Uitwerking opgave 3
Tentamen Algoritmiek, augustus 2007

a. Brute force: botweg alle $\frac{1}{2}n(n-1)$ paren `(i,j)` aflopen (met `i` $<$ `j`), en zoeken naar een paar met som gelijk aan `W`. Algoritme in pseudocode:

```
eindi := 0; eindj := 0;
for i := 1 to n do
   for j := i+1 to n do
      if (A[i] + A[j] = W) then
         eindi := i; eindj := j;
      fi
   od
od
```

b. Divide and conquer: verdeel het array in twee gelijke stukken, een linkerhelft en een rechterhelft. Ga vervolgens links zoeken naar een paar (i,j) waarvoor A[i] + A[j] = W, en idem rechts (*recursie*). Als we dan nog geen goed paar gevonden hebben, moeten we vervolgens alleen nog alle paren (i,j) bekijken met i in de linkerhelft en j in de rechterhelft.
Aangezien door de recursie steeds een ander, kleiner stuk van het array wordt doorzocht (altijd een 2-macht lang, omdat n een 2-macht is) schrijven we een functie void zoeken(A,l,r), waarin A[l] t/m A[r] wordt bekeken. De functie wordt bij aanvang aangeroepen als: zoeken(A,1,n,W,ii,jj);, met $n \geq 1$ en de aanroepende variabelen ii en jj gelijk aan nul.

```
void zoeken(int A[ ], int l, int r, int W, int& i, int& j) {
   // basisgeval: twee elementen
   if (r == l+1) {
      if (A[l] + A[r] == W) {
         i=l; j=r;
      } // if
   } // if
   else { // meer dan 2 elementen
      int midden = (l+r)/2; // dus ⌊(l+r)/2⌋
      zoeken(A,l,midden,W,i,j);
      // hier kun je nog testen of i en j nul zijn
      // zo nee, dan heb je een i en j gevonden en kun je stoppen
      zoeken(A,midden+1,r,W,i,j);
      // ook hier kun je eerst testen of i en j nul zijn
      for (k=1; k≤n/2; k++) {
         for (l=n/2+1; l≤n; l++)
            if (A[k] + A[l] == W) {
               i = k; j = l;
            }
      } // for
   } // else
} // zoeken
```

Opmerking: in de functie hierboven wordt het laatst tegengekomen paar (i,j) met A[i] + A[j] = W geretourneerd. Door na elk van beide recursieve aanroepen (op de linkerhelft resp. rechterhelft van het (deel)array) de test: if ((i==0)&&(j==0)) toe te voegen, kunnen we eerder stoppen, namelijk als de test false oplevert. Dan is zo'n paar (i,j) gevonden.

c. Decrease by half (algemeen): het probleem ter grootte n wordt gereduceerd tot (één versie van) hetzelfde probleem ter grootte $\frac{n}{2}$. We kunnen ons hier beperken tot het zoeken naar de gevraagde p in ofwel de linkerhelft, ofwel de rechterhelft van het array. Dit kan als volgt.
Laat midden $= \lfloor \frac{l+r}{2} \rfloor$. Bekijk nu A[midden-1],A[midden] en A[midden+1]. (Dat kan dus alleen als het aantal elementen r-l+1 $\geq$ 3.)
Als A[midden-1] $<$ A[midden] $<$ A[midden+1], dan is de rij daar nog stijgende, en derhalve zit p rechts van het midden: p $\geq$ midden+1. Dus rechts verder zoeken.
Als A[midden-1] $<$ A[midden], maar A[midden] $>$ A[midden+1], dan hebben we p gevonden: p = midden. Dus stoppen.
Als A[midden-1] $>$ A[midden], dan wordt er blijkbaar al gedaald, dus dan zit p links van het midden: p $\leq$ midden-1. Dus links verder zoeken.

```
int zoek(int A[ ], int l, int r) {
   // basisgevallen: aantal elementen = 0 of 1
   // voor aantal elementen ≥ 3 kunnen we bovenstaande toepassen
   if (r == l)
      return l;
   if (r == l+1) {
      if (A[l] < A[r])
         return r;
      else
         return l;
   }
   // aantal elementen ≥ 3
   midden = (l+r)/2; // dus ⌊(l+r)/2⌋
   if (A[midden] > A[midden-1]) {
      if (A[midden] > A[midden+1]) }
         return midden;
      else
         return zoek(A,midden+1,r);
   } // if
   else
      return zoek(A,l,midden-1);
} // zoek
```