# Teme Divide et Impera

1. Dându-se `N` numere întregi sub forma unei secvenţe de numere strict crescătoare, care se continuă cu o secvenţă de întregi strict descrescătoare, se doreşte determinarea punctului din întregul şir înaintea căruia toate elementele sunt strict crescătoare, şi după care, toate elementele sunt strict descrescătoare.

   Considerăm evident faptul că acest punct nu există dacă cele `N` numere sunt dispuse într-un şir fie doar strict crescător, fie doar strict descrescător.

   **Date de intrare**: Fișierul `secventa.in` care conține pe primul rând numărul de elemente N, iar pe al doilea rând conține secvența de numere întregi, separate prin spații.
   **Date de ieșire**: Fișierul `secventa.out` care conține pe primul rând poziția în șirul citit unde se găsește punctul căutat.

2. Mini "motor de căutare": avem un tablou de documente și vrem să găsim toate documentele care conțin un anumit termen. Fiecare document este format dintr-un tablou de termeni. Un termen este un șir de caractere (poate conține spații). Facem comparațiile necesare cu `strcmp`.
   **Date de intrare:**
   - Fișierul `documente.txt` conține pe fiecare rând calea către un fișier.
   - Fișierele text de la căile specificate, care conțin pe fiecare rând câte un termen prezent în document (maxim 30 de caractere). Indicație: folosiți funcția `fgets`.

   **Date de ieșire:** Fișierul `index.out` care conține pe primul rând termenul căutat și pe rândurile următoare căile fișierelor în care a fost găsit termenul.

3. Să se aproximeze radical gradul `n` folosind metoda divide et impera, unde `n` este un număr natural. Nu se va folosi nicio funcție pentru calculul radicalului implementată într-o bibliotecă de funcții matematice (e.g. `math.h`). Se acceptă aproximarea cu o marjă de eroare $\varepsilon$, astfel încât dacă soluția reală este $x$, și soluția aproximată este $x_{apr}$, atunci $x^n_{apr} \in (x^n - \varepsilon, x^n + \varepsilon)$.
   **Date de intrare**: Fișierul `radicaln.in` conține pe primul rând numărul natural `n`, pe al doilea rând numărul real pozitiv $\varepsilon$, și pe al treilea rând numărul natural `m` din care se va calcula radicalul.
   **Date de ieșire**: Fișierul `radicaln.out` conține pe primul rând soluția aproximată.

4. Se dă un arbore binar prin parcurgerea în preordine în care apar şi valorile null (semnificând lipsa unui fiu). Să se construiască în memorie arborele (alocat dinamic și eliberat din memorie corespunzător) şi să se verifice dacă este arbore binar de căutare cu complexitate O(n), unde n=numărul de vârfuri.

   Indicații: se poate reprezenta fiecare nod din arbore definind un tip de date de tip `struct` conținând un câmp unde este stocată valoarea din nod și câte un câmp pentru a stoca fiecare fiu (folosind pointeri către nodurile fii).

   Parcurgerea în preordine înseamnă că întâi se parcurge întâi rădăcina, apoi subarborele stâng și apoi subarborele drept.

   Un arbore binar de căutare are proprietatea că toate elementele din subarborele stâng sunt mai mici decât rădăcina și toate elementele din subarborele drept sunt mai mari decât rădăcina. Această regulă se aplică recursiv.

| **Date.in** | **Date.out** |
|---|---|
| 4 1 null 3 null null 7 6 null null null | da |

5. Se dau `n` numere naturale. Să se afișeze al `k`-ulea lea cel mai mic element din șir.
   **Date de intrare**: fisierul `date.in` contine:
   - pe prima linie un număr natural `n` și un număr natural `k` și
   - pe a doua linie, un șir de n numere naturale, separate prin câte un spațiu.

   **Date de ieșire:** fișierul `date.out` conține valoarea cerută.

6. Maria locuiește într-un bloc cu `n` etaje numerotate de la `0` la `n`. Ea vrea să afle etajul maxim de la care poate arunca o sticlă fără ca aceasta să se spargă. Are la dispoziție `2` sticle. Folosind metoda divide et impera afișați etajele de la care trebuie să arunce sticla succesiv ca să afle răspunsul, minimizând numărul total de aruncări.
   **Date de intrare:** Fișierul `etaje.in` în care pe primul rând se găsește numărul de etaje n.
   **Date de ieșire**: Fișierul `etaje.out` în care se scriu în ordine etajele de la care trebuie aruncată sticla pentru a afla răspunsul; pe fiecare rând se scrie câte un etaj.
   **Indicații**: Dacă aveam o singură sticlă, singura strategie posibilă era să arunc sticla succesiv începând de la etajul 1 și urcând succesiv pană se sparge. Dacă am 2 sticle, o variantă optimă combină strategia specificată anterior cu căutarea binară.

