# Structuri de date

Lect. Dr. Marius Dumitran

# Organizatorice

- Notare
- Laboratoare / laboranți
- Curs live

# Notare

- 50% laborator
  - **Nota minim 5!!**
- 20% seminar
  - Prezență, activitate și teme
  - Trebuie să aveți camera pornită pentru a fi luată în calcul prezența / activitatea
- 30% examen
  - Examen oral (dacă nu se va stabili că examenele se dau live)
  - **Nota minim 5!!**
- 10% Kahoot
  - Vom face teste la sfârșitul fiecărui curs (azi vom avea exemplu)
  - Se primește bonus pentru suma punctajelor pe tot semestrul dar și pentru rezultate bune la hakooturi individuale.

# Notare

- 50% laborator
  - **Nota minim 5!!**
  - 3 teme
    - Sortari 10p
      - Deadline 14 martie pentru cei cu lab în SI
    - Structura de date complexa 10p
    - Mix&Match 30p
  - Nota laborator + bonus maxim 1p de la laborant (bonusul îl pot primi doar cei care au punctaj din teme

# Laboratoare / laboranți

- La multe grupe veți avea laboranți din industrie, care lucrează la:
  - Google
  - Adobe
  - Microsoft
  - Startup-uri
- Nu toți au experienta la predat dar au experiență în industrie așa ca profitați de ocazie:
- Întrebați de toate, nu doar de SD
  - Cum se codează / lucrează într-o firmă
  - Interviuri
  - Tehnologii
  - Sfaturi de carieră

# Curs live

- Luni facem toți cursul online!
- Joi ? Marți ? voi încerca să predau cursul din facultate
  - Cursul va fi filmat live și voi încerca să răspund la întrebările care vin din online, dar și a celor din sală (dacă există studenți care vor dori să vină)
- Materia va fi aceeași!
- Practic, voi încerca să predau în 2 moduri aceleași lucruri

# Curs live

- Marți ? voi încerca să predau cursul din facultate
  - Incepem pe 23 sau pe 30 februarie.
  - Cursul va fi filmat live și voi încerca să răspund la întrebările care vin din online, dar și a celor din sală (dacă există studenți care vor dori să vină)
- Voluntari ? Ideal ar fi să rămână și la Programare Competitivă
  - Radu Filipescu & Teodor Mititelu

# Overview al materiei

- Curs 1-2 sortari/cautare binara
  - count sort, radix sort, quick sort, merge sort
- Curs 3 Vectori/Liste inlantuite
  - Cozi
  - Stive
  - Deque
- Curs 4 Heapuri
- Curs 5 Heapuri binomiale - fibonacci
- Curs 6 Huffman
- Curs 7 Arbori binari de căutare
- Curs 8 AVL / Red black
- Curs 9 Skil Lists / treaps
- Curs 10 Arbori de intervale
- Curs 11 RMQ & LCA & LA
- Curs 12-13 Hashuri
- Curs 14  Tries / Suffix trees ?

# Overview al materiei

- Curs 1-2 sortari/cautare binara
  - count sort, radix sort, quick sort, merge sort
- Curs 3 Vectori/Liste inlantuite
  - Cozi
  - Stive
  - Deque
- Curs 4 Heapuri
- Curs 5 Heapuri binomiale - fibonacci
- Curs 6 Huffman
- Curs 7 Arbori binari de cautare
- Curs 8 AVL / Red black
- Curs 9 Skil Lists / treaps
- Curs 10 Arbori de intervale
- Curs 11 RMQ & LCA & LA
- Curs 12-13 Hashuri
- Curs 14  Tries / Suffix trees ?

Propuneri ?

- Alocatori ... mai curand la OOP

# Algoritmi de sortare

Ce algoritmi de sortare cunoașteți?

# Algoritmi de sortare

Ce algoritmi de sortare cunoașteți?

- Bubble O(n^2)
- Merge O(nlogn)
- Interschimbare O(n^2)
- Radix
- Quick O(nlogn)?
- Heap O(nlogn)
- Bucket Sort
- Count Sort
- Bogo Sort O(n!*n)
- Gravity Sort O(n^2)
- Selection Sort O(n^2)
- Insert sort O(n^2)
- Shell Sort O(n sqrt n)
- Intro Sort O(nlogn)
- Tim Sort O(nlogn)

Putem grupa dupa:

- Complexitate
- Complexitate spatiu
- Stabil
- Dacă se bazează pe comparatii sau nu.

# Algoritmi de sortare stabili

- Un algoritm de sortare este stabil dacă pastreaza ordinea elementelor egale.
- 5 5 5 -> 5 5 5 (sortare stabila)
- 5 5 5 -> 5 5 5 (sortare instabilă)

Atenție și unii algoritmi instabili pot sorta stabil uneori, algoritmii stabili garantează asta pentru orice input.

Pentru numere naturale nu este important dar cand sortam altfel de obiecte acest lucru poate deveni important.

# Algoritmi de sortare

**Clasificare**

| Elementari | Prin comparație | Prin numărare |
|---|---|---|
| Insertion sort → $O(n^2)$ | Quick sort → O(n logn) | Bucket sort |
| Selection sort → $O(n^2)$ | Merge sort → O(n logn) | Counting sort |
| Bubble sort → $O(n^2)$ | Heap sort → O(n logn) | Radix sort |
| | Intro sort → O(n logn) | |

**Tabel cu sortări:**
https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms

# Sortare prin numarare/Counting Sort

- Algoritm de sortare a numerelor întregi mici
- Presupunem că vectorul de sortat **v** conține **n** elemente din mulțimea {1, ..., max}

**IDEE:**

- Creem un vector de frecvență **fr**
- Numărăm aparițiile fiecărui element din **v**
- Modificăm vectorul **fr** a.î.
  - `fr[i] = numărul de elemente cu valoare = i`
- `La final iteram prin vectorul fr[i] i afisam i de fr[i] pentru toate numerele de la 1 la max.`

# Sortare prin numarare/Counting Sort

Exemplu: sortam note

| note | 5 | 3 | 2 | 9 | 4 | 5 | 1 | 7 | 10 | 3 | 9 | 2 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| nota | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | | |
| fr | 1 | 2 | 2 | 1 | 3 | | 1 | | 2 | 1 | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

# Sortare prin numarare/Counting Sort

Exemplu: sortam note

| note | 5 | 3 | 2 | 9 | 4 | 5 | 1 | 7 | 10 | 3 | 9 | 2 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| nota | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | | |
| fr | 1 | 2 | 2 | 1 | 3 | 0 | 1 | 0 | 2 | 1 | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |

# Sortare prin numarare/Counting Sort

Exemplu: sortam note

| nota | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fr | 1 | 2 | 2 | 1 | 3 | 0 | 1 | | 2 | 1 | | | |
| | | | | | | | | | | | | | |
| solutie | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 7 | 9 | 9 | 10 |

# Sortare prin numarare/Counting Sort

Exemplu: sortam note

| note | 5 | 3 | 2 | 9 | 4 | 5 | 1 | 7 | 10 | 3 | 9 | 2 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| | | | | | | | | | | | | | |
| solutie | 1 | 2 | 2 | 3 | 3 | 4 | 5 | 5 | 5 | 7 | 9 | 9 | 10 |

# Cod

```
//Pasul 1: Crestem frecventa fiecărui element din vector:
 for (int i = 1; i <=n; ++i)  // O(n)
       fr[note[i]]++;
// Pasul 2, afisam fiecare element de atatea ori cat apare în vectorul de frecenta
// O(maxn * n)
// O(maxn + n)
  for (int i = 0; i <= maxn; ++i) {   // pana la maxn
       for (int j = 1; j <= fr[i]; ++j) {  // worst case se duce pana la n   // for-ul va face n + maxn
          cout<< i<< " ";  // Afisam de fix n ori
       }
    }
```

Complexitate ?  Spatiu ?  Timp ?

# Counting Sort

Complexitate

- Timp:
  - O(n + max)

- Spațiu:
  - O(max)

# Counting Sort

Vizualizare:

https://visualgo.net/bn/sorting

# Counting Sort

Ce ne facem dacă avem de sortat numere mari...

- Pana la 10^6 ?
- Pana la 10^18 ?
- Numere care nu sunt întregi ?

# Counting Sort

Ce ne facem dacă avem de sortat numere mari...

- Pana la 10^6 ?
  - Depinde de N dar **Count Sort** poate fi cea mai buna opțiune...
- Pana la 10^18 ?
  - Nu mai putem folosi Count Sort putem folosi în schimb **Radix Sort**
- Numere care nu sunt întregi ?
  - Mai greu și cu Radix Sort (nu e imposibil, dacă sunt doar 1-2 zecimale putem înmulți cu 10, 100) ... altfel putem folosi **Bucket Sort**

# Bucket Sort

- Elementele vectorului sunt distribuite în bucket-uri după anumite criterii
- Bucket-urile sunt reprezentate de elemente ale unui vector de liste înlănțuite
- Fiecare bucket conține elemente care îndeplinesc aceleași condiții

**IDEE:**

- Fie **v** vectorul de sortat și **b** vectorul de buckets
- Se inițializează vectorul auxiliar cu liste (buckets) goale
- Iterăm prin **v** și adăugăm fiecare element în bucket-ul corespunzător
- Sortam fiecare bucket (discutam cum)
- Iterăm prin fiecare bucket, de la primul la ultimul, adăugând elementele înapoi în **v**

# Bucket Sort

Vizualizare:

https://www.cs.usfca.edu/~galles/visualization/BucketSort.html

# Bucket Sort

Cum adăugăm elementele în bucket-ul corespunzător?

-

# Kahoot

https://create.kahoot.it/creator/2281f10b-f400-43fe-981c-85377fd66c12

# Final