Algoritmi si Structuri de Date

- seminar II -

Sortari.

Sortarea este e problema clasica, intens studiata, care cere aranjarea elementelor (numere naturale, reale, litere, etc.) dintr-o lista intr-o anumita ordine (crescatoare, descrescatoare, lexicografica, etc.).

Exista mai mai multe tipuri de sortari:

- Sortari bazate pe comparatie (unde nu se poate obtine un rezultat mai bun decat $\Omega(n \log n)$ – best case)
- Sortari care nu sunt bazate pe comparatie (Radix sort)
- Sortari recursive
- Sortari iterative

Vom studia urmatoarele sortari:

- Bubble sort (cat timp sirul nu este sortat, interschimbam elementele adiacente)
- Insertion sort (avem un sir sortat, vrem sa adaugam inca un element)
- Selection sort (gasim elementul min/max si il interschimbam cu elementul curent)
- Merge sort (impartim sirul in doua parti egale, le sortam individual, le interclasam)
- Quicksort (alegem un pivot, mutam elementele mai mici in stanga, cele mai mari in dreapta, repetam recursiv pentru cele doua parti)
- Counting sort (numaram cate elemente de fiecare tip avem)
- Bucket sort (se impart elementele in “galeti”, se sorteaza individual fiecare)
- Radix sort (sortam numerele mai intai dupa prima cifra, apoi dupa a doua, etc.)

Site interactiv: https://visualgo.net/bn/sorting

Bubble sort

In romana: “Sortare prin metoda bulelor”

Algoritm in pseudocod:

Input: A = vector de numere intregi
Output: A = vector sortat
Algoritm:

```
def bubble_sort(A):
    n = len(A)          # obtinem numarul de elemente a lui A

    change = True       # folosim variabila change pentru a verifica
                        # daca a avut loc vreo interschimbare
    while change == True:    # cat timp au loc interschimbari
        change = False       # presupunem ca nu vor avea loc swap-uri
        for i in range(0, n - 1):     # parcurgem vectorul
            if A[i] > A[i + 1]:  # doua elemente adiacente asezate gresit
                A[i], A[i + 1] = A[i + 1], A[i]   #  facem swap
                change = True                     # modificam variabila
    return A
```

Exemplu:

0: 5 9 2 1 8 6 0 7 3
1: 5 2 9
5 2 1 9
5 2 1 8 9
5 2 1 8 6 9
5 2 1 8 6 0 9
5 2 1 8 6 0 7 9
5 2 1 8 6 0 7 3 **9**
2: 2 5
2 1 5 8
2 1 5 6 8
2 1 5 6 0 8
2 1 5 6 0 7 8
2 1 5 6 0 7 3 **8 9**
3: 1 2 5 0 6 3 **7 8 9**
4: 1 2 0 5 3 **6 7 8 9**
5: 1 0 2 3 **5 6 7 8 9**
6: 0 1 2 **3 5 6 7 8 9**
7: 0 1 **2 3 5 6 7 8 9**
(0 interschimbari)

Insertion sort

In romana: “Sortare prin insertie”

Presupunem ca avem primele k pozitii sortate crescator in vector si incercam sa inseram elementul k + 1 pe pozitia corespunatoare.

Algoritm in pseudocod:

Input: A = vector de numere intregi

Output: A = vector sortat

Algoritm:

```
def insertion_sort(A):
    n = len(A)

    for i in range(1, n):          # la inceput avem primul element “sortat”
        cp_i = i                   # salvam pozitia curenta in care ne afla
        while cp_i:                # incercam sa gasim pozitia corecta a lui A[i]
            if A[cp_i] > A[cp_i – 1]:          # comparam cu elem. vecin
                A[cp_i], A[cp_i – 1] = A[cp_i – 1], A[cp_i]
                cp_i -= 1
            else:                              # altfel, am gasit pozitia
                break                          # corecta si continuam
    return A
```

Exemplu:

0:	**5** 9 2 1 8 6 0 7 3	4:	**1 2 5 8 9** 6 0 7 3	8:	**0 1 2 3 5 6 8 7 9**
1:	**5 9** 2 1 8 6 0 7 3	5:	**1 2 5 6 8 9** 0 7 3		
2:	**2 5 9** 1 8 6 0 7 3	6:	**0 1 2 5 6 8 9** 7 3		
3:	**1 2 5 9** 8 6 0 7 3	7:	**0 1 2 5 6 8 7 9** 3		

Selection sort (selectia minimului)

In romana: “Sortare prin selectia minimului”

Algoritm in pseudocod:

Input: A = vector de numere intregi

Output: A = vector sortat

Algoritm:

```
def selection_sort(A):
    n = len(A)

    for i in range(0, n):
        poz_min = i          # cautam pozitia pe care se afla elem. minim
        elem_min = A[i]      # salvam si valoarea elementului minim

        for j in range(i, n):
            if A[j] < elem_min:
                poz_min = j
                elem_min = A[j]

        A[i], A[poz_min] = A[poz_min], A[i] # interschimbam elem.

    return A
```

Exemplu:

0: 5 9 2 1 8 6 *0* 7 3

1: **0** 9 2 *1* 8 6 5 7 3

2: **0 1** *2* 9 8 6 5 7 3

3. **0 1 2** 9 8 6 5 7 *3*

4: **0 1 2 3** 8 6 *5* 7 9

5: **0 1 2 3 5** *6* 8 7 9

6: **0 1 2 3 5 6** 8 *7* 9

7: **0 1 2 3 5 6 7** *8* 9

8: **0 1 2 3 5 6 7 8** *9*

9: **0 1 2 3 5 6 7 8 9**

Merge sort

In romana: “Sortare prin interclasare”

Vom folosi procedeul de interclasare pentru a sorta. Algoritmul de interclasare primeste doi vectori sortati si returneaza un singur vector **sortat** care contine toate elementele celor doi vectori initiali.

Exemplu: A = [1, 2, 5, 8, 9], B = [0, 3, 6, 7]. Dorim sa facem reuniunea celor doi vectori eficient (un algoritm ineficient este sa punem toate elementele laolalta si sa folosim un algoritm de mai sus pentru a le sorta – complexitatea fiind in cel mai rau caz O(n log n)).

Algoritmul de interclasare folosit aici va folosi doi indici i si j, unul pentru vectorul A si altul pentru vectorul B, pentru a sti care este elementul curent din fiecare vector, va compara cele doua elemente A[i] si B[j] si va decide ce element urmeaza in vectorul sortat.

Initial: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; i = 0; j = 0; C = []
1: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [] => C += [0], j++;
2: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0] => C += [1], i++;
3: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1] => C += [2], i++;
4: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1, 2] => C += [3], j++;
5: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1, 2, 3] => C += [5], i++;
6: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1, 2, 3, 5] => C += [6], j++;
7: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1, 2, 3, 5, 6] => C += [7], j++;
8: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1, 2, 3, 5, 6, 7] => C += [8], i++;
9: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1, 2, 3, 5, 6, 7, 8] => C += [9], i++;
10: A = [1, 2, 5, 8, 9]; B = [0, 3, 6, 7]; C = [0, 1, 2, 3, 5, 6, 7, 8, 9]

Observatie! Dupa ce terminam un vector, putem sa adaugam elementele celuilalt vector la final (pasul 8, 9, 10).

Algoritm in pseudocod:

```
Input:      A = vector de numere intregi
Output:     A = vector sortat
Algoritm:
    def merge_sort(A):
        n = len(A)

        if n <= 1:                  # daca avem un singur element in vector
            return A                # il returnam pentru ca este sortat

        A_stanga = merge_sort(A[:n / 2])    # sortam recursiv jumatatea stanga
        A_dreapta = merge_sort(A[n / 2:])   # si pe cea dreapta

        # avand doi vectori sortati (A_stanga, A_dreapta) ii putem interclasa
        A_rezultat = interclasare(A_stanga, A_dreapta)

        return A_rezultat
```

Exemplu:

```
0:                          5 9 2 1 8 6 0 7 3
1:          5 9 2 1 8                       6 0 7 3
2:     5 9 2          1 8                 6 0          7 3
3:  5 9     2       1     8              6     0       7     3
4:     2 5 9          1 8                 0 6          3 7
5:          1 2 5 8 9                       0 3 6 7
6:                          0 1 2 3 5 6 7 8 9
```

Quicksort

Partitionare:

1. Se alege un pivot (de ex. primul element sau un element aleator)
2. Se pleaca cu doi indici (unul de la stanga, altul de la dreapta) si se cauta elemente astfel:
   - din partea stanga se va cauta un element mai mare decat pivotul
   - din partea dreapta se va cauta un element mai mic decat pivotul
   (cautam deci elemente pozitionate gresit in raport cu pivotul ales)
3. Se interschimba elementele si se muta indicii
4. Ne oprim cand cei doi indici sunt egali.

Exemplu: 5 9 2 1 8 6 0 7 3

Aleg ca pivot elementul 6. Indicele stanga este marcat cu albastru, dreapta cu rosu.

5 9 2 1 8 **6** 0 7 3

Incrementez indicele stanga pana gasesc o valoare mai mare decat pivotul.

