THEORIEGEDEELTE (max. 50 punten)

Opgave 1 – Complexiteit (max. 10 punten)

1.1

Geef de rekencomplexiteit **én** de geheugencomplexiteit van de volgende **recursieve** methode. Onderbouw je antwoorden. Gebruik de O()-notatie voor complexiteit.

```
double f (double x) {
     if (x <= 0)
          return 1.0;
     return x * (f(x-1) + f(x-2));
}
```

Rekencomplexiteit:
Functie roept zichzelf twee keer aan. Elke verhoging van x zorgt dus voor een verdubbeling in het aantal aanroepen. Dat wil zeggen O(2^N) = O(2^N) = **exponentieel**.
Geheugencomplexiteit:
De functie roept zichzelf twee keer **na elkaar** aan; dat vraagt niet om extra stackruimte. Maar je kunt zien dat als x met één stijgt, de aanroep-diepte met één toeneemt. Dit is **lineair** afhankelijk van x, dus O(N)

1.2

Geef de rekencomplexiteit **én** de geheugencomplexiteit van de volgende methode. Onderbouw je antwoorden. Gebruik de O()-notatie voor complexiteit. Bestudeer de code goed.

```
int f (int n) {
     int t = 0;
     for(int i = 0; i < n; i++) {
          for(int j = 0; j < 10; j++) {
               t += n;
          }
     }
    return t;
}
```

Rekencomplexiteit:
Binnenste lus is niet afhankelijk van n: O(1). buitenste lus is lineair: O(N). Resultaat = O(1 x N) = **O(N) = lineair**
Geheugencomplexiteit:
De functie is niet recursief en maakt geen extra variabelen of objecten voor een grotere n. Dus **O(1)**.

Opgave 2 – Arrays (max. 10 punten)

Stukje Java:

```
int[] anArray = new int[100];
ArrