16.34

1 (a) 4 slangen (2 mogelijkheden voor S en 2 mogelijkheden voor B)

16.39

(b)
```
bool checkSlang (char A[MaxN][MaxN], int m, int n,
      pair<int, int> sl[], string str)
{ bool gehad [MaxN][MaxN];  bool OK;

  for (int i=0; i<m; i++)
     for (int j=0; j<n; j++)
        gehad[i][j]= false;

  OK=true
  for (int k=0; OK && k< str.length(); k++)
  { int i = sl[k].first;
    int j = sl[k].second;
    if (gehad[i][j] || A[i][j] ≠ str[k])
       // vakje al gebruikt of verkeerde letter
       OK = false
  }

  return OK;
} // checkSlang
```

16.49 / 18.11

(c)
```
int aantalSlangen (char A[MaxN][MaxN], int m, int n,
      pair<int,int> sl[], string str, int pos)
{   int aantal;
    if (pos == str.length()) // genoeg vakjes gevuld
    { if (checkSlang (A, m, n, sl, str))
        aantal =1
      else
        aantal=0
    }
    else // pos < str.length
    { aantal =0; // initialisatie
      if (pos ==0) // gewoon alle mogelijke vakjes
      { for (int i=0; i<m; i++)
        { for (int j=0; j<n; j++)
          { sl[pos] = make_pair (i,j);
            aantal += aantalSlangen (A, m, n, sl, str, pos+1);
          }
        }
      }
      else // 0<pos< str.length() ⇒ probeer vakjes die aansluiten
      {    // bij laatst toegevoegde vakje
        int i = sl[pos-1].first;
        int j = sl[pos-1].second
```

```
        if (i>0) // vakje erboven
        { route[pos] = make_pair (i-1, j);
          aantal += aantalSlangen (A, m, n, sl, str, pos+1);
        }
        if (i<m-1) // vakje eronder
        { route[pos] = make_pair (i+1, j);
          aantal += aantalSlangen (A, m, n, sl, str, pos+1);
        }
        if (j>0) // vakje links
        { route[pos] = make_pair (i, j-1);
          aantal += aantalSlangen (A, m, n, sl, str, pos+1);
        }
        if (j<n-1) // vakje rechts
        { route[pos] = make_pair (i, j+1);
          aantal += aantalSlangen (A, m, n, sl, str, pos+1);
        }
      } // 0<pos<str.length()
    } // pos<str.length()
    return aantal;
  } // aantalSlangen.
```

18.37

(d) Als aantalSlangen gebruik zou maken van backtracking, dan zouden we de controles die nu aan het eind, in checkSlangen, worden uitgevoerd, al uitvoeren in aantalSlangen

