Sorteeralgoritmes

Er bestaan een aantal verschillende manieren om een rij van gegevens te sorteren. We gaan in volgende enkele gekende sorteeralgoritmes verduidelijken aan de hand van een rij van N getallen die moeten gesorteerd worden van klein naar groot.

In volgende een stukje code om in een rij twee elementen van plaats te verwisselen

```
function verwissel (rij, i, j){
        if (i >= 0 && i < rij.length && j >= 0 && j < rij.length){
                var hulp= rij[i];
                rij[i]= rij[j];
                rij[j]= hulp;
        }
}
```

1. Selection sort

   Selection sort is een eenvoudig sorteeralgoritme waarbij voor elke index i vanaf het eerste tot het voorlaatste element van de rij, de index *indexKleinste* wordt gezocht van het kleinste element in de rij vanaf plaats i tot het laatste element. Het element op de plaats *indexKleinste* wordt dan verwisseld met het element op de plaats i.

Voorbeeld voor de rij					en	i	indexKleinste
15	45	23	66	6		1	5
6	45	23	66	15		2	5
6	15	23	66	45		3	3
6	15	23	66	45		4	5
6	15	23	45	66			

In javascript:

```
function selectionSort(rij){
        for (var i = 0 ; i < rij.length – 1 ; i++){
                var     indexKleinste= i;
                for (var j = i + 1 ; j < rij.length ; j++){

                        if (rij[j] < rij[indexKleinste]) {
                                indexKleinste= j;}
                }
                verwissel(rij , i , indexKleinste);
        }
}
```

2. Bubble sort

Bij dit algoritme zullen we de rij N-1 keer doorlopen, N-1 zogenaamde **passes**. Bij elke pass beginnen we achteraan de rij en vergelijken we elke element met het element dat er net voor ligt. Wanneer een element wordt voorafgegaan door een groter element, zullen de twee elementen verwisseld worden.

Na de eerste pass zal op die manier het kleinste element op de eerste plaats staan, na de tweede pass zal het tweede kleinste element op de tweede plaats staan,…, na de (N-1)-de pass zal het op één na grootste element op de voorlaatste plaats staan en zal dus het grootste element op de laatste plaats van de rij staan.

In javascript:

```
function bubbleSort(rij){
        for (var i = 1; i < rij.length ; i++){
                for (var j = rij.length – 1; j >= 1 ; j--){
                        if (rij[j]  < rij[j-1]) {
                          verwissel(rij , j, j-1);
                        }
                }
         }
}
```

Merk op dat na de i-de pass de eerste i elementen van de rij al gesorteerd zijn en dat we dus niet meer tot helemaal vooraan in de rij moeten gaan.

optimalisatie 1:

```
function bubbleSort(rij){
        for (var i = 1; i < rij.length ; i++){
          for (var j = rij.length – 1; j >=  i ;  j--){
                if (rij[j-1] > rij[j]) {
                        verwissel(rij, j , j -1);
                }
        }
       }
}
```

Wanneer een rij na de ide pass al gesorteerd is, dan zullen we in de (i+1) de pass geen verwisseling meer uitvoeren.

optimalisatie 2:

```
function bubbleSort(rij){
        var ok= true;
        var i = 1;

        while( i < rij.length && ok){
          ok= false;

          for (var j = rij.length – 1; j >= i ; j--){
                if (rij[j-1] > rij[j]) {
                        ok= true;
                        verwissel(rij , j , j-1);
                }
        } // ok zal op dit ogenblik enkel nog true zijn indien er een verwisseling is uitgevoerd

        i++;
        }
}
```

3. Insertion sort

Bij dit sorteeralgoritme gaan we ook N-1 keer door de rij. We onthouden voor elke i (i gaande van 2 tot het aantal elementen van de rij) het element op plaats i in een hulp-variable. Daarna schuiven we alle elementen vanaf plaats (i – 1) één plaats op naar rechts totdat we een element gevonden hebben dat kleiner is dan hulp. Op die manier vinden we de plaats waar hulp geplaatst kan worden.

Op die manier gaan we dus voor elk i-de element (i:2.. aantal elementen van de rij) dit element op de goede plaats zetten in de rij gaande van element1 … element(i-1) . Merk op dat op elk moment de rij gaande van plaats 1 tot (i - 1) gesorteerd is voor elke i gaande van 2 tot einde van de rij.

Dit is de manier waarop kaarters hun kaarten sorteren en dit algoritme wordt dan ook al eens “kaarters-algoritme” of cardSort genoemd.

De volgende rij		22	3	5	4	6	zal dus als volgt evolueren
i	Hulp						
2	3	3	22	5	4	6	
3	5	3	5	22	4	6	
4	4	3	4	5	22	6	
5	6	3	4	5	6	22	

In javascript:

```
function insertionSort(rij){
for (var i = 1; i < rij.length ; i++){
  var hulp= rij[i];

 var j= i-1;
while (j >= 0 && rij[j] > hulp)  {rij[j+1]= rij[j];   j--;} // opschuiven van elementen naar rechts

rij[j+1]= hulp;
}
}
```

4. Merge sort

Het mergen van twee gesorteerde rijen gebeurd als volgt:
Bijvoorbeeld: rij1 = [1,2,4,7]
rij2 = [3,5,6,8]

Maak een nieuwe rij met aantal elementen= ri1.length + rij2.length. Element van de eerste rij vergelijken met element van de tweede rij ; volgend element van de resultaat rij wordt opgevuld met kleinste van de twee en we gaan in deze rij één element verder.

Dus in javascript:

```
function merge(rij1,rij2){

var rij = new Array(rij1.length + rij2.length);
var i1= 0;
var i2= 0;
var i= 0;

while ( i1 < rij1.length && i2 < rij2.length)
        { if (rij1[i1] < rij2[i2]) {
                rij[i]= rij1[i1];
                i1++;
                }
        else {
                rij[i]= rij2[i2];
                i2++;
                }
        i++;
}
// resterend gedeelte van rij1 indien rij2 eerst “op”
while (i1 < rij1.length) { rij[i]= rij1[i1];
                           i1++;
                           i++;
                           }

// resterend gedeelte van rij2 indien rij1 eerst “op”
while (i2 < rij2.length) { rij[i]= rij2[i2];
                           i2++;
                           i++;
                           }

return rij;
}
```

Bij het merge-sort algoritme gaan we dit mergen toepassen op een rij waarvan we veronderstellen dat de eerste helft gesorteerd is en de tweede helft gesorteerd is.
Bijvoorbeeld
Rij = [1,2,4,7, 3,5,6,8] → [1,2,3,4,5,6,7,8]

Om ervoor te zorgen dat de rij zodanig getransformeerd wordt dat op een bepaald ogenblik de eerste helft gesorteerd is en de tweede helft gesorteerd is, zal het mergeSort algoritme recursief worden opgeroepen.

Volgend algoritme zal een rij die van plaatsen “van” tot en met ”midden-1” gesorteerd is en die van “midden” tot en met “tot” gesorteerd is samenvoegen (mergen) tot 1 gesorteerde rij.

```
function merge(rij,van,midden,tot){
        var iLinks= van;
        var iRechts= midden;

        var tempRij= new Array(tot - van);
        var i = 0;

        while (iLinks < midden &&  iRechts <= tot){

         if (rij[iLinks] < rij[iRechts]) {
                tempRij[i]= rij[iLinks];
                iLinks++;}
         else {
                tempRij[i]= rij[iRechts];
                iRechts++;}

         i++;
        }

        while (iLinks < midden) {tempRij[i]= rij[iLinks]; iLinks++;i++}

        while (iRechts <= tot) {tempRij[i]= rij[iRechts]; iRechts++;i++}


   for (var i = van ; i <= tot ; i++) rij[i]= tempRij[i - van]  ;
}
```

Volgend recursief algoritme zal er bij oproep van mergeSort(rij,0,rij.length-1) dan voor zorgen dat de volledige rij gesorteerd wordt:

```
function mergeSort(rij,van,tot){

        if (van < tot){
                        var helft = ((van + tot) - (van + tot)%2)/ 2;
                        mergeSort(rij,van,helft); // sorteer de eerste helft recursief
                        mergeSort(rij,helft+1,tot); // sorteer de tweede helft recursief

                        merge(rij,van,helft+1,tot); // voeg de gesorteerde deelrijen samen

                        }

        }
```

Voorbeeld stel de rij bestaat uit twee elementen voorbeeld rij = [110,5] dan mergeSort(rij,0,1) → helft = 0

➔ mergeSort(rij,0,0) → ok
➔ mergeSort(rij,1,1) → ok
➔ merge(rij,0,1,1) → [5,110] → gesorteerd

ander voorbeeld: rij = [110 , 5 , 111 , 22] dan mergeSort(rij,0,3)

➔ mergeSort(rij,0,1) → zie vorige → [5 , 110 ,111 , 22]
➔ mergeSort(rij,2,3) → [5,110,22,111]

➔ merge(rij,0,2,3) → [5,22,110,111] gesorteerd !

5. QuickSort

Ook quicksort is een recursief algoritme. Hierbij bestaat het algoritme eruit om een spil te zoeken en op basis van die spil de elementen in de rij zodanig te herschikken dat alle elementen links van de spil kleiner zijn dan deze spil en alle elementen rechts van de spil groter (ook de spil verandert hierbij van plaats). Dit systeem wordt dan recursief toegepast op de deelrijen voor en achter de spil.

```
function quickSort(rij, iLinks, iRechts){

if (iLinks < iRechts){

        var i= iLinks;
        var j= iRechts;

        var spil= rij[iRechts];

        while (i < j){
                while (i < j && rij[i] <= spil) {i++;}
                while (rij[j] > spil) {j--;}

                if (i < j) verwissel(rij,i,j);
        } // na deze lus zullen alle elementen links van de spil kleiner zijn en
          // alle elementen rechts van de spil groter + spil staat op ide plaats

        quickSort(rij, iLinks, i-1); //doe zelfde voor deelrij van iLinks … (plaats van spil) - 1
        quickSort(rij, i, iRechts); // doe zelfde voor deelrij van (plaats spil ) + 1 tot iRechts

 } // einde if
}
```