# PA - Sortări

Daniel Chiș - 2022, UPB, ACS, An I, Seria AC

# Sortări

| | Insertion | Selection | Bubble | Shell | Merge | Heap | Quick | Quick3 |
|---|---|---|---|---|---|---|---|---|
| Random |  |  |  |  |  |  |  |  |
| Nearly Sorted |  |  |  |  |  |  |  |  |
| Reversed |  |  |  |  |  |  |  |  |
| Few Unique |  |  |  |  |  |  |  |  |

# Bubble Sort

# Bubble Sort

Bubble sort este un algoritm de sortare bazat pe comparație, în fiecare pereche de 2 elemente alăturate se realizează comparația și se face o interschimbare în caz că nu sunt în ordine.

Nu este bun pentru seturi mari de date deoarece complexitatea sa este de O(n^2).

```
begin BubbleSort(list)

   for all elements of list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      end if
   end for

   return list

end BubbleSort

```

Algoritm Bubble Sort

# Insertion Sort

# Insertion Sort

Insertion sort este un algoritm de comparare prin care o sublistă a structurii de date este păstrată și sortată mereu. Vectorul este parcurs secvențial iar fiecare element trebuie introdus în lista sortată astfel încât lista să rămână sortată.

Nu este recomandat pentru seturi mari de date deoarece complexitatea este O(n^2).

```
Pas 1 – Dacă este primul element, este deja sortat vectorului
Pas 2 – Se alege elementul următor
Pas 3 – Elementul ales se compară cu elementele din sublista sortată
Pas 4 – Se sifteaza toate elementele din sublistă care sunt mai mari decăt
    elementul ales
Pas 5 – Se inserează elementul
Pas 6 – Se repetă până când structura de date este sortată
```

Algoritm Insertion Sort

# Selection Sort

# Selection Sort

Selection sort este un algoritm de comparare, în care vectorul este împărțit în două părți, la stânga partea sortată și cea nesortată la dreapta. Inițial, partea sortată este goală, iar cea nesortată este reprezentată de toate datele.

Cel mai mic element este selectat din lista nesortată si interschimbat cu elementul de la incipitul listei nesortate, care devine parte din lista sortată. Procesul continuă până când ajungem să avem lista nesortată goală.

Nu este recomandat pentru seturi mari de date deoarece complexitatea este O(n^2).

```
Pas 1 – Setăm MIN la locatia 0
Pas 2 – Căutăm elementul minim din vector
Pas 3 – Se facă interschimbare cu elementul de la locația MIN
Pas 4 – Se incrementează MIN
Pas 5 – Se repetă până când vectorul este sortat
```

Algoritm Selection Sort

# Merge Sort

# Merge Sort

Merge sort este un algoritm bazat pe tehnica divide and conquer. Vectorul este impărțit în jumătăți până pe care le sortează la recombinare.


Este unul dintre cei mai buni algoritmi de sortare. Complexitatea este  O(n log n).

Calcularea complexității:

https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/analysis-of-merge-sort

https://www.cs.auckland.ac.nz/courses/compsci220s1c/lectures/2016S1C/CS220-Lecture09.pdf

```
procedure mergesort( var a as array )
   if ( n == 1 ) return a

   var l1 as array = a[0] ... a[n/2]
   var l2 as array = a[n/2+1] ... a[n]

   l1 = mergesort( l1 )
   l2 = mergesort( l2 )

   return merge( l1, l2 )
end procedure

procedure merge( var a as array, var b as array )

   var c as array
   while ( a and b have elements )
      if ( a[0] > b[0] )
         add b[0] to the end of c
         remove b[0] from b
      else
         add a[0] to the end of c
         remove a[0] from a
      end if
   end while

   while ( a has elements )
      add a[0] to the end of c
      remove a[0] from a
   end while

   while ( b has elements )
      add b[0] to the end of c
      remove b[0] from b
   end while

   return c

end procedure
```

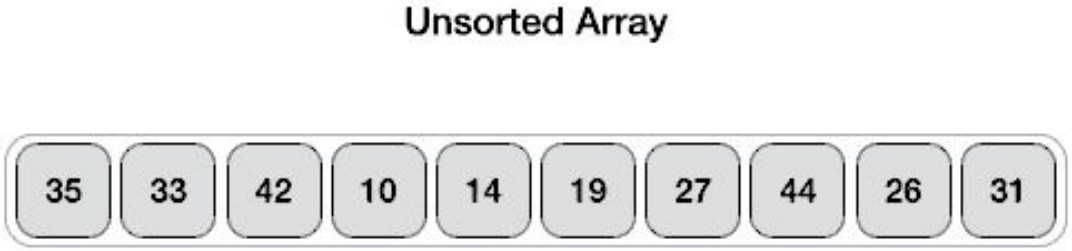
Algoritm Merge Sort

# Quick Sort

# Quick Sort

Quick sort este un algorithm extrem de eficient bazat pe împărțirea vectorului în sub seturi de date. Vectorul este împărțit în două părți, una cu valori mai mici decât un pivot ales iar cealaltă cu valori mai mari. Quick sort împarte vectorul în două și apoi se apelează recursiv de două ori pentru a sorta vectorii generați.

Algoritmul este eficient pentru seturi de date mari și are complexitatea O(n^2).



```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex)
    quickSort(array, pivotIndex + 1, rightmostIndex)

partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
  if element[i] < pivotElement
    swap element[i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

Algoritm Quick Sort

# Exerciții

# Exerciții

1. Se dau următorii vectori
   25 1 44 56 100 9 2

6, 5, 3, 2, 8, 10, 9

100 67 34 22 15 10 2

Să se sorteze cei trei vectori crescător utilizând toți algoritmii prezentați în laborator.

2. Creati un vector cu 10000 de valori random. Să se afișeze timpul de rulare al fiecărui algoritm și care a fost cel mai performant.

Notă: s-ar putea ca datele sa fie prea puține și timpul de rulare să dea mereu 0, în acest caz măriți vectorul.

# Exerciții FIIR

Se dau următorii vectori
25 1 44 56 100 9 2

6, 5, 3, 2, 8, 10, 9

100 67 34 22 15 10 2

Să se sorteze cei trei vectori crescător utilizând 3 algoritmi la alegere prezentați în laborator.

Tutorial pentru lucrul cu vectori în C: **https://www.tutorialspoint.com/cprogramming/c_arrays.htm**