Programmeermethoden

Recursie

Walter Kosters en Jonathan Vis

week 11: 18–22 november 2024

www.liacs.leidenuniv.nl/~kosterswa/pm/

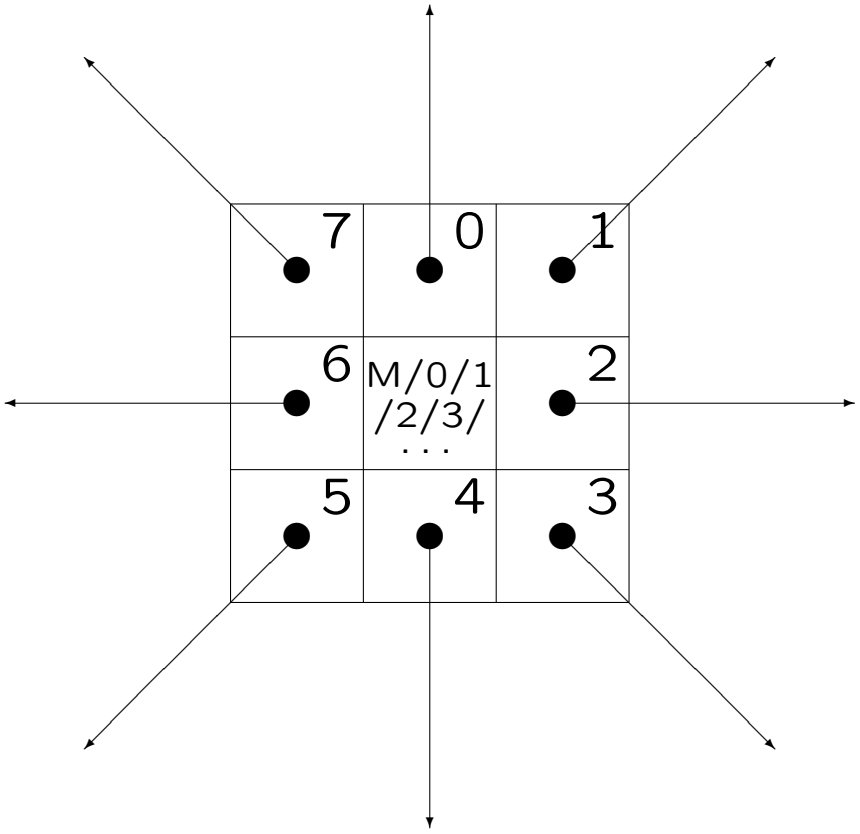
Vierde programmeeropgave — 1

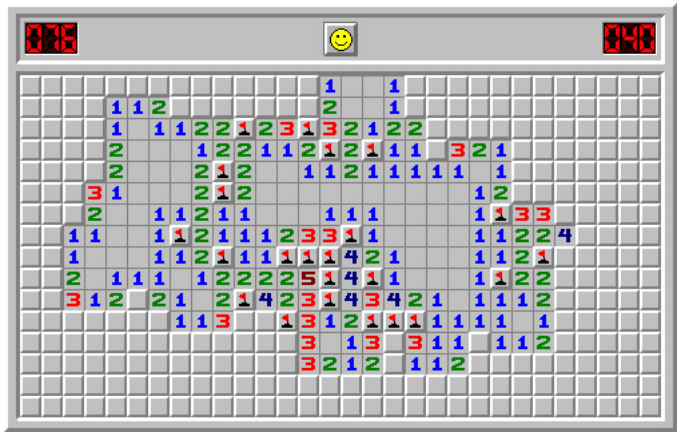
Koffiesweeper programmeren we als volgt:

- week 1 (“10”): pointerpracticum, opgave lezen
- week 2 (“11”): klassen, pointerbord, meerdere files, ruw spelen
- week 3 (“12”): spel helemaal in orde maken, stapel
- week 4 (“13”): recursie, experiment (`gnuplot`), verslag

www.liacs.leidenuniv.nl/~kosterswa/pm/op4pm.php

Vierde programmeeropgave — 2





Vierde programmeeropgave — 3

Als je C++-code over meerdere files verdeelt, helpt een **makefile** bij het compileren (aanroep: `make`). Stel je hebt:

file `kobord.h`

```
class kobord {
  ...
  void print ( );
};//kobord
```

file `kobord.cc`

```
#include <iostream>
#include "kobord.h"
// implementatie van
// prototypes uit
// kobord.h
void kobord::print ( ) {
  ...
}//kobord::print
```

file `hoofd.cc`

```
#include <iostream>
#include "kobord.h"
...
int main ( ) {
  kobord X;
  X.print ( );
  ...
}//main
```

De makefile ziet er dan bijvoorbeeld uit als (let op tabs!):

kobord = koffiebord

```
all: kobord.o hoofd.o
←TAB→g++ -Wall -Wextra -o koffiesweeper kobord.o hoofd.o
kobord.o: kobord.cc kobord.h
←TAB→g++ -Wall -Wextra -c kobord.cc
hoofd.o: hoofd.cc kobord.h
←TAB→g++ -Wall -Wextra -c hoofd.cc
```

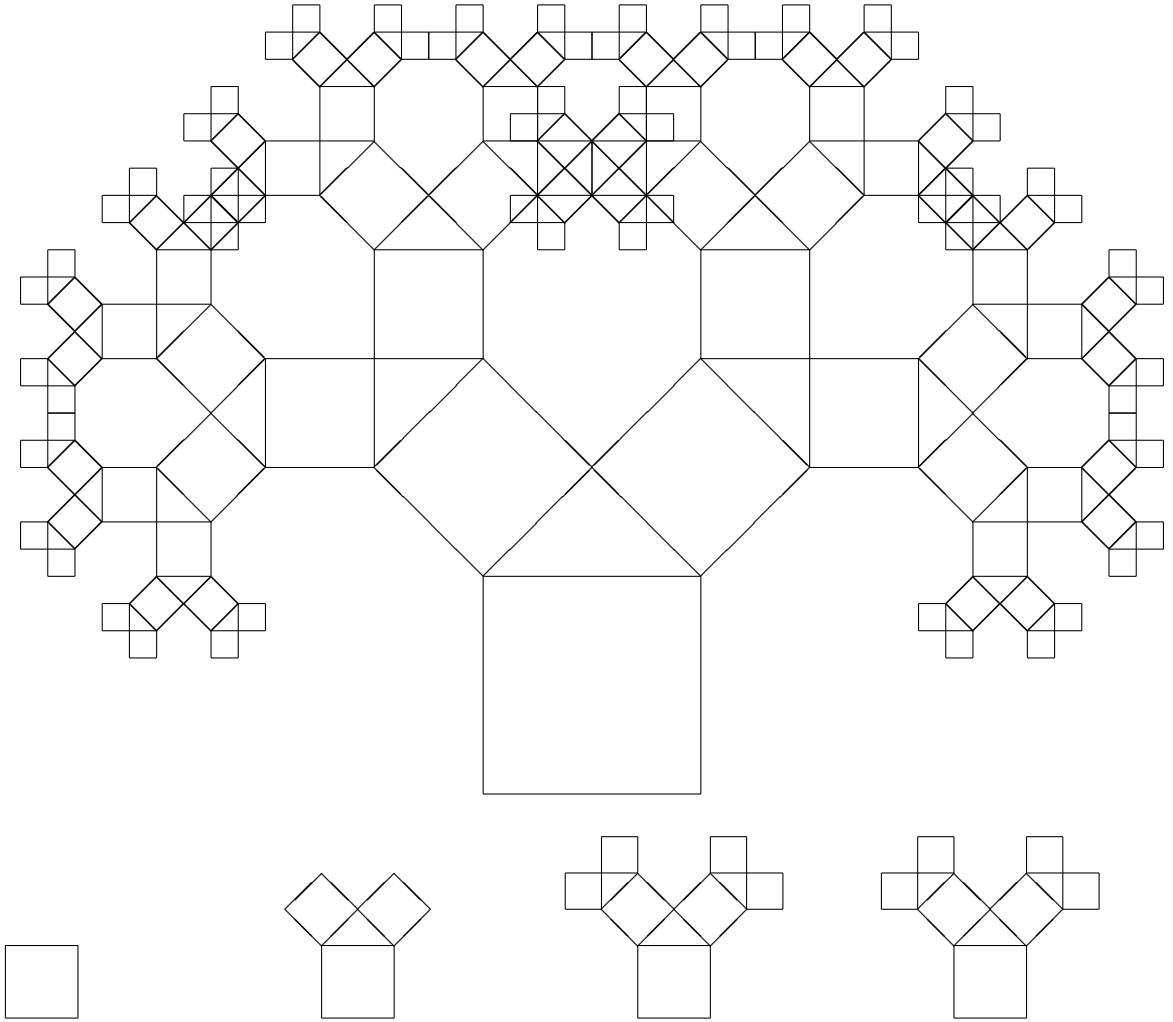
in Code::Blocks via projecten, zie video

Vierde programmeeropgave — 4

```
// file kobord.h
class bordvakje {
  public:  bordvakje* buren[8];
           char info; // ... TODO
};//bordvakje

class kobord {
  private: bordvakje* ingang;
  public:  void print ( );
           // ... TODO
};//kobord

// file kobord.cc = koffiebord.cc
void kobord::print ( ) { ... }
```



Boom van Pythagoras

Basisidee:

proces is recursief als het naar zichzelf verwijst

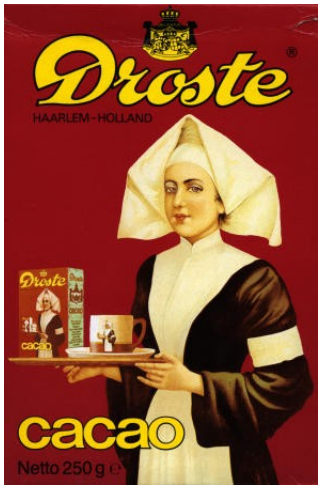
functie is recursief als deze zichzelf (in)direct aanroept

Voorbeeld ("inductie"), met $S[1] = 1$ (of $S[0] = 0$):

$$S(n) = \sum_{i=1}^{n} i^{42} = \sum_{i=1}^{n-1} i^{42} + n^{42} = S(n-1) + n^{42}$$

Woordenboek:

Recursie: zie Recursie
Fiets: zie Rijwiel
Rijwiel: zie Fiets



Recursief oplossen

Een recursief/ve proces/procedure/functie bestaat in het algemeen uit twee delen:

1. één of meer klein(st)e (eenvoudig(st)e) gevallen die direct oplosbaar zijn: de basisgevallen

2. een algemene methode die een bepaald geval reduceert tot één of meer kleinere (eenvoudiger) gevallen, waarbij men uiteindelijk op een basisgeval uitkomt

Algemene gedaante van een recursieve functie:

if basisgeval **then**

los op zonder recursie; // makkelijk

else

één of meer recursieve eenvoudigere aanroepen;

fi

Let op de symbolische notatie in pseudo-code.

Probleem:

Betaal(X) = Betaal het bedrag X

Oplossing:

Betaal(0) = Doe niets

Betaal(X) = Geef de grootste munt $Y \leq X$

Betaal daarna(?) de rest Betaal($X - Y$)

Vraag:

Wat kan er nog fout gaan? betaal 30 met 25/10 (vóór 2002)

Faculteit recursief

Recursieve definitie van n-faculteit ($n!$):

$$\text{fac}(n) = \begin{cases} 1 & \text{als } n = 0 \\ n \times \text{fac}(n-1) & \text{als } n > 0 \end{cases}$$

Recursieve C++-functie ($\texttt{n} \geq 0$):

```
long faculteit (int n) {
  if ( n == 0 ) // basisgeval
    return 1;
  else
    return n * faculteit (n-1); // recursie
}//faculteit
```

Een functie mag zichzelf (in)direct aanroepen: recursie.

```
int som (int n) { // berekent 1 + 2 + ... + n   versie 1
  int i, res = 0;
  for ( i = 1; i <= n; i++ ) res += i;
  return res;
}//som

int somrecursief (int n) { // idem, recursief   versie 2
  if ( n == 0 ) return 0;
  else return n + somrecursief (n-1);
}//somrecursief

int somslimGauss (int n) { // en nog eens ...   versie 3
  return ( n * ( n + 1 ) ) / 2;
}//somslim
```

De ggd kan ook recursief berekend worden:

```
int ggdrecursief (int x, int y) {
  if ( y == 0 ) return x;
  else return ggdrecursief (y,x % y);
}//ggdrecursief
```

Je gebruikt eigenlijk:

$$\text{ggd}(x,y) = \begin{cases} x & \text{als } y = 0 \\ \text{ggd}(y, x \bmod y) & \text{als } y \neq 0 \end{cases}$$

Definitie Fibonacci-getallen:

$$\text{fib}(n) = \begin{cases} 1 & \text{als } n = 0 \text{ of } n = 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{als } n > 1 \end{cases}$$

Alternatief: $\text{fib}(1) = \text{fib}(2) = 1$.

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, ...

Fibonacci — recursief

Recursieve C++-functie:

```
long fib1 (int n) {
  if ( ( n == 0 ) || ( n == 1 ) )
    return 1;
  else
    return ( fib1 (n-1) + fib1 (n-2) );
}//fib1
```

Er is hier sprake van een watervaleffect:
de aanroep `fib1 (5)` veroorzaakt
14 andere (dubbele) aanroepen.

Fibonacci — Dynamisch programmeren

```
const int MAX = 100;
long memo[MAX]; // globaal, initialiseer met 0-en!

// recursie met array
long fib2 (int n) {
  if ( n >= MAX ) // helaas
    return fib2 (n-1) + fib2 (n-2);
  else
    if ( memo[n] > 0 ) // al eerder berekend
      return memo[n];
    else {
      if ( ( n == 0 ) || ( n == 1) )
        memo[n] = 1;
      else
        memo[n] = fib2 (n-1) + fib2 (n-2);
      return memo[n];
    }//else
}//fib2
```

Iteratief: opsommen tot je bij fib(n) bent:

```
long fib3 (int n) {
  long eerste = 1, tweede = 1, hulp;
  int teller;
  for ( teller = 2; teller <= n; teller++ ) {
  // nu geldt: eerste == fib (teller-2) en
  // tweede == fib (teller-1)  ("invariant")
     hulp = tweede;
     tweede = eerste + tweede;
     eerste = hulp;
  }//for
  return tweede;
}//fib3
```

Deze versie is erg geschikt voor “grote getallen”.

Gesloten formule (nauwelijks te berekenen!):

$$\texttt{fib}(n) = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{n+1}\right)$$

↑

klein in absolute waarde

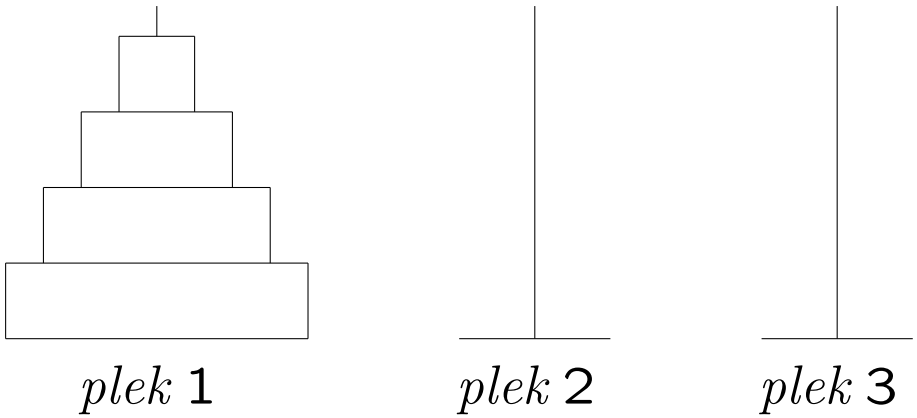
Met matrices:

