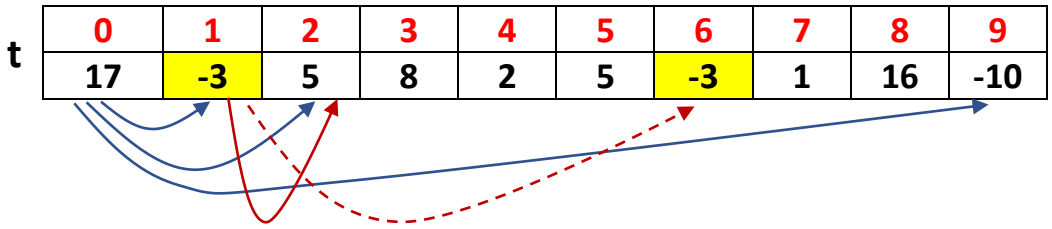
CURS 08 – FP
Tablouri unidimensionale (vectori)

1. Să se verifice dacă un tablou unidimensional are elemente distincte sau nu (este *mulțime* sau nu).



Idee de rezolvare:

- presupunem că tabloul ar avea elemente distincte
- comparăm fiecare element `t[i]`, unde `i ∈ {0, 1,...,n-2}`, cu toate elementele `t[j]` aflate în dreapta sa (unde `j ∈ {i+1,...,n-1}`), iar în cazul în care `t[i] == t[j]` presupunerea făcută devine falsă

```
#include <stdio.h>

int main()
{
    int t[100], n, i, j, dist;

    //citim elementele tabloului t
    printf("n = ");
    scanf("%d", &n);

    for(i = 0; i < n; i++)
    {
        printf("t[%d] = ", i);
        scanf("%d", &t[i]);
    }

    //presupunem ca tabloul are elemente distincte
    dist = 1;
    //comparam fiecare element cu toate cele aflate
    //in dreapta sa
    for(i = 0; i < n-1; i++)
        for(j = i+1; j < n; j++)
            //daca doua elemente sunt egale, atunci
            //presupunerea facuta devine falsa
            if(t[i] == t[j])
```

```
                dist = 0;

    if(dist == 1)
        printf("\nTabloul are elemente distincte!\n");
    else
        printf("\nTabloul nu are elemente distincte!\n");

    return 0;
}
```

Numărul de comparări efectuate:

- t[0] se compară cu t[1],…,t[n-1] => n-1 comparări
- t[1] se compară cu t[2],…,t[n-1] => n-2 comparări
- .....
- t[n-2] se compară cu t[n-1] => 1 comparare

Total: $(n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2}$ comparări

Variantă de optimizare:

Întrerupem cele două instrucțiuni for în momentul în care întâlnim două elemente t[i] și t[j] egale:

Varianta 1:

Introducem condiția `dist==1` (nu am găsit încă două elemente egale) în condițiile celor două instrucțiuni for:

```
//presupunem ca tabloul are elemente distincte
dist = 1;
//comparam fiecare element cu toate cele aflate
//in dreapta sa
for(i = 0; i < n-1 && dist == 1; i++)
    for(j = i+1; j < n && dist == 1; j++)
        //daca doua elemente sunt egale, atunci
        //presupunerea facuta devine falsa
        if(t[i] == t[j])
            dist = 0;
```

Varianta 2:

Folosim două instrucțiuni `break` pentru a întrerupe instrucțiunile `for`:

```
//comparam fiecare element cu toate cele aflate
//in dreapta sa
for(i = 0; i < n-1; i++)
{
    for(j = i+1; j < n; j++)
```

```
            //daca doua elemente sunt egale, atunci
            //nu are rost sa mai continuam (nu este multime)
            if(t[i] == t[j])
                break;

        //dacă instructiunea for dupa j a fost intrerupta
        //fortat (adica j < n), atunci intrerupem fortat
        //si instructiunea for dupa i
        if(j < n)
            break;
    }

    if(i < n-1)
        printf("\nTabloul nu are elemente distincte!\n");
    else
        printf("\nTabloul are elemente distincte!\n");
```

