Analiza Algoritmilor
Tema 1

Dan-George Filimon
322 CA

26 decembrie 2010

1	2	3	4	5	6	7	8	9	Total

1 Rezolvați exercițiile:

1. Ce relații de ordine există între algoritmii $A \in O(n)$, $B \in o(n^2)$, $C \in \omega(n)$ și $D \in \Theta(n^2)$?
   R. Explicitând complexitățile algoritmilor și notând cu T_A, T_B, T_C și T_D timpii de execuție ai acestora:

$$\begin{array}{rll} \exists\, c_A, n_0 > 0 & \text{astfel încât} & \forall\, n \geq n_0,\ T_A \leq c_A n \\ \forall\, c_B > 0, \exists\, n_0 > 0 & \text{astfel încât} & \forall\, n \geq n_0,\ T_B \leq c_B n^2 \\ \forall\, c_C > 0, \exists\, n_0 > 0 & \text{astfel încât} & \forall\, n \geq n_0,\ T_C \geq c_C n \\ \exists\, c_{D1}, c_{D2}, n_0 > 0 & \text{astfel încât} & \forall\, n \geq n_0,\ c_{D1} n^2 \leq T_D \leq c_{D2} n^2 \end{array}$$

   Deci, algoritmul A are complexitate cel mult liniară, B are complexitate strict subpătratică, C strict supraliniară și D pătratică. Astfel, cu siguranță:

$$T_A < T_C,\ T_B < T_D \text{ și } T_A < T_D$$

   Complexitățile algoritmilor B și C (respectiv D și C) nu pot fi comparte exact. Subpătratic poate însemna și liniar, e clar că $n \in o(n^2)$; similar, supraliniar poate însemna pătratic, $n^2 \in \omega(n) \Rightarrow T_C > T_B$. De altfel și $n^3 \in \omega(n)$, deci algoritmul C ar putea avea chiar o complexitate

mai mare decât D; la fel cum s-ar putea ca $T_C \in \Theta(n \log_2 n)$ și $T_B \in \Theta(n \log^2 n)$, caz în care $T_C < T_B < T_D$.

2. Găsiți răspunsul și justificați egalitatea: $\Omega(f(n)) + o(f(n)) = ...(f(n))$.
R. Fie $g(n) \in \Omega(f(n))$ și $h(n) \in o(f(n))$. Atunci, $\exists\, c, n_0 > 0$ astfel încât $\forall\, n \geq n_0$, $g(n) \geq cf(n)$ și $\forall d > 0, \exists\, n_1 > 0$ astfel încât $\forall\, n \geq n_1$, $h(n) \leq df(n)$. Trebuie deci aflat unde se află $g(n) + h(n)$. Folosind forma asimptotică, $\lim_{n\to\infty} \frac{g(n)}{f(n)} \geq c$ și $\lim_{n\to\infty} \frac{h(n)}{f(n)} = 0$. Atunci, folosind liniaritatea operației de trecere la limită rezultă că $\lim_{n\to\infty} \frac{g(n)+h(n)}{f(n)} \geq c$. Rezultă deci că $g(n) + h(n) \in \Omega(f(n))$, sau $\Omega(f(n)) + o(f(n)) = \Omega(f(n))$.

2 Pentru fiecare dintre afirmațiile de mai jos, scrieți valoarea de adevăr și justificați-o:

1. $f(n) = \Theta(n)$ și $g(n) = \Omega(n) \Rightarrow f(n)g(n) = \Omega(n^2)$:
R. $f(n) \in \Theta(n) \Rightarrow f(n) \in \Omega(n) \Rightarrow \exists\, c, n_f > 0 : f(n) \geq cn$, $\forall n \geq n_f$. Analog, $\exists\, d, n_g > 0 : g(n) \geq dn$, $\forall n \geq n_g$. Deoarece $f(n)$ și $g(n)$ sunt funcții pozitive și crescătoare, $f(n)g(n) \geq (cn)(dn)$, $\forall n \geq \max(n_f, n_g)$. Deci, $f(n)g(n) \geq en^2$, $\forall n \geq n_{fg}$, unde $e = cd$, $n_{fg} = \max(n_f, n_g) \Rightarrow f(n)g(n) \in \Omega(n^2)$. **adevărat**.

2. $f(n) = \Theta(1) \Rightarrow n^{f(n)} = O(n)$:
R. $f(n) \in \Theta(1) \Rightarrow f(n) \in O(n) \Rightarrow \exists\, c, n_0 > 0 : f(n) \leq c$, $\forall n \geq n_0$. Deoarece pentru $a < b$, $\lim_{x\to\infty} \frac{x^a}{x^b} = \lim_{x\to\infty} x^{a-b} = \frac{1}{\infty} = 0$, rezultă că asimptotic $n^{f(n)} \leq n^c$. Ceea ce înseamnă că $n^{f(n)} \notin O(n)$ pentru $c > 1$. **fals**.

3. $f(n) = \Omega(n)$ și $g(n) = O(n^2) \Rightarrow \frac{f(n)}{g(n)} = O(n)$:
R. Fie c, n_f constantele pentru care $f(n) \geq cn$, $\forall n \geq n_f$ și d, c_g constantele pentru care $g(n) \leq dn^2$, $\forall n \geq n_g$. Atunci, $\frac{1}{g(n)} \geq \frac{1}{dn^2}$ și deci $\frac{f(n)}{g(n)} \geq \frac{c}{dn} \Rightarrow \frac{f(n)}{g(n)} \in \Omega(n)$. Deoarece și funcții din $O(n) \ni n \geq k\frac{1}{n}$ pentru un k fixat și de la un n_0, nu se poate trage nicio concluzie despre apartenența lui $\frac{f(n)}{g(n)}$ la $O(n)$. **fals**.

4. $f(n) = O(n^2)$ și $g(n) = O(n) \Rightarrow f(g(n)) = O(n^3)$:
R. $f(n) \leq cn^2$ și $g(n) \leq dn \Rightarrow f(g(n)) \leq f(dn) \leq c(dn)^2 = (cd^2)n^2 \Rightarrow f(g(n)) \in O(n^2) \subset O(n^3)$. De fapt, la fel de adevărat este că și $f(n), g(n) \in O(n^3)$. **adevărat**.

5. $f(n) = O(\log n) \Rightarrow 2^{f(n)} = O(n)$:
   R. Există un $c > 0$ fixat astfel încât $f(n) \leq c \log n$ pentru $n \geq n_0$. Atunci, cum funcția exponențială este strict crescătoare, $2^{f(n)} \leq 2^{c \log n} = 2^{\log n^c} = n^c$. Deci, doar pentru $c \leq 1$, $2^{f(n)} \in O(n)$. **fals**.

6. $f(n) = \Omega(n) \Rightarrow 2^{f(n)} = \Omega(n)$:
   R. $f(n) \geq cn \Rightarrow 2^{f(n)} \geq 2^{cn}$. Deoarece asimptotic, $2^{cn} \to \infty$, $\forall c > 0$ iar creșterea este exponențială, $2^{f(n)} \geq dn$, pentru un d fixat, dn având o creștere liniară. Deci, $2^{f(n)} \in \Omega(f(n))$ (chiar în $\omega(n)$). **adevărat**.

3 Fie n comutatoare corespunzând unor n becuri, $(1..n)$, inițial toate stinse. Se aplică următorii n pași: la pasul k se modifică starea comutatoarelor din k în k începând cu primul ($1 \leq k \leq n$). Câte becuri rămân aprinse?

1. Scrieți un algoritm iterativ în pseudocod care simulează cei n pași descriși anetrior și determinați complexitatea sa.
   R. B va fi vectorul de becuri, în care 0 codifică starea *stins* iar 1 codifică starea *aprins*:

```
    for i = 1 → n do
      B[i] = 0
    end for
    for k = 1 → n do
 5:   i = 1
      while i ≤ n do
        B[i] = (B[i] + 1)%2
        i = i + k
      end while
10: end for
    sum = 0
    for i = 1 → n do
      sum = sum + B[i]
    end for
15: return sum
```

   Admițând ca operații elementare atribuirile în $B[i]$, în prima bulcă *for* se fac n operații, ca și în ultima în care se calculează suma, iar în interiorul buclei *for* care chiar simulează algoritmul se fac la fiecare pas $\frac{n}{k}$, $k = 1 \to n$ operații.

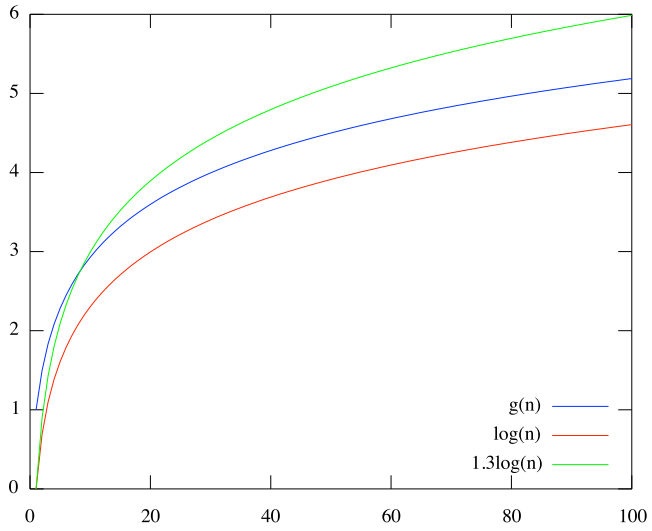
Astfel, $T(n) = 2n + \sum_{k=1}^{n} \frac{n}{k} = 2n + n \sum_{k=1}^{n} \frac{1}{k}$. Să notăm $g(n) = \sum_{k=1}^{n} \frac{1}{k}$. Plecând de la inegalitatea:

$$\frac{1}{k+1} \leq \int_k^{k+1} \frac{1}{x}\,\mathrm{d}x \leq \frac{1}{k}$$

și însumând pentru $k = 1 \to n$, rezultă:

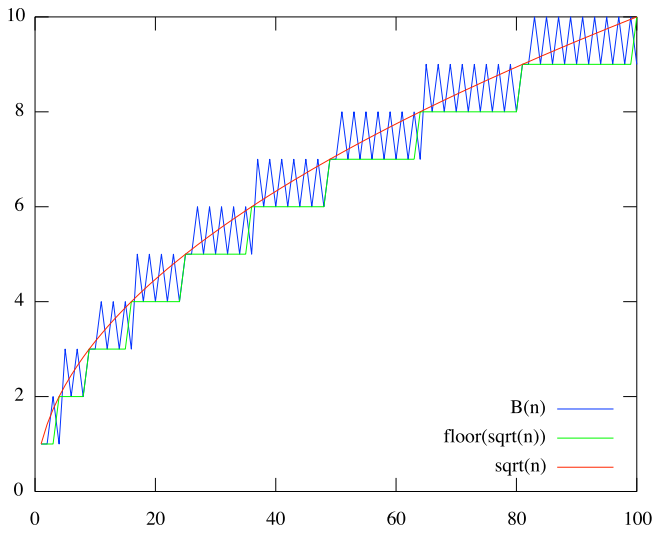
$$\begin{aligned}
\sum_{k=1}^{n} \frac{1}{k+1} \leq \sum_{k=1}^{n} \int_k^{k+1} \frac{1}{x}\,\mathrm{d}x \leq \sum_{k=1}^{n} \frac{1}{k} \Leftrightarrow \\
\sum_{k=2}^{n} \frac{1}{k} - \frac{1}{n+1} \leq \int_1^{n+1} \frac{1}{x}\,\mathrm{d}x \leq g(n) \Leftrightarrow \\
g(n) - \left(1 + \frac{1}{n+1}\right) \leq \log x|_1^{n+1} \leq g(n) \Leftrightarrow \\
\log(n+1) \leq g(n) \leq \log(n+1) + \left(1 + \frac{1}{n+1}\right) \Leftrightarrow \\
\log(n+1) \leq g(n) \leq c \log(n+1),\ c > 1 \Leftrightarrow \\
g(n) \in \Theta(\log n)
\end{aligned}$$

În plus, grafic funcția $g(n)$ (albastru) cu $\log n$ (roșu) respectiv $1.3 \log n$ (verde), se observă că $g(n) \in \Theta(\log n)$.



Deci, $T(n) = n\,(2 + g(n)) \in \Theta(n \log n)$.

2. Găsiți un algoritm cât mai eficient de rezolvare a problemei. *Ce pași acționează asupra comutatorului k?* Analizați complexitatea folosind notația Theta.
R. Intuitiv e clar că asupra unui bec b acționează toți pașii k care sunt divizori ai lui $b - 1$. Dacă $T'(n)$ este complexitatea noului algoritm, vrem ca $T'(n) \in o(n \log n)$. Ar trebui cumva renunțat la bucla interioară *while*, ceea ce ar putea conduce la $T'(n) \in \Theta(n)$, dar asta ar însemna determinarea parității numărului de divizori al unui număr în $\Theta(1)$, ceea ce nu este posibil fiindcă determinarea divizorilor implică obligatoriu o factorizare a numărului ceea ce este de fapt cel puțin la fel de costisitoare ca $\Theta(\log n)$. Înseamnă că ar trebui să existe o formulă care să se bazeze pe faptul că se fac exact n pași.
Făcând graficul numărului de becuri rămase aprinse după n pași, $B(n)$ constatăm:



Cu ajutorul acestui grafic, putem observa că $B[n] = \lfloor \sqrt{n} \rfloor + (n \mod 2)$. Comportamentul oscilant dat de factorul $n \mod 2$ apare fiindcă numărarea becurilor începe de la 1, deci primul comutator va fi acționat la fiecare pas, ceea ce înseamnă că primul bec depinde exclusiv de numărul de pași. Pentru un număr oarecare m, numărul său de divizori va fi par

atunci când $\forall d_i$, divizori ai lui m, d_i/m, $\frac{m}{d_i}/m$ și în plus, $\frac{m}{d_i} \neq d_i$. Singurul caz când asta nu se întâmplă sunt pătratele perfecte, $m = d_i^2$, al căror număr de la 1 la n sigur va fi $\lfloor\sqrt{n}\rfloor$. Atunci, putem scrie direct:

return $\lfloor\sqrt{n}\rfloor + (n \mod 2)$

care are desigur complexitate $\Theta(1)$.

4 Se consideră un tablou de n numere reale $X[1..n]$. Ne interesează să răspundem la întrebări de genul: **care este maximul subtabloului** $X[i..j]$? Practic, va trebui sa gasim un algoritm eficient per întrebare. Pentru aceasta vom împărți vectorul in $\frac{n}{k}$ "bucăți" de lungime k (eventual cu excepția ultimei "bucăți"). Pentru fiecare dintre aceste "bucăți" vom calcula maximul și il vom retine într-un vector max.

1. Scrieți în pseudocod un algoritm eficient care găsește maximul din subvectorul $X[i..j]$. Analizați complexitatea rezolvării a m întrebări, luând în calcul și construirea lui max.
   R. Pentru simplitate, conisderăm indexarea de la 0, cele m întrebări într-o matrice de m linii și 2 coloane, $Q[0..m-1]$ și punem răspunsurile într-un vector $Ans[0..m-1]$.

```
for i = 0 → n/k − 1 do
  max[i] = 0
end for
for i = 0 → n − 1 do
  if X[i] > max[i/k] then
    max[i/k] = X[i]
  end if
end for
for q = 0 → m − 1 do
  l = Q[q][0]/k
  r = Q[q][1]/k
  Ans[q] = 0
  for b = l + 1 → r − 1 do
    if max[b] > Ans[q] then
      Ans[q] = max[b]
    end if
  end for
```

```
    li = Q[q][0]%k
    ri = Q[q][1]%k
20: if li = 0 then
      if max[l] > Ans[q] then
        Ans[q] = max[l]
      end if
    else
25:   for j = kl + li → k(l + 1) − 1 do
        if max[j] > Ans[q] then
          Ans[q] = max[j]
        end if
      end for
30: end if
    if ri = k − 1 then
      if max[r] > Ans[q] then
        Ans[q] = max[r]
      end if
35: else
      for j = kr → kr + ri do
        if max[j] > Ans[q] then
          Ans[q] = max[j]
        end if
40:   end for
    end if
  end for
```

Pentru simplitate presupunem că vectorul inițial X se poate împărți exact în bucăți de câte k elemente. La fiecare întrebare, se determină bucățile conținute în subtabloul $X[i..j]$, bucățile complete putând fi verificate direct, iar cele (maxim) două bucăți de la capete trebuie verificate element cu element fiindcă s-ar putea ca maximul lor să fie chiar un element care nu este în intervalul $i..j$.
Astfel, rezolvarea unei întrebări, ale cărei capete în vectorul X sunt i și j se face în $(k - i\%k) + (j/k - i/k - 1) + (k - j\%k)$ pași.

- $k - i\%k$ este numărul de elemente care trebuie verificate atunci când capătul subtabloului se află în interiorul bucății i/k, cu precizarea că în cazul particular, $i\%k = 0$, verificarea maximului se

face într-un singur pas, fiindcă bucata este conținută complet în subtabloul $X[i..j]$.

- $(j/k - i/k - 1)$ este numărul de bucăți care sunt cuprinse în subtablou și pentru care se poate verifica direct maximul lor.

În cazul cel mai defavorabil, trebuie verificate $\frac{n}{k} - 2$ bucăți complet iar din cele două de la capete trebuie verificate $k - 1$ elemente. Acest caz corespunde unui subtablou $X[1..n-1]$. Atunci, se verifică $T(n,k) = 2k - 2 + \frac{n}{k} - 2$ elemente.
Complexitatea va fi dată de numărul de verificări respectiv atribuiri și per total, în cazul cel mai defavorabil se vor face $\tau(n,m,k) = 2n + mT(n,k)$ operații. Fiindcă complexitatea depinde de 3 variabile, n, m, k din care 2 sunt independente, n, m, clasa în care trebuie încadrat algoritmul depinde de magnitudinea fiecărei variabile.
De exemplu, pentru un număr mare de întrebări, $m \gg n$, și pentru bucăți de dimensiune $\sqrt{n}$, complexitatea în cazul cel mai defavorabil va fi $\Theta(m\sqrt{n})$, în timp ce pentru un număr de întrebări comparabil cu numărul de elemente, $m \approx n$ și bucăți de dimensiune 1, $k = 1$, complexitatea în același caz va fi $\Theta(n^2)$.

