri. Avem $n < 2 \cdot (n/2 + 1)$. În acest caz rezulta faptul ca $greedy < 2OPT$. Din faptul ca am lucrat cu cazul cel mai nefavorabil înseamnă cî algoritmul greedy prezentat efectuează cel mult 2OPT drumuri.