Varianta 3:

Întrerupem complet programul:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int t[100], n, i, j;

    //citim elementele tabloului t
    printf("n = ");
    scanf("%d", &n);

    for(i = 0; i < n; i++)
    {
        printf("t[%d] = ", i);
        scanf("%d", &t[i]);
    }

    //comparam fiecare element cu toate cele aflate
    //in dreapta sa
    for(i = 0; i < n-1; i++)
    {
        for(j = i+1; j < n; j++)
            //daca doua elemente sunt egale, atunci
            //nu are rost sa mai continuam (nu este multime)
            if(t[i] == t[j])
            {
                printf("\nTabloul nu are elemente distincte!\n");
                //intrerupem fortat programul, adica functia main
                return 0;
```

```
                //varianta: exit(0);
            }
    }

    printf("\nTabloul are elemente distincte!\n");

    return 0;
}
```

2. Să se construiască mulțimea asociată unui tablou de numere întregi (să se extragă valorile distincte).

Problemă de "exprimare":

v = (2, 1, 5, 2, 3, 2, 1, 4, 5)

Valorile distincte din tabloul v:

- toate valorile, dar o singură dată fiecare (mulțime) => d = (2, 1, 5, 3, 4)
- valorile care nu au duplicate => d = (3, 4) (restul valorilor au duplicate)

	~~i=0~~	~~i=1~~	~~i=2~~	~~i=3~~	~~i=4~~	~~i=5~~	~~i=6~~	~~i=7~~	~~i=8~~	~~i=9~~
v	0	1	2	3	4	5	6	7	8	9
	17	-3	5	17	17	5	-3	1	16	-3

d = un tablou care va conține valorile distincte din tabloul **t**
m = numărul valorilor distincte extrase în tabloul **d** până în momentul respectiv

	~~m=0~~	~~m=1~~	~~m=2~~	~~m=3~~	~~m=4~~	m=5				
d	0	1	2	3	4	5	6	7	8	9
	17	-3	5	1	16					

Idee de rezolvare:

- căutăm fiecare element `t[i]`, unde `i ∈ {0, 1,...,n-1}` în tabloul d, deci îl comparăm cu toate elementele `d[j]`, unde `j ∈ {0,...,m-1}`, iar în cazul în care nu apare, îl adăugăm la sfârșitul tabloului d

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    //v = tabloul initial cu n elemente
    //d = tabloul format din cele m <= n
```

```
    //valori distincte din tabloul v
    int v[100], d[100], n, m, i, j;

    //citim elementele tabloului t
    printf("n = ");
    scanf("%d", &n);

    for(i = 0; i < n; i++)
    {
        printf("v[%d] = ", i);
        scanf("%d", &v[i]);
    }

    //initial, nu am gasit nicio valoare distincta
    m = 0;
    //consideram fiecare valoare din tabloul v
    for(i = 0; i < n; i++)
    {
        //cautam t[i] in tabloul d, intre pozitiile 0 si m-1
        for(j = 0; j < m; j++)
            if(v[i] == d[j])
                //am gasit t[i] in tabloul d
                break;

        //daca t[i] nu apare in tabloul d,
        //atunci il adaugam la sfarsitul sau
        if(j == m)
        {
            d[m] = v[i];
            m++;
        }
    }

    printf("\nValorile distincte din tabloul t:\n");
    for(j = 0; j < m; j++)
        printf("%d ", d[j]);

    return 0;
}
```

Dacă trebuie să modificăm tabloul v, atunci folosim algoritmul de mai sus, după care copiem tabloul d în tabloul v:

```
.....
//copiem tabloul d in tabloul initial v
n = m;
for(i = 0; i < n; i++)
  v[i] = d[i];
```