2. Cum ar trebui ales k astfel încât să se obțină o complexitate cât mai mică?
R. Deoarece numărul de întrebări este independent de dimensiunea bucăților, va fi suficient să minimizăm funcția $T(n,k)$, pentru o valoare fixată a lui n, $T(k) = T(n,k) = 2k - 2 + \frac{n}{k} - 2$. Atunci:

$$T(k) = 2k + \frac{n}{k} - 4 \Leftrightarrow \frac{\mathrm{d}}{\mathrm{d}k}\left[T(k)\right] = 2 - \frac{n}{k^2}$$

Funcția $T(k)$ atinge un extrem pentru $T'(k) = 0$. Deci:

$$2 - \frac{n}{k^2} = 0 \Rightarrow k_0 = \sqrt{\frac{n}{2}}$$

Să comparăm $T(k_0) = T(\sqrt{\frac{n}{2}})$ cu $T(\sqrt{n})$ pentru a vedea ce fel de punct de extrem este k_0, de minim sau de maxim. Obținem:

$$T(k_0) = 2\sqrt{\frac{n}{2}} + \frac{n}{\sqrt{\frac{n}{2}}} - 4 = 2\sqrt{2n} - 4$$
$$T(\sqrt{n}) = 2\sqrt{n} + \sqrt{n} - 4 = 3\sqrt{n} - 4$$

Și cum $2\sqrt{2} < 3 \Rightarrow k_0 = \sqrt{\frac{n}{2}}$ este un punct de minim.

5 Intuiți o soluție și rezolvați prin metoda substituției recurența:

$$T(n) = T\left(\frac{n}{2} + \sqrt{n}\right) + n$$

R. Vom încerca să demonstrăm că $T(n) \in O(n \log n)$. Astfel, ar trebui să existe $c > 0$ și un $n_0 > 0$ astfel încât $\forall n \geq n_0$, $T(n) \leq cn \log n$.
Considerăm ipoteza de inducție $T(\frac{n}{2}+\sqrt{n}) \in O(n \log n)$ cu aceleași constante, deci:

$$T\left(\frac{n}{2} + \sqrt{n}\right) \leq c\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right) \Leftrightarrow \lim_{n\to\infty} \frac{T\left(\frac{n}{2} + \sqrt{n}\right)}{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)} \leq c$$

Deoarece $T(n) = T\left(\frac{n}{2} + \sqrt{n}\right) + n$, vrem să demonstrăm că $\lim_{n\to\infty} \frac{T(n)}{n \log n} \leq c$.
Să explicităm partea din stânga a inegalității folosind ipoteza de inducție:

$$l = \lim_{n\to\infty} \frac{T(n)}{n \log n} = \lim_{n\to\infty} \frac{T\left(\frac{n}{2} + \sqrt{n}\right) + n}{n \log n} =$$

$$\lim_{n\to\infty} \frac{T\left(\frac{n}{2} + \sqrt{n}\right) + n}{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)} \frac{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)}{n \log n} =$$

$$\lim_{n\to\infty} \left[\frac{T\left(\frac{n}{2} + \sqrt{n}\right)}{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)} + \frac{n}{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)} \right] \frac{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)}{n \log n}$$

Al doilea termen din paranteza dreaptă va tinde la 0 fiindcă:

$$\lim_{n\to\infty} \frac{n}{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)} = \lim_{n\to\infty} \frac{1}{\left(\frac{1}{2} + \frac{1}{\sqrt{n}}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)} =$$

$$\lim_{n\to\infty} \frac{2}{\log\left(\frac{n}{2} + \sqrt{n}\right)} = 0$$

Se pune deci problema determinării lui:

$$k = \lim_{n\to\infty} \frac{\left(\frac{n}{2} + \sqrt{n}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)}{n \log n} = \lim_{n\to\infty} \frac{\left(\frac{1}{2} + \frac{1}{\sqrt{n}}\right) \log\left(\frac{n}{2} + \sqrt{n}\right)}{\log n} =$$

$$\frac{1}{2} \lim_{n\to\infty} \frac{\log\left(\frac{n}{2} + \sqrt{n}\right)}{\log n} = \frac{1}{2} \lim_{n\to\infty} \frac{\log\left[n\left(\frac{1}{2} + \frac{1}{\sqrt{n}}\right)\right]}{\log n} = \frac{1}{2} \lim_{n\to\infty} \frac{\log n - \log 2}{\log n} =$$

$$\frac{1}{2}\left(1 - \lim_{n\to\infty} \frac{\log 2}{\log n}\right) = \frac{1}{2}$$