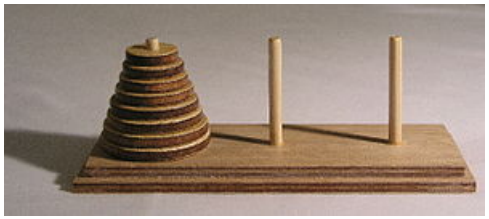
$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} \texttt{fib}(n) & \texttt{fib}(n-1) \\ \texttt{fib}(n-1) & \texttt{fib}(n-2) \end{pmatrix}$$

Torens van Hanoi — 1

Gegeven: n ($n \geq 1$) schijven met gat in het midden, alle verschillend in grootte, en 3 palen = plekken
Beginsituatie: alle schijven liggen boven op elkaar om één paal, en de andere 2 palen zijn leeg
Restrictie: een grotere schijf ligt nooit op een kleinere

Voorbeeld: beginsituatie voor $n = 4$

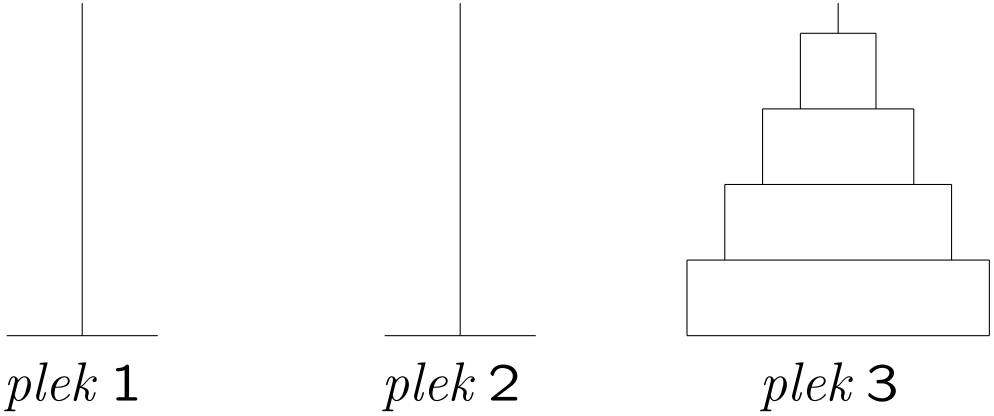




Torens van Hanoi — 2

Doel: breng de hele toren naar een van de lege palen
Acties: per keer mag je één schijf verplaatsen (de bovenste van een stapel), en deze bovenop een andere stapel leggen
Restrictie: er mogen alleen kleinere schijven op grotere worden gelegd

Voorbeeld: eindsituatie voor $n = 4$



Oplossing:

n schijven zo efficiënt mogelijk van start naar doel verplaatsen (via hulp) =

> eerst de bovenste $n-1$ *schijven zo efficiënt mogelijk van start naar hulp verplaatsen (via doel)*,
>
> dan de grote schijf van start naar doel,
>
> en tenslotte de $n-1$ *schijven zo efficiënt mogelijk van hulp naar doel verplaatsen (via start)*

Dat is recursie! Wat is het basisgeval?

Algoritme:

```
// Torens van Hanoi: recursief
// zet toren van n stuks (optimaal) van a naar b via c
// print de zetten
void zet (int n, int a, int b, int c) {
  if ( n > 0 ) {
    zet (n-1,a,c,b);
    cout << "Zet van " << a << " naar " << b << endl;
    zet (n-1,c,b,a);
  }//if
}//zet
```

Het aantal zetten is in totaal $2^n - 1$.

Aanroep: `zet (aantal,1,3,2);`, waarbij `aantal` het gewenste aantal schijven is. Basisgeval: "niets doen".