7. Să se calculeze suma a n numere în virgulă mobilă prin metoda pairwise summation și să se compare eroarea cu suma calculată iterativ.

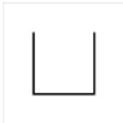
   Metoda pairwise summation este similară cu suma unui vector prezentată în curs, dar pentru problema direct rezolvabilă se rezolvă iterativ atunci când avem mai puțin de m elemente. Experimentați cu mai multe valori ale lui m.

   **Date de intrare:** fișierul suma.in are pe prima linie numărul n și pragul m, iar pe a doua linie cele n numere reale.
   **Date de ieșire:** suma numerelor calculată prin cele două metode.
   Resurse: Pairwise summation - Wikipedia

8. Considerăm un plan euclidian ce conţine n puncte date prin coordonatele lor. Distanţa euclidiană dintre două puncte $A(x_1, y_1)$ şi $B(x_2, y_2)$ se calculează conform formulei: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
   Să se determine distanţa dintre cele mai apropiate două puncte.
   **Date de intrare:**
   Fişierul de intrare puncte.in conţine pe prima linie un număr n cu semnificaţia din enunţ. Pe următoarele n linii se vor afla câte două numere $x_i$ şi $y_i$, separate printr-un spaţiu, semnificând coordonatele celui de al i-lea punct. Exemple aici.
   **Date de ieşire:**
   În fişierul de ieşire puncte.out se va afişa distanţa dintre cele mai apropiate două puncte din planul euclidian.
   **Indicații și exemple**: https://www.infoarena.ro/problema/cmap

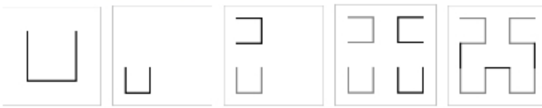
9. Pentru a înmulți două numere de n cifre, metoda naivă înmulțește fiecare cifră a multiplicatorului cu fiecare cifră a multiplicantului. În 1960, Karatsuba a observat că următorul calcul naiv al unui produs:
   $$(a \times 10^k + b)(c \times 10^k + d) = ac \times 10^{2k} + (ad + bc) \times 10^k + bd$$
   pare să necesite cele patru produse `ac`, `ad`, `bc` și `bd`, de fapt se poate face numai cu cele trei produse `ac`, `bd` și `( a - b ) ( c - d )` prin gruparea calculelor în următoarea formă:
   $$(a \times 10^k + b)(c \times 10^k + d) = ac \times 10^{2k} + (ac + bd - (a - b)(c - d)) \times 10^k + bd$$
   Pentru numere mari, metoda poate fi aplicată recursiv pentru calculele `ac`, `bd` și `( a - b ) ( c - d )` împărțind din nou `a, b, c` și `d` în jumătate și așa mai departe.
   Calculați produsul a două numere folosind metoda lui Karatsuba.
   **Date de intrare**: un fișier `date.in` care conține pe primele două rânduri cele două numere
   **Date de ieșire**: un fișier `date.out` care conține produsul.

10. Numim curba de ordin Hilbert de ordinul K curba curba realizată după următoarele reguli ce trece prin fiecare nod al unei grile de $2^K*2^K$ noduri și trece prin noduri vecine ale grilei.
Curba Hilbert de ordinul 1 este o curbă simplă:

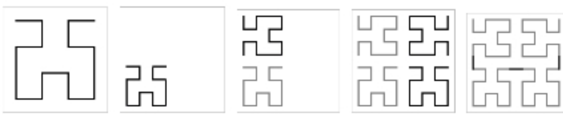


Descrise în următoarele imagini sunt trecerile de la o curbă de ordin x la o curbă de ordin x+1:

`Ordin 1 -> Ordin 2`



`Ordin 2 -> Ordin 3`



`Ordin 3 -> Ordin 4`

Să se calculeze în câți pași se ajunge la coordonatele `(x,y)` dacă punctele din pătrat sunt parcurse în ordinea dată de curba Hilbert de ordin K.
**Date de intrare:** Fișierul `fractal.in` care conține pe primele rânduri numerele K, x și y, unde K este ordinul unei curbe, iar x și y sunt coordonate întregi în interiorul unui pătrat de dimensiune $2^K*2^K$.
**Date de ieșire**: Fișierul de ieșire `fractal.out` în care se va scrie numărul de pași în care se ajunge la `(x,y)`.