3. Să se afișeze frecvența fiecărui element distinct dintr-un tablou unidimensional. De exemplu, pentru v = (2, 1, 5, 2, 3, 2, 1, 4, 5) și n = 9 se va afișa:

```
Valoarea 2 apare de 3 ori
Valoarea 1 apare de 2 ori
Valoarea 5 apare de 2 ori
Valoarea 3 apare de 1 ori
Valoarea 4 apare de 1 ori
```

Idee de rezolvare:

Extragem în tabloul d = (2, 1, 5, 3, 4) cele m = 5 valori distincte din tabloul v, după care parcurg tabloul d element cu element și număr de câte ori apare elementul curent în tabloul inițial v!

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int v[100], n, i, m, d[100], j, contor;

    printf("Introduceti numarul de elemente: ");
    scanf("%d", &n);

    printf("\nElementele vectorului:\n");

    for(i = 0; i < n; i++)
    {
        printf("v[%d] = ", i);
        scanf("%d", &v[i]);
    }

    m = 0;
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < m; j++)
            if(v[i] == d[j])
                break;
        if(j == m)
        {
            d[j] = v[i];
            m++;
        }
    }
```

```
    for(i = 0; i < m; i++)
    {
        contor = 0;
        for (j = 0; j < n; j++)
            if (d[i] ==v[j])
                contor++;
        printf("Valoarea %d apare de %d ori\n", d[i], contor);
    }

