# Structuri de Date
# Seminar 1

**Reminder!**

*$N^2$: Bubble Sort, Insertion Sort, Selection Sort, etc

***N log N**: Heap Sort, Merge Sort, Tim Sort, etc.

*$N+Max$: Counting Sort

## Problema 1:

***a)*** *Se dau N numere sortate. Câte perechi de numere au suma S?*

### Idee naivă:

Parcurgem toate perechile posibile (*for* în *for*) => **O($N^2$)**

### Idee:

Ne punem întrebarea *„Cum facem să ne folosim de faptul că nr. sunt sortate?"*

Parcurgem numerele, și pentru fiecare număr *X* folosim căutare binară să descoperim aparițiile lui *(S-X)*. Căutare binară = log N; o facem pentru fiecare element din cele N => **O(N log N)**

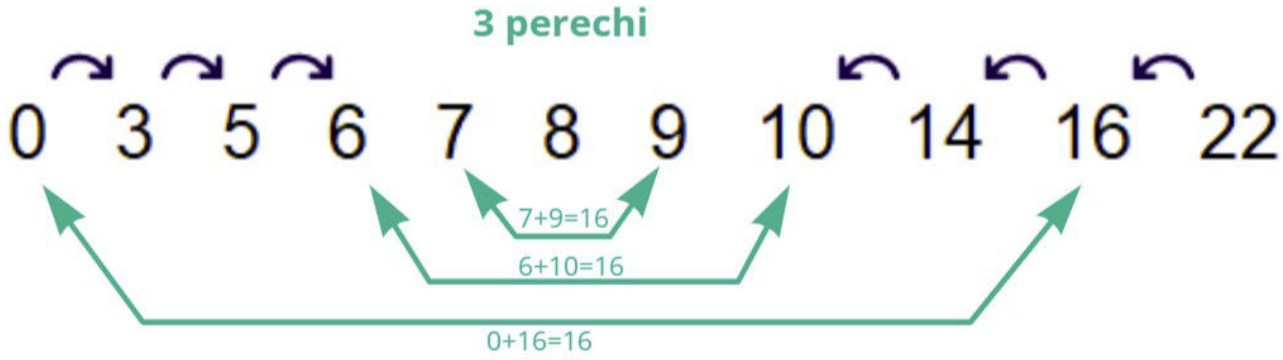
### Idee îmbunătățită:

Folosim doi pointeri *i* (pornește din stânga) și *j* (pornește din dreapta); comparăm numerele aflate la indicii i și j, și scădem j sau creștem i corespunzător:

- a[i] + a[j] < S => creștem i
- a[i] + a[j] > S => scădem j

=> **O(N)**

**Exemplu**: N = 11, S = 16




***b)*** *Cum procedăm dacă numerele nu sunt sortate?*

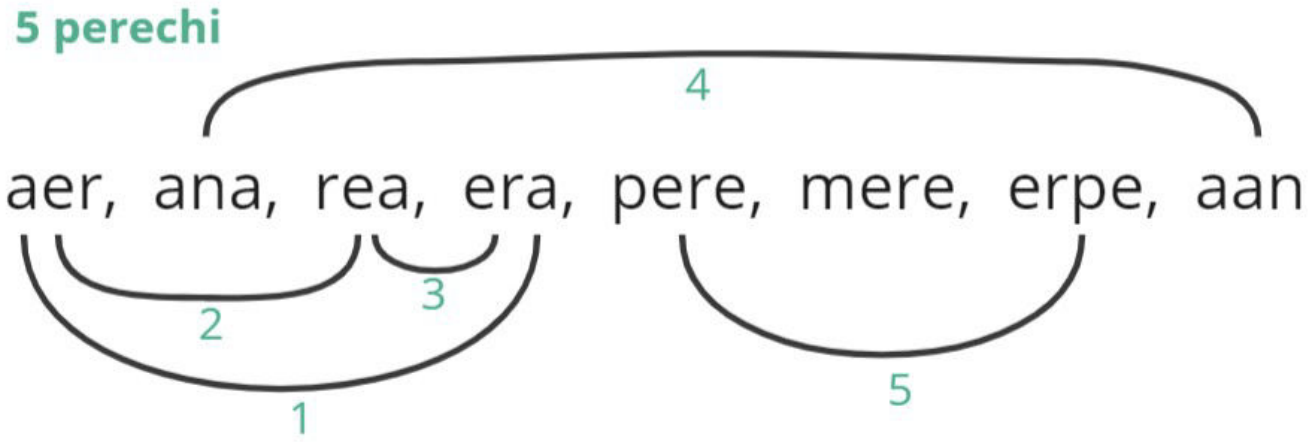
- ➔ *Cu două for-uri =>* $O(N^2)$
- ➔ *Sortăm și facem ca la a) =>* $O(N \log N)$
- ➔ *Vector de frecvențe =>* $O(N+Max)$
- ➔ *Hash Tables =>* $O(N)$

**Pentru acasă:** *Se dă un vector sortat și un număr x. De câte ori apare x în vector?*

## Problema 2:

*Se dau N cuvinte. Câte perechi de anagrame sunt?*

**Exemplu**: N = 8



Considerăm L = lungimea maximă a unui cuvânt.

### Pasul 1:

Sortăm fiecare cuvânt crescător.

- O(N*L*log L) pentru sortări bazate pe comparare
- **O(N*L)** pentru counting sort (maximul în acest caz este 26, o constantă, deci complexitatea sortării unui cuvânt va fi O(L) în loc de O(L+Max)).

### Pasul 2:

Ținem un dicționar de frecvențe să reținem pentru fiecare cuvânt (sortat) de câte ori apare. Să hash-uim un cuvânt durează O(L), așadar pentru N cuvinte va dura **O(N*L)**.

### Complexitate finală:

O(N*L + N*L) = O(2*N*L) = **O(N*L)**

## Problema 3:

*Se dă un vector care era sortat și a fost rotit circular. Găsiți valoarea maximă din vector.*

**Exemplu**:


### Idee naivă:

Parcurgem toate elementele și găsim maximul. => **O(N)**

**Idee :**

Folosim o căutare binară modificată, astfel încât să găsim elementul pentru care vecinul din dreapta lui este mai mic decât el (înseamnă că acolo a avut loc rotația).
=> O(log N)

## Căutare binară îmbunătățită (pe biți)

```cpp
int binary_search(const vector<int>& v, int target)
{
    int step = 1 << 20;  // 2^20

    int position = 0;
    while (step) {

        // dacă această condiție este evaluată la True, înseamnă că
        // valoarea căutată este mai în dreapta decât position+step,
        // așadar creștem poziția la care căutăm
        if (position + step < v.size()
            && v[position + step] <= target)
            position += step;

        step >>= 1;  // step = step / 2;
    }

    // dacă am găsit => returnează poziția, altfel returnează -1
    return v[position] == target ? position : -1;
}
```

**Pentru acasă:** *Se dă un vector cu N elemente. Câte inversiuni are acesta?*

Inversiune = pereche **(i, j)** unde **i < j && v[i] > v[j]**