23.48

1(a) Alle rijtjes van drie goed-geneste haakjesparen:
( ) ( ) ( ), ( ) ( ( ) ), ( ( ) ) ( ), ( ( ) ( ) ), ( ( ( ) ) )

23.51 / 23.52

(b)
```
int sluithaakje (string A, int l)
{ int teller = 0;     // bereken openingshaakjes min sluithaakjes
  int r = l+1;

  while (teller > 0 || A[r] != ')' )    //
  { if (A[r] == '(')
      teller ++;
    else
      teller --;
    r++;
  }

  // nu is teller == 0 && A[r] == ')', dus we hebben
  // het corresponderende sluithaakje.
  return r;
}
```

00.00

(c)
```
int totaleNesting (string A, int l)
{ int som = 0

  if (l < A.size() && A[l] != ')')
                    // A[l] moet dan openingshaakje zijn
  { int r = sluithaakje (A, l);
    som += (r - l - 1) / 2;  // er zijn r-l-1 posities, dus haakjes
                             // tussen posities l en r;
                             // deel door 2 voor de haakjesparen
    som += totaleNesting (A, l+1) + totaleNesting (A, r+1);
                  // tussen l en r in,     na r
  }
  return som;
}
```

00.13

2(a)
Zagen op plekken 1, 2, 3: 23 + 18 + 10 = 51
Zagen op plekken 3, 1, 2: 23 + 20 + 15 = 58

00.17

(b)

```
void berekenL ( int n, int t[] )
{
  for (int i = 1; i <= n; i++)
  { L[i][i] = t[i];    // diagonaal
    for (int j = i+1; j <= n; j++)    // verderop in rij i:
      L[i][j] = L[i][j-1] + t[j];     // steeds een t[j] meer.
  }
}
```

M(i,i) = 0

00.22 We moeten stukken i t/m j in losse stukken zagen.

(c) Als i = j dan bestaat de deelstam die we in stukken willen zagen alleen uit stuk i. Er valt dus niets te zagen. Dus ...

Als i < j, valt er wel iets te zagen. We voeren de eerste zaagoperatie uit op positie k = i, of k = i+1, of k = i+2, ..., of k = j-1.

direct na stuk i

direct na stuk i+1

direct na stuk [i+2]

direct na stuk j-1, dus vóór stuk j.

Deze eerste zaagoperatie kost L(i,j) euro.
Daarna moeten we nog de deelstam bestaande uit stukken i t/m k en de deelstam bestaande uit stukken k+1, t/m j in stukken zagen. Dit kost minimaal M(i,k) + M(k+1,j).
Omdat we minimale kosten willen, kiezen we die k, waarbij deze som minimaal is. Deze minimale som tellen we dus op bij de vaste kosten L(i,j) van de eerste zaagoperatie.

00.36

(d)

| M(i,j) | j = 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| i = 1 | 0 | 13 | 33 | 46 |
| 2 | | 0 | 15 | 28 |
| 3 | | | 0 | 10 |
| 4 | | | | 0 |

00.40

(e) We berekenen M(i,j) diagonaal voor diagonaal:
eerst de hoofddiagonaal met j = i,
dan de diagonaal ernaast met j = i+1,
dan de diagonaal daarnaast met j = i+2, enz.
We berekenen M(i,j) binnen een diagonaal van linksboven naar rechtsonder.

```
int bepaalMinKostenBU (int n);
{ int M [n+1][n+1];

  for (int i=1; i≤ n; i++)
    M [i][i] =0;     // j=i

  for (int d=1; d≤ n-1; d++)  // d is j-i, en bepaalt diagonaal
  { for (int i=1; i≤ n-d; i++)
    { int j = i+d;
      // bereken nu M [i][j]. N.B: inderdaad is i<j
      int mini = M [i][i] + M [i+1][j]; // initialisatie, k=i
      for (int k=i+1; k≤ j-1; k++)
      { int tmp = M [i][k] + M [k+1][j];
        if (tmp < mini)   // beter minimum
          mini = tmp;
      }
      M [i][j] = L [i][j] + mini;
    } // for i
  } // for d

  return M [1][n]
} // bepaalMinKostenBU
```