18.39/21.14

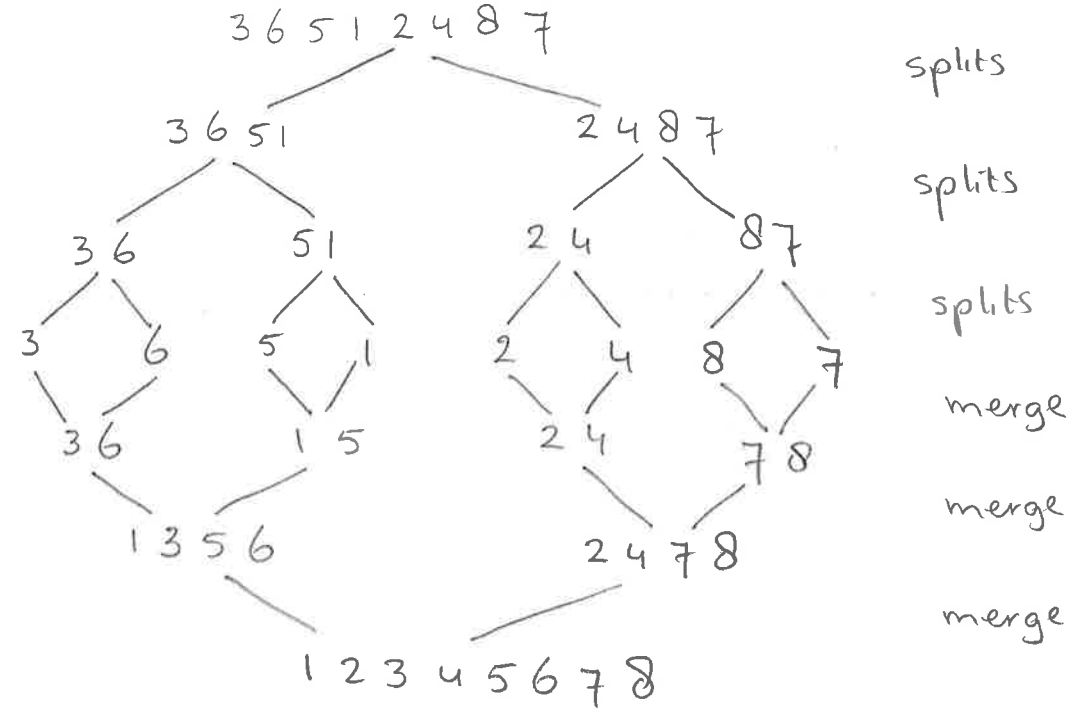
Concreet zouden we

* vóór elke recursieve aanroep controleren of het aan de slang toegevoegde vakje de juiste letter str[pos] bevat, en of datzelfde vakje niet al vóórkomt in array sl.
  Als een van deze tests (of allebei) verkeerd uitpakt, zouden we de recursieve aanroep overslaan.
  → Deze tweede test hoeven we natuurlijk niet te doen als pos == 0.
* de aanroep van checkSlang overslaan en gewoon aantal = 1 maken.

Om alle controles of een vakje niet al voorkomt efficiënt te maken, zouden we het 2D bool-array gehad uit checkSlang meegeven als parameter aan aantalSlangen en consequent bijhouden.

21.23

2(a)



21.29

(b)

```
void merge (int B[], int p, int C[], int q, int A[])
{ int i=0,    // index in B
      j=0,    // index in C
      k=0;    // index in A

  while (i<p && j<q)
  { if (B[i] ≤ C[j]) // pak B[i]
    { A[k] = B[i];
      k++;
      i++;
    }
    else // C[j] is kleiner. Voeg die toe.
    { A[k] = C[j]
      k++;
      j++;
    }
  }

  while (i<p) // kennelijk j=q ⇒ heel C gehad ⇒ kopieer B
  { A[k] = B[i]
    k++;
    i++;
  }
  while (j<q) // kennelijk i=p ⇒ heel B gehad ⇒ kopieer C
  { A[k] = C[j]
    k++;
    j++;
  }

} // merge
```

21.40

(c)
```
void sort (int A[], int N)
{ int B[MaxN], C[MaxN]; // we gaan ervanuit dat MaxN bestaat.
  if (N > 1) // er valt nog wat te sorteren
  { int mid = N/2;
    for (int i = 0; i < mid; i++) // kopieer eerste helft van array A
      B[i] = A[i];                // naar B
    for (int j = 0; j < N-mid; j++) // kopieer tweede helft van
      C[j] = A[mid+j]               // array A naar b.
    sort (B, mid);
    sort (C, N-mid);
    merge (B, mid, C, N-mid, A);
  } // N > 1
} // sort.
```

21.39

(d) De tijdcomplexiteit van merge(sort) is in $\Theta(N \log N)$

In het antwoord bij (a) zien we dat de functie merge op drie niveaus (in het algemeen: $^2\log N$ niveaus) wordt toegepast.
Per niveau worden alle N getallen door de functie merge verwerkt, waarbij ze in een groter deelarray worden gezet dan ze zaten.
Bij onderdeel (c) zien we dat dit per getal een constante hoeveelheid werk / tijd kost. Oftwel, per niveau een hoeveelheid werk in $\Theta(N)$. Met $^2\log N$ niveaus, worden alle 'N' getallen bij elkaar $N \times {}^2\log N$ keer bekeken, waarbij er steeds een constante hoeveelheid werk wordt gedaan. De totale hoeveelheid werk / tijd is dan in $\Theta(N \times {}^2\log N)$.