Probleem: Zoek een waarde in een *oplopend gesorteerd* array A met n elementen

Oplossing: binair zoeken

- Kijk of het middelste (†) element het gezochte is.
- Stop indien het element gevonden is, of als het te onderzoeken array leeg is.
- Anders: bepaal op grond van vergelijken met dat middelste element of verder (recursie!) gezocht moet worden in de linker helft óf in de rechter helft van het array en herhaal dit.

(†) Als het aantal elementen even is: kies één van de twee middelste.

Binair zoeken: iteratief

```
// Geeft index met A[index] = getal, als getal voorkomt;
// zo niet: resultaat wordt -1.

int binairzoeken (int A[ ], int n, int getal) {
  int links = 0, rechts = n-1;  // zoek tussen links en rechts
  int midden;

  while ( links <= rechts ) {
    midden = ( links + rechts ) / 2;
    // beter(!): midden = links + ( rechts - links ) / 2;
    if ( getal == A[midden] )
      return midden;  // of gevonden = true etc.
    else if ( getal > A[midden] )
      links = midden + 1;
    else
      rechts = midden - 1;
  }//while

  return -1;
}//binairzoeken
```

Binair zoeken: recursief

```
int binairzoeken (int A[ ], int n, int links, int rechts, int getal) {
  int midden;
  if ( links > rechts )           // basisgeval: leeg interval
    return -1;                    // dus stop; niet aanwezig
  else {                          // nu echt zoeken
    midden = ( links + rechts ) / 2;
    if ( getal == A[midden] )     // gevonden!
      return midden;
    else                  // verder zoeken: recursieve aanroepen
      if ( getal > A[midden] )    // rechts hetzelfde doen
        return binairzoeken (A,n,midden+1,rechts,getal);
      else                        // links hetzelfde doen
        return binairzoeken (A,n,links,midden-1,getal);
  }//else echt zoeken
}//binairzoeken
```

Aanroep: `iets = binairzoeken (A,n,0,n-1,getal);`

Recursief sorteren

```
sorteer (rij) =
  if ( rij heeft meer dan 1 element ) then
    verdeel rij in linkerrij en rechterrij;
    sorteer (linkerrij);
    sorteer (rechterrij);
    combineer (linkerrij, rechterrij);
  fi
```

⇓ (zie elders)

Mergesort: $O(n \lg n)$

Quicksort: $O(n \lg n)$

Insertion sort: $O(n^2)$

$n =$ aantal elementen van de rij; $\lg n = {}^2\log n = \log_2 n$

Recursie en parameters 1 ⋈

↑
geen tentamenstof

```
void print1 (int    a) {  // call by value
  if ( a > 0 ) {
    a--;
    print1 (a);
    cout << a << ", ";

  }//if
}//print1
```

Nu doen we:

```
getal = 3; print1 (getal); cout << getal << endl;
```

Dat levert: `0, 1, 2, 3`

```
void print2 (int & a) {  // call by reference
  if ( a > 0 ) {
    a--;
    print2 (a);
    cout << a << ", ";

  }//if
}//print2
```

Nu doen we:

```
   getal = 3; print2 (getal); cout << getal << endl;
```

Dat levert: `0, 0, 0, 0`

```
void print3 (int & a) {  // call by reference
  if ( a > 0 ) {
    a--;
    print3 (a);
    cout << a << ", ";
    a++; // en a weer terugzetten
  }//if
}//print3
```

Nu doen we:

```
   getal = 3; print3 (getal); cout << getal << endl;
```

Dat levert: `0, 1, 2, 3`

Recursie wordt ook vaak gebruikt bij het programmeren van spellen als Schaken, Go en Boter, kaas en eieren.

We willen het aantal vervolgpartijen $S.Aantal\ (\ )$ weten vanuit een gegeven stand (= positie) S:

```
S.Aantal ( ) ::
    Teller ← 0;
    if S is eindstand then
        return 1;
    fi
    for alle mogelijke zetten z do
        S.DoeZet (z);
        Teller ← Teller + S.Aantal ( );
        S.OntDoeZet (z);
    od
    return Teller;
```

Bij deze oplossing is ervoor gekozen de Stand S niet "kapot" te maken, vandaar de aanroep $OntDoeZet\ (z)$. Gebruik makend van de eigenschap dat recursieve aanroepen S niet verstoren, doet de buitenste aanroep dat nu ook niet.