    return 0;
}
```

METODA SORTĂRII PRIN SELECȚIE

Ideea generală:
Un tablou v cu n elemente este sortat crescător dacă și numai dacă $v[i] = \min\{v[i], v[i+1], \dots, v[n-1]\}$, pentru orice i cuprins între 0 și $n-2$!

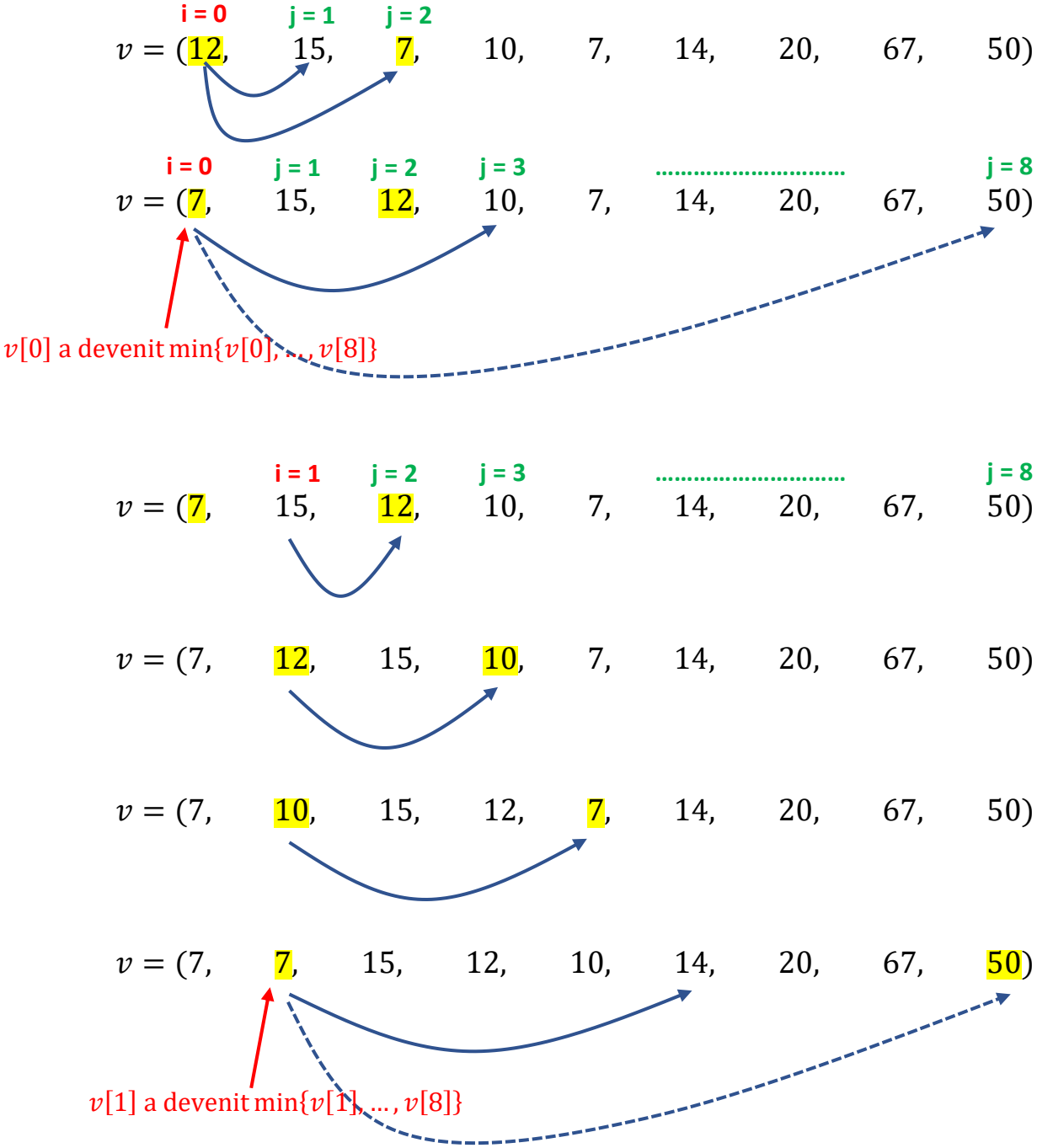
v = (7, 10, 10, 15, 75, 101, 202, 202)

v[0] = min {v[0], ..., v[n-1]} => v[0] <= v[1], v[0] <= v[2], ..., v[0] <= v[n-1]
v[1] = min {v[1], ..., v[n-1]} => v[1] <= v[2], v[1] <= v[3], ..., v[1] <= v[n-1]
v[2] = min {v[2], ..., v[n-1]} => v[2] <= v[3], v[2] <= v[4], ..., v[2] <= v[n-1]
.....
v[n-3] = min {v[n-3], v[n-2], v[n-1]} => v[n-3] <= v[n-2], v[n-3] <= v[n-1]
v[n-2] = min {v[n-2], v[n-1]} => v[n-2] <= v[n-1]

v[0] <= v[1] <= v[2] <= ... <= v[n-2] <= v[n-1] => tabloul v este sortat crescător!

Practic, orice element $v[i]$ trebuie să fie mai mic decât toate elementele aflate în dreapta sa, deci îl voi compara cu toate elementele $v[j]$ din dreapta sa și, în cazul în care găsesc unul mai mic, interschimbăm $v[i]$ cu $v[j]$.

Exemplu:



Se reia procedeul de mai sus pentru i = 1, i = 2, ..., i = n-2!

```c
#include <stdio.h>

int main()
{
    //t = tabloul initial cu n elemente
    int t[100], n, i, j, aux;

    //citim elementele tabloului t
    printf("n = ");
    scanf("%d", &n);

    for(i = 0; i < n; i++)
    {
        printf("t[%d] = ", i);
        scanf("%d", &t[i]);
    }

    //compar fiecare element t[i] cu toate elementele t[j]
    //aflate in dreapta sa
    for(i = 0; i < n-1; i++)
        for(j = i+1; j < n; j++)
            //daca elementul t[j], din dreapta lui t[i],
            //este mai mic decat t[i], atunci interschimb
            //t[i] cu t[j]
            if(t[j] < t[i])
            {
                aux = t[i];
                t[i] = t[j];
                t[j] = aux;
            }

    printf("\nTabloul sortat crescator:\n");
    for(i = 0; i < n; i++)
        printf("%d ", t[i]);

    return 0;
}
```