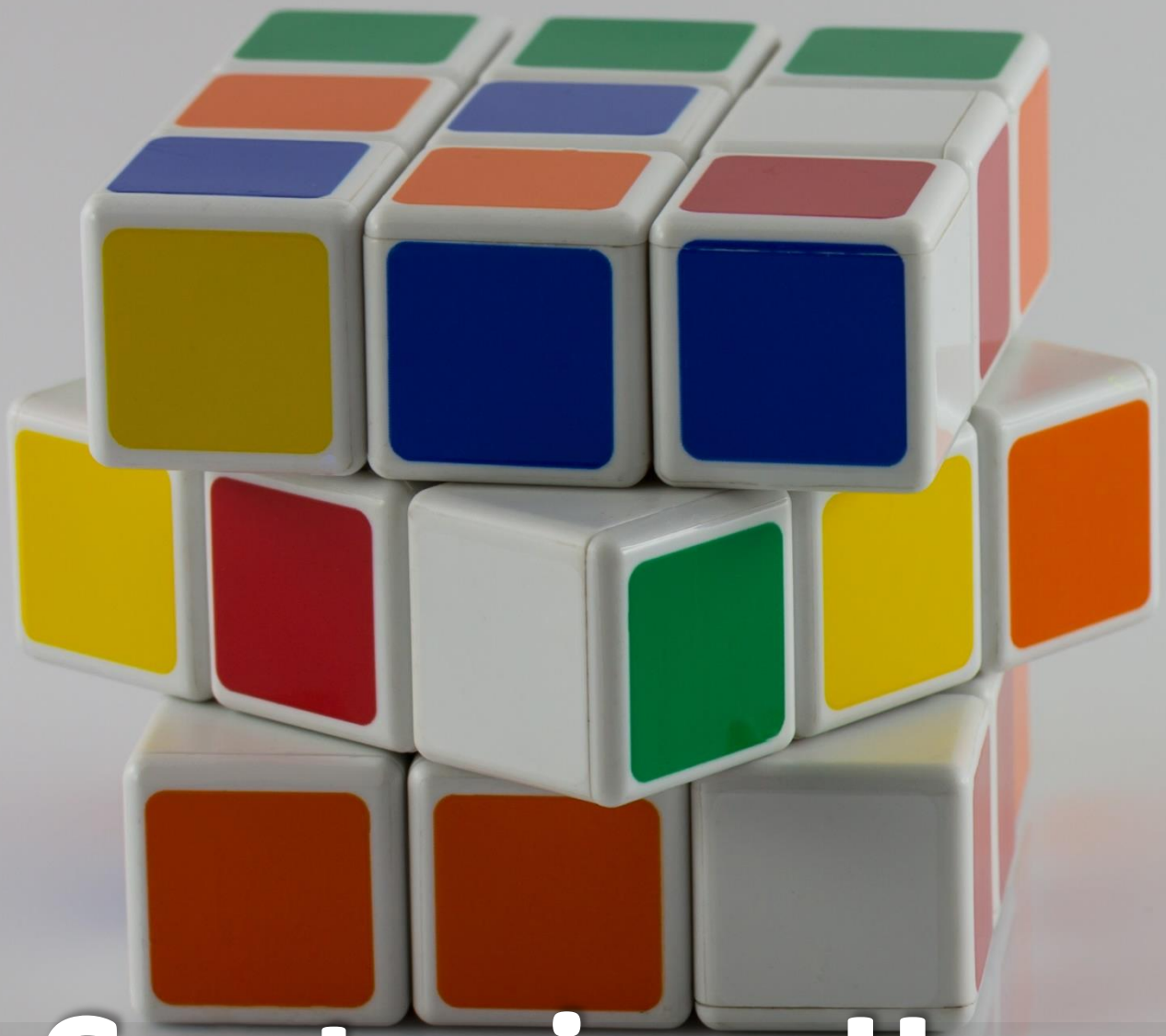

# Sortering II

# Sortering

I skal nu implementere nogle af øvelserne (øvelse 1, 3, 4, 5, 6, 7 og 8) af opgaven Sortering i C prøv med bøde store og små datamængder samt med og uden pointers.

**Analyser algoritmerne grundigt.**

På Moodle er der vedhæftet pseudo-kode på SelectionSort, InsertionSort og MergeSort. Endvidre kan i finde hjælp her https://www.geeksforgeeks.org/sorting-algorithms/#algo. På siden kan i se den implementeret C kode. Hvis i har behov sammenlign C koden med nedenstående pseudokode,

## Sorteringsalgoritmer pseudokode

Dette er en version af algoritmerne, denne kan være lidt forskellige fra bøger og hvis i søger hjælp på nettet.

### Selection-sort:

Input: Array A [1 ... n], af elementer i vilkårlig rækkefølge; array størrelse n

Output: Array A [1 ... n] af de samme elementer, men i stigende rækkefølge (mere præcist i ikke-faldende rækkefølge - nogle elementer er ens)

*Algoritme*:

```
for i= 1 to n do
// find min element in A[i...n]
// and put it in the i'th position (i.e. at A[i])

      min_index <-- i

      //locate min
      for j= i+1 to n do

            if A[j] < A[min_index] then min_index <-- j

      //put the min where it belongs
      swap( A[i], A[min_index] )
```

### Insertion-sort:

Input: Array A [1 ... n], af elementer i vilkårlig rækkefølge; array størrelse n

Output: Array A [1 ... n] af de samme elementer, men i ikke-faldende rækkefølge

*Algoritme:*

```
for i= 2 to n do
// insert i'th element A[i] into the already sorted A[1...i-1]

      // keep swapping the value in A[i] with previous element
      // till it "bubbles" down the right place
      // (i.e. keep swapping till the previos element is smaller)

      j <-- i
      while j>1 AND A[j-1]>A[j] do // when i=1 there is no prev element

            swap( A[j], A[j-1] )
            j <-- j-1
```

Begge af ovenstående algoritmer benytter swap() som subrutine. Nedenstående er pseudo-kode at swap()  denne kode bruger ingen ekstra plads. Subrutinen benytter en bit-wise XOR-operation. Lignende resultater kunne opnås med tal ved hjælp af + og - operatorer. Prøv som en øvelse. (Og sørg for at du også forstår nedenstående version! ;-)

## swap( A, B ):

Input: To variabler A og B, udtrykt i binær notation

Output: De samme variabler, men med deres værdier er byttet om

*Algoritme:*

```
A <-- A XOR B
B <-- A XOR B
A <-- A XOR B
```

### Merge-sort:

Det er lettere at se denne algoritme som sortering af separate sektioner af array A. Således er det generiske input af algoritmer ovenfor ændret lidt ved at tilføje to parametre først og sidste.

Input: Array A [1 ... n], af elementer i vilkårlig rækkefølge; også første og sidste position 1 <=første <=sidste <=n

Output: Array A [1 ... n] af de samme elementer, men elementerne mellem første og sidste (dvs. A [første ... sidste]) er i ikke-faldende rækkefølge. Alle elementer før først og efter sidst er uændrede (og således ændres ikke sæt af elementer mellem første og sidste - kun sorteret).

*Algoritme:*

```
if (last ≤ first) then RETURN //there's nothing between first & last: 1 element is always sorted
else

      middle <-- |_ (first+last)/2 _|
      merge_sort( A, first, middle )
      merge_sort( A, middle+1, last )
      merge( A[first...middle], A[middle+1...last]; A[first...last] )
```

Denne algoritme tager to sorterede arrays og kombinerer dem til et. De to inputarrays sendes med værdi (dvs. indholdet kopieres implicit af algoritmen, før den starter).

Input: Arrays L1 [...] og L2 [...], hver med elementer i ikke-faldende rækkefølge; det sidste element er "dummy", lig med uendelig. Disse arrays ændres ikke af algoritmen og påvirkes heller ikke af ændringer foretaget inden for algoritmen, mens den afvikles;

n = det samlede antal ikke-dummy-elementer i L1, L2

Output: Array A [1 ... n] af de samme elementer som i L1 og L2, men alle elementerne er i ikke-faldende rækkefølge

```
// set two pointers at the begining of the array
p1 <-- 1 // pointer for L1
p2 <-- 1 // pointer for L2

for i=1 to n do
// the next - i'th - element of A will be the smaller of the two pointed by p1,p2

      if ( L1[p1] < L2[p2 )
       then

                A[i] <-- L1[p1]
                p1++

       else

                A[i] <-- L2[p2]
                p2++
```