Atunci, l devine:

$$l = \lim_{n\to\infty}\left[\frac{T\left(\frac{n}{2}+\sqrt{n}\right)}{\left(\frac{n}{2}+\sqrt{n}\right)\log\left(\frac{n}{2}+\sqrt{n}\right)} + 0\right]\frac{1}{2} \leq \frac{c}{2} \leq c \Rightarrow \lim_{n\to\infty}\frac{T(n)}{n\log n} \leq c \Rightarrow$$
$$T(n) \in O(n\log n)$$

R2. Vom demonstra că $T(n) \in \Theta(n) \Leftrightarrow T(n) \in \Omega(n) \wedge T(n) \in O(n)$.

- $T(n) \in \Omega(n)$: $\exists\, c, n_1 > 0 : T(n) \geq cn, \forall n \geq n_1$. Presupunem $T(\frac{n}{2}+\sqrt{n}) \in \Omega(n) \Leftrightarrow T(\frac{n}{2}+\sqrt{n}) \geq c\left(\frac{n}{2}+\sqrt{n}\right)$. Atunci, $T(n) \geq \left(\frac{c}{2}+1\right)n + c\sqrt{n} \geq cn$ deoarece pentru $c = 1$, $1 \geq -\frac{\sqrt{n}}{2}$.
- $T(n) \in O(n)$: $\exists\, d, n_2 > 0 : T(n) \leq dn, \forall n \geq n_2$. Presupunem $T(\frac{n}{2}+\sqrt{n}) \in O(n) \Leftrightarrow T(\frac{n}{2}+\sqrt{n}) \leq d\left(\frac{n}{2}+\sqrt{n}\right)$. Atunci, $T(n) \leq \left(\frac{d}{2}+1\right)n + d\sqrt{n} \leq dn$ deoarece pentru $d > 4$, $d \leq \left(\frac{d}{2}-1\right)\sqrt{n}$.

Deci, $T(n) \in \Theta(n)$.

6 Rezolvați următoarele recurențe:

1. $T(n) = T(\sqrt{n}) + \log n$
   R. Fie $n = 2^m$. Atunci recurența devine:

   $$T(2^m) = T(2^{\frac{m}{2}}) + m$$

   Notând $T(2^m) = S(m)$, obținem:

   $$S(m) = S\left(\frac{m}{2}\right) + m$$

   Explicitând $S(\frac{m}{2^k})$:

   $$\begin{array}{rcl} S(m) & = & S\left(\frac{m}{2}\right) + m \\ S\left(\frac{m}{2}\right) & = & S\left(\frac{m}{4}\right) + \frac{m}{2} \\ S\left(\frac{m}{4}\right) & = & S\left(\frac{m}{8}\right) + \frac{m}{4} \\ S\left(\frac{m}{8}\right) & = & S\left(\frac{m}{16}\right) + \frac{m}{8} \\ & \ldots & \\ S\left(\frac{m}{2^{\lfloor\log m\rfloor}}\right) & = & 1 \end{array}$$

Adunând toate relațiile de mai sus, se reduc termenii de forma $S(\frac{m}{2^k})$, $k > 0$:

$$S(m) = \sum_{k=0}^{\lfloor \log m \rfloor} \frac{m}{2^k} = m \frac{1 - \frac{1}{2^{\lfloor \log m \rfloor + 1}}}{1 - \frac{1}{2}} = 2m \frac{2m - 1}{2m} = 2m - 1$$

Astfel, deoarece $m = \log m \Rightarrow T(n) = 2 \log n - 1$.

2. $T(n) = 2T(\sqrt{n}) + 1$
R. Fie $n = 2^m$. Recurența devine:

$$T(2^m) = 2T\left(2^{\frac{m}{2}}\right) + 1$$

Notând $S(m) = T(2^m)$ și explicitând:

$$\begin{array}{rcccrcl}
 & & & & S(m) & = & 2S\left(\frac{m}{2}\right) + 1 \\
S\left(\frac{m}{2}\right) & = & 2S\left(\frac{m}{4}\right) + 1 & \Leftrightarrow & 2S\left(\frac{m}{2}\right) & = & 4S\left(\frac{m}{4}\right) + 2^1 \\
S\left(\frac{m}{4}\right) & = & 2S\left(\frac{m}{8}\right) + 1 & \Leftrightarrow & 4S\left(\frac{m}{4}\right) & = & 8S\left(\frac{m}{8}\right) + 2^2 \\
S\left(\frac{m}{8}\right) & = & 2S\left(\frac{m}{16}\right) + 1 & \Leftrightarrow & 8S\left(\frac{m}{8}\right) & = & 16S\left(\frac{m}{16}\right) + 2^3 \\
 & & & \cdots & & & \\
S\left(\frac{m}{2^{\lfloor \log m \rfloor}}\right) & = & 1 & \Leftrightarrow & 2^{\lfloor \log m \rfloor} S\left(\frac{m}{2^{\lfloor \log m \rfloor}}\right) & = & 2^{\lfloor \log m \rfloor}
\end{array}$$

Adunând toate egalitățile rezultă că:

$$S(m) = \sum_{k=0}^{\lfloor \log m \rfloor} 2^k = \frac{2^{\lfloor \log m \rfloor + 1} - 1}{2 - 1} = 2m - 1 \Rightarrow T(n) = 2 \log n - 1$$

7 Fie tipul de date `BTree` definit prin constructorii:

`BTempty`: $\rightarrow$ `BTree`

`BTnode`: `BTree`$\times$`T`$\times$`BTree` $\rightarrow$ `BTree`

și definițiile:

1. `flattenTree(t)`: `BTree` $\rightarrow$ `List`

   (a) `flattenTree(BTempty) = []`

   (b) `flattenTree(BTnode(left, i, right)) = flattenTree(left) ++ [i] ++ flattenTree(right)`

2. `numBTelem(t): BTree` $\rightarrow$ `N`

   (a) `numBTelem(BTempty) = 0`

   (b) `numBTelem(BTnode(left, i, right)) = 1 + numBTelem(left) + numBTelem(right)`

3. `length(l): List` $\rightarrow$ `N`

   (a) `length([]) = 0`

   (b) `length(h:t) = 1 + length(t)`

Știind că avem constructorii `[]` și `h:t` pentru tipul `List` și că operatorul `++` (concatenarea a două liste) este definit astfel încât este adevărat `length(l1 ++ l2) = length(l1) + length(l2)`, $\forall$ `l1, l2` $\in$ `List`, demonstrați că:

`numBTelem(t) = length(flattenTree(t))`, $\forall$ `t` $\in$ `BTree`

Vom folosi inducția structurală pentru a demonstra afirmația de mai sus. O vom demonstra pentru constructorii nulari ai tipului `BTree`, mai exact `BTempty` (cazul de bază):

- *left*: `numBTelem(BTempty)` $\overset{2a}{=}$ `0`
- *right*: `length(flattenTree(BTempty))` $\overset{1a}{=}$ `length([])` $\overset{3a}{=}$ `0`

Considerăm adevărate ipotezele de inductive `numBTelem(left) = length(flattenTree(left))` și `numBTelem(right) = length(flattenTree(right))` Pentru pasul de inducție vom demonstra că *left* = *right* pentru constructorii din Σ_1, adică pentru `BTnode`:

- *left*: `numBTelem(BTnode(left, i, right))` $\overset{2b}{=}$ `1 + numBTelem(left) + numBTelem(right)` $\overset{\text{Ip. Ind.}}{=}$ `1 + length(flattenTree(left)) + length(flattenTree(right))`
- *right*: `length(flattenTree(BTNode(left, i, right))` $\overset{1b}{=}$ `length(flattenTree(left) ++ [i] ++ flattenTree(right))` $\overset{\text{Def. ++}}{=}$ `length(flattenTree(left)) + length([i] ++ flattenTree(right))` $\overset{\text{Def. ++}}{=}$ `length(flattenTree(left)) + length([i]) + length(flattenTree(right))` $\overset{3b}{=}$ `length(flattenTree(left)) + 1 + length([]) + length(flattenTree(right))` $\overset{3a}{=}$ `1 + length(flattenTree(left)) + length(flattenTree(right))`

Deci, *left* = *right*, pentru toți constructorii tipului `BTree`, deci propoziția este mereu adevărată.