Je kunt ook, voor iedere z opnieuw, de zet doen in een kopie van S, zodat je S nooit vernielt.

Overigens: er zijn 255.168 verschillende *partijen* Boter, kaas en eieren. En je hebt dan meteen het hele spel doorgerekend (zie later).

Opgave 1 van het tentamen van 6 januari 2020:

In het array `int A[n]` staan `n` (een `const int` ≥ 3) verschillende gehele getallen.

a. (6) Schrijf een Booleaanse C++-functie `gem (A,n)` die `true` geeft als er precies één array-element in `A` is (niet eerste of laatste) dat *exact* het gemiddelde is van zijn beide directe buren, en anders `false`. Dus `true` voor array `10 8 6 1`, en `false` voor `2 5 9`.

b. (7) Schrijf een C++-functie `int stijg (A,b,n)` die de lengte van een langste stijgende aaneengesloten deelrij van `A` geeft, en diens begin-index in `b` oplevert. Als er meerdere deelrijen het langste zijn: de kleinste `b`. Dus array `2 7 4 5 6 1 3 8` geeft 3, met `b` = 2 (deelrij `4 5 6`, even lang als `1 3 8`).

c. (4) We nemen in dit onderdeel aan dat `A` uit precies twee stijgende aaneengesloten deelrijen bestaat. Schrijf een C++-functie `int kl (A,n)` die het kleinste element van `A` oplevert, door de functie van **b** te gebruiken, en dan de twee kandidaten te vergelijken.

d. (4) Schrijf een C++-functie `bu (A,n)` die `A` *aflopend* sorteert met *bubblesort*. De functie moet stoppen als er tijdens een doorgang/ronde geen verwisselingen waren.

e. (4) Hoeveel vergelijkingen tussen array-elementen doet de functie van **d** minimaal en maximaal, uitgedrukt in `n`? En wanneer gebeurt dat?

Uitwerking 6.1.2020: Opgave 1

```
a. bool gem (int A[ ], int n) {
     int i, tel = 0;
     for ( i = 1; i < n-1; i++ )
       if ( A[i-1] + A[i+1] == 2 * A[i] ) tel++;
     return ( tel == 1 );
   }//gem
b. int stijg (int A[ ], int & b, int n ) {
     int i, langste = 1, lang = 1, begin = 0; b = 0;
     for ( i = 1; i < n; i++ )
       if ( A[i] > A[i-1] ) {
         lang++; if ( lang > langste ) { langste = lang; b = begin; }
       }//if
       else { begin = i; lang = 1; }//else
     return langste;
   }//stijg
c. int kl (int A[ ],int n) {
     int b, s; s = stijg (A,b,n); if ( b == 0 ) b = s;
     if ( A[0] < A[b] ) return A[0]; else return A[b];
   }//kl
d. void bu (int A[ ], int n) {
     int i, j = 0, tmp; bool wissel = true;
     while ( wissel ) {
       wissel = false; j++;
       for ( i = 0; i < n-j; i++ ) // of i < n-1 (*)
         if ( A[i] < A[i+1] ) { // aflopend sorteren
           tmp = A[i]; A[i] = A[i+1]; A[i+1] = tmp; wissel = true; }}//if&while
   }//bu
e. Al aflopend gesorteerd: n-1 vergelijkingen; omgekeerd gesorteerd:
   n-1 + n-2 + ... + 1 = n(n-1)/2 = O(n^2)  (bij (*): n(n-1))
```

- maak de vierde programmeeropgave — de deadline is op maandag 9 december 2024
- lees Savitch Hoofdstuk 13
- lees dictaat Hoofdstuk 3.10, 4.2.2, 4.2.7
- maak opgaven 57/61 uit het opgavendictaat
- www.liacs.leidenuniv.nl/~kosterswa/pm/