22.04

3 (a)

reeks (i, j)'

	0	1	2	3	4	5	6	7	8	9
voorhand: j=0	1	2	3	4	2	3	7	8	1	3
achterhand: j=1	1	2	3	1	5	6	4	1	2	3

22.17

(b)

Als i=0, kijken we naar de eerste kaart. Die vormt in zijn eentje een niet-dalende deelrij. Omdat er geen kaarten vóór liggen, kunnen we deze deelrij ook niet verlengen: Dit is onafhankelijk van j.
Als $i \geq 1$ en beide getallen op de voorafgaande kaart i-1 zijn hoger dan a[i][j], dan kunnen we de deelrij met alleen kaart i ook niet naar voren verlengen. De lengte blijft 1.
Als $i \geq 1$ en $a[i-1][0] \leq a[i][j]$ en $a[i-1][1] > a[i][j]$, dan kunnen we kaart i met kant j vooruit, alleen naar voren

verlengen, als we de voorhant van haart i-1 boven leggen, met getal a[i-1][0] erop. We kunnen dan in het gunstigste geval alle reeks(i-1, 0) haarten vóór a[i][j] leggen. Totaal, met haart i erbij, krijgen we een deelrij van 1 + reeks(i-1,0) haarten

Het volgende geval, i≥1 en a[i-1][0] > a[i][j] en a[i-1][1] ≤ a[i][j] is analoog en levert een reeks van lengte 1 + reeks(i-1, 1) op.

Ten slotte: als i≥1 en a[i-1][0] ≤ a[i][j] en a[i-1][1] ≤ a[i][j] dan kunnen we haart i-1 zowel met de voorhant als met de achterhant boven vóór haart i leggen. In het eerste geval kunnen we een deelrij van lengte 1 + reeks(i-1,0) krijgen, in het tweede geval een deelrij van lengte 1 + reeks(i-1,1) krijgen. We kiezen het maximum daarvan. We zoeken namelijk de langste aaneenge-

22.41 sloten deelrij.

(c)

```
int langsteRij (int a[MaxN][2], int n)   // pre: n≥1
{ int reeks[MaxN][2];

  reeks[0][0] = reeks[0][1] = 1;  // basis geval

  for (int i=1; i<n; i++)  // vul array reeks
  { for (int j=0; j<2; j++)
    { if (a[i-1][0] > a[i][j] && a[i-1][1] > a[i][j])
        reeks[i][j] = 1
      else
      { if (a[i-1][0] ≤ a[i][j] && a[i-1][1] > a[i][j])
          reeks[i][j] = 1 + reeks[i-1][0];
        else
        { if (a[i-1][0] > a[i][j] && a[i-1][1] ≤ a[i][j])
            reeks[i][j] = 1 + reeks[i-1][1];
          else  // laatste geval
            reeks[i][j] = 1 + max(reeks[i-1][0], reeks[i-1][1]);
        }
      }
    } // for j
  } // for i
```

22.51

```
int max=1; // initialisatie, komt overeen met i=0
for (int i=1; i<n; i++)
{ for (int j=0; j<2; j++)
  { if (reeks[i][j] > max)
      max = reeks[i][j];
  }
}
return max;
```

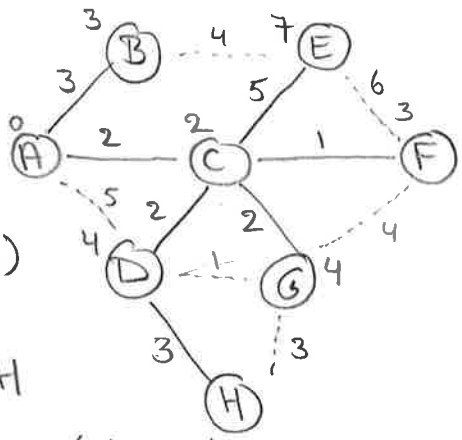
22.56 }

23.04

4) Niet op klaadpapier:

A	B	C	D	E	F	G	H	
0	∞	∞	∞	∞	∞	∞	∞	begin met A
–	3	2	5	∞	∞	∞	∞	kies C, vanaf A
–	3	–	4	7	3	4	∞	kies B, vanaf A
								(alternatief: kies F)
–	–	–	4	7	3	4	∞	kies F, vanaf C
–	–	–	4	7	–	4	∞	kies D, vanaf C
								(alternatief, kies G)
–	–	–	–	7	–	4	7	kies G, vanaf C
–	–	–	–	7	–	–	7	kies E, vanaf C
								(alternatief, kies H
–	–	–	–	–	–	–	7	kies H vanaf D

Resulterende boom:



(stippellijntjes zijn takken die niet in boom belanden)

23.17

(a)

Goede volgordes: 1, 6

23.24

(b) Goede bomen: 2 (als we G kiezen vóór D), 6

Boom 1 is niet goed, want pad A-B-E levert geen verbetering voor E t.o.v. pad A-C-E dat we eerder vinden;

23.30 zie pseudocode.