5 9 2 1 8 **6** 0 7 3

Decrementez indicele dreapta pana gasesc o valoare mai mica decat pivotul.

5 9 2 1 8 **6** 0 7 3

Interschimb cele doua elemente (doar elementele, indicii raman pe loc).

5 3 2 1 8 **6** 0 7 9

Se modifica pozitiile indicilor (stanga creste, dreapta descreste).

5 3 2 1 8 **6** 0 7 9

Cautam in continuare un element mai mare decat pivotul la stanga si unul mai mic decat pivotul in partea dreapta.

5 3 2 1 8 **6** 0 7 9

Interschimbam elementele.

5 3 2 1 0 **6** 8 7 9

Modificam pozitiile indicilor (stanga creste, dreapta descreste).

5 3 2 1 0 **6** 8 7 9

Indicii se suprapun, ceea ce inseamna ca am terminat partitionarea. In stanga pivotului se afla doar elemente mai mici decat acesta, iar in dreapta avem elemente mai mari.

Daca dorim sa sortam vectorul, vom apela recursiv acelasi algoritm pentru partea stanga si pentru cea dreapta.

Algoritm in pseudocod – Divide et Impera:

Input: A = vector de numere intregi
Output: A = vector sortat
Algoritm:

```
def partition(A, begin, end):                   # partitionam vectorul A
    pivot = A[begin]                            # alegem un pivot

    while True:
        while A[begin] < pivot:                 # cautam un element mai mare
            begin += 1                          # decat pivotul in stanga
        while A[end] > pivot:                   # cautam un element mai mic
            end -= 1                            # decat pivotul in dreapta

        if begin >= end:                        # daca s-au intercalat cei doi
            return end                          # indici, returnam poz. pivotului

        A[begin], A[end] = A[end], A[begin] # interschimbam valorile
        begin += 1
        end -= 1

    return begin                                # returnam pozitia pivotului

def quicksort(A, begin, end):
    if begin >= end:                            # daca primim un vector cu un
        return A                                # singur element, e sortat

    pozitie_pivot = partition(A, begin, end)    # partitionam vectorul
    quicksort (A, begin, pozitie_pivot – 1)     # sortam in partea stanga
    quicksort (A, pozitie_pivot + 1, end)       # sortam in partea dreapta

    return A                                    # returnam vectorul sortat
```

Algoritm in pseudocod – algoritm exemplu:

Input: A = vector de numere intregi

Output: A = vector sortat

Algoritm:

```
def quick_sort(A, left, right):
    pivot = A[left]                    # alegem ca pivot primul element

    begin = left                       # plecam cu doi indici begin stanga
    end = right                        # si end dreapta
    while begin <= end:                # cat timp nu se intercaleaza
        while A[begin] < pivot:            # cautam in stanga un element
            begin += 1                     # mai mare decat pivotul
        while A[end] > pivot:              # cautam in dreapta un element
            end -= 1                       # mai mic decat pivotul

        if begin <= end:               # daca am gasit
            A[begin], A[end] = A[end], A[begin] # facem swap
            begin += 1                 # incrementam idx stanga
            end -= 1                   # decrementam idx dreapta

    if left < end:          # sortam in partea stanga
        quick_sort(A, left, end)

    if begin < right:     # sortam in partea dreapta
        quick_sort(A, begin, right)

    return A                # dupa ce efectuam sortarile, A este sortat
```

Invarianti.

Un invariant este o proprietate care se pastreaza la fiecare iteratie a algoritmului.

- Bubble sort: dupa pasul k, ultimele/primele k elemente al sirului sunt sortate
- Insertion sort: dupa pasul k, primele k + 1 elemente sunt sortate (nu neaparat din sir)
- Selection sort: dupa pasul k, primele/ultimele k elemente ale sirului sunt sortate
- Quicksort: dupa pasul k, avand pivotul p, elementele mai mici decat p se afla in stanga pivotului iar cele mai mari decat p se afla in dreapta acestuia

Exercitii.

1. Se da vectorul: 2 5 13 11 7 10 19 20 - adevarat sau fals?
   - a rezultat in urma aplicarii a 2 pasi din selection sort – maxim
   - a rezultat in urma aplicarii a 3 pasi din selection sort – minim
   - a rezultat in urma aplicarii a 3 pasi din bubble sort
   - a rezultat in urma aplicarii a 2 pasi din bubble sort
   - a rezultat in urma aplicarii a 2 pasi din insertion sort
   - ce elemente are putea fi considerate elemente pivot – quicksort
2. Partitionati urmatorul vector, folosind ca pivot elementul 7: 16 8 9 13 4 **7** 1 2 10