8 O expresie aritmetică complet parantezată deste definită ca:

```
0, 1, x, [e1 + e2], [e1 * e2], [-e2]
```

2 constructori nulari, 1 constructor extern, și 3 constructori interni. Notăm acest tip de date `E` și definim operatorii:

1. `eval(e, n)`: `E`×`N` → `N`
   (a) `eval(0, n) = 0`
   (b) `eval(1, n) = 1`
   (c) `eval(x, n) = n`
   (d) `eval([e1 + e2], n) = eval(e1, n) + eval(e2, n)`
   (e) `eval([e1 * e2], n) = eval(e1, n) * eval(e2, n)`
   (f) `eval([-e], n) = -eval(e, n)`
2. `subst(e, f)`: `E`×`E` → `E`
   (a) `subst(0, f) = 0`
   (b) `subst(1, f) = 1`
   (c) `subst(x, f) = f`
   (d) `subst([e1 + e2], f) = [subst(e1, f) + subst(e2, f)]`
   (e) `subst([e1 * e2], f) = [subst(e1, f) * subst(e2, f)]`
   (f) `subst([-e], f) = [-subst(e, f)]`

Demonstrați prin inducție structurală că $\forall\, e,\ f \in \texttt{E},\ n \in \texttt{N}$, este adevărată proprietatea:

```
eval(subst(e, f), n) = eval(e, eval(f, n))
```

R. Vom demonstra că proprietatea este adevărată fixând f și n și considerând diversele posibilități pentru a construi e. Vom demonstra pentru următorii constructori proprietatea:

1. *constructori nulari*: `0`, deoarece pentru `1` relațiile sunt identice;
   - *left*: `eval(subst(0, f), n)` $\overset{2a}{=}$ `eval(0, n)` $\overset{1a}{=}$ `= 0`
   - *right*: `eval(0, eval(f, n))` $\overset{1a}{=}$ `0`

2. *constructori externi*: `x`;

   - *left*: `eval(subst(x, f), n)` $\overset{2c}{=}$ `eval(f, n)`
   - *right*: `eval(x, eval(f, n))` $\overset{1c}{=}$ `eval(f, n)`

3. *constructori interni*: `[e1 + e2]` (pentru `[e1 * e2]` relațiile sunt analoage) și pentru `[-e]`.
   Presupunem ca ipotezele inductive `eval(subst(e1, f), n) = eval(e1, eval(f, n)` și `eval(subst(e2, f), n) = eval(e2, eval(f, n)` sunt adevărate în continuare.
   $e =$ `[e1 + e2]`

   - *left*: `eval(subst([e1 + e2], f), n)` $\overset{2d}{=}$ `eval([subst(e1, f) + subst(e2, f)], n)` $\overset{1d}{=}$ `eval(subst(e1, f), n) + eval(subst(e2, f), n)` $\overset{\text{Ip.Ind.}}{=}$ `eval(e1, eval(f, n)) + eval(e2, eval(f, n))`
   - *right*: `eval([e1 + e2], eval(f, n))` $\overset{1d}{=}$ `eval(e1, eval(f, n)) + eval(e2, eval(f, n)`

   $e =$ `[-e1]`

   - *left*: `eval(subst([-e1], f), n)` $\overset{2f}{=}$ `eval([-subst(e1, f)], n)` $\overset{1f}{=}$ `-eval(subst(e1, f), n)` $\overset{\text{Ip.ind}}{=}$ `-eval(e1, eval(f, n))`
   - *right*: `eval([-e1], eval(f, n))` $\overset{1f}{=}$ `-eval(e1, eval(f, n))`

Astfel, *left* = *right*, pentru toți constructorii tipului `E`, deci propoziția este mereu adevărată.

9 Demonstrați corectitudinea sortării prin selecție folosind invarianți la ciclare.
R. Fie algoritmul de sortare prin selecție de mai jos:

for $i = 1 \to n$ **do**
  $iMin = i$
  for $j = i + 1 \to n$ **do**
    if $A[j] < A[iMin]$ **then**
5:    $iMin = j$
    end if
  end for

$swap(A[i], A[iMin])$
end for

Considerăm invariantul la ciclare $I = A[1..i-1]$ sortat. În starea inițială $P : i = 1$ înainte de începutul buclei iar în starea finală, $Q : i = n + 1$. Condiția care ține în timpul buclei *for* este $C : i \leq n$.

1. *inițializare:* $P \Rightarrow I$, pentru $i = 1$, vectorul nu este sortat;

2. *ciclu:* $I : A[1..i-1]$ sortat $\wedge\, C : i \leq n$
   corpul: ..., $swap(A[i], A[iMin])$, $i = i+1$, interschimb minimul secvenței $A[i..n]$ cu $A[i-1]$, rezultând că $A[1..i]$ devine sortat. Adică $I : A[1..(i+1)-1]$ ține deoarece și i este incrementat.

3. *terminare:* $I : A[1..i-1]$ sortat $\wedge \neg(C : i \leq n) \Rightarrow I \wedge Q$. Atunci, rezultă că $A[1..n+1-1]$ este sortat, deci algoritmul este corect.