00.54

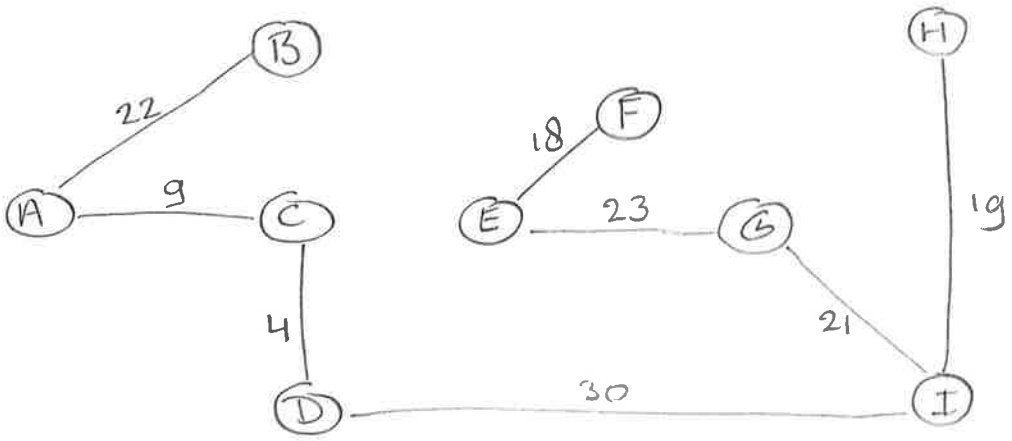
3 (a) Een minimale opspannende boom van $G_1$ is
* een boom (en dus acyclisch)
* bestaande uit takken van $G_1$
* die alle knopen van $G_1$ bevat
* en minimaal totaal gewicht heeft.

00.58

(b) Bekijk: C – D (minimaal gewicht): voeg toe, want geeft geen kring
A – C, voeg toe
A – D, voeg niet toe, want zou kring A–D–C–A geven
E – F, voeg toe
H – I, voeg toe
G – I, voeg toe
A – B, voeg toe
E – G, voeg toe
F – H, voeg niet toe, want zou kring F–H–I–G–E–F geven

G-H, voeg niet toe, want zou kring G-H-I-G geven
D-I, voeg toe
Klaar, want 8 takken toegevoegd, terwijl n=9

Resulterende boom




01.10
(c)

```
bool bereikbaar (int a, int b, bool bezocht[], Buur *graaf[])
{
  bezocht[a] = true;
  if (a == b)  // bereikt ?
    return true;
  else
  { Buur *tmp = graaf[a]; // loop buren van a af
    bool bereikt = false;
    while (tmp != nullptr && !bereikt)
    { if (! bezocht[tmp -> knoopnummer])
        bereikt = bereikbaar (tmp -> knoopnummer, b,
                              bezocht, graaf)
      tmp = tmp -> volgende;
    }
    return bereikt;
  }
}
```

n knopen en

01.22 / 01.28
(d) Allereerst worden de $m$ takken van G, gesorteerd op gewicht.
Dat kan in tijd in $\Theta(m \cdot \log m)$
De while-lus kent maximaal m iteraties.
In elke iteratie van de while-lus wordt een depth-first search uitgevoerd in de graaf met takken uit $E_T$. Deze graaf bevat maximaal n-2 takken, want we stoppen als takteller = n-1.
De depth-first search heeft dan tijdcomplexiteit in $\Theta(n-2)$ vanuit knoop a
is in $\Theta(n)$

Het toevoegen van een tak aan ET binnen de while-lus kan in $\Theta(1)$ tijd en gebeurt $n-1$ keer.
De tijdcomplexiteit van de while-lus in de worst-case wordt dus in $\Theta(m \cdot n + n-1)$, is in $\Theta(m.n)$
De totale worst-case tijdcomplexiteit van het algoritme van Kruskal is dus in $\Theta(m \cdot \log m + m \cdot n)$, is in $\Theta(m.n)$
(er vanuitgaande dat $m \leq \frac{1}{2} n(n-1)$, dus zeker $m < n^2$, dus $\log m < \log n^2 = 2 \cdot \log n$).

1.40 / 1.48

4 (a) best-first branch-and-bound

(b) backtracking en best-first branch-and-bound

(c) exhaustive search en backtracking

1.52

10.14

2 (e) Alternatief antwoord, ook goed.
We berekenen $M(i,j)$ rij voor rij, van beneden (hoge i) naar boven (lage i), en binnen de rij van links naar rechts.

```
int bepaalMinKostenBU (int n)
{ int M[n+1][n+1];

  for (int i=n; i>=1; i--)
  { M[i][i] = 0;
    for (int j= i+1; j<=n; j++)    // berekenen M[i][j] met i<j
    { int mini = M[i][i] + M[i+1][j]; // initialisatie, k=i
      for (int k = i+1; k <= j-1; k++)
      { int tmp = M[i][k] + M[k+1][j];
        if (tmp < mini)    // beter minimum
          mini = tmp;
      }
      M[i][j] = L[i][j] + mini;
    } // for j
  } // for i
  return M[1][n];
} // bepaalMinKostenBU
```

10.23.

2 (e) Nog een alternatief antwoord, ook goed
We berekenen $M(i,j)$ kolom voor kolom van links naar rechts, en per kolom van beneden (hoge i) naar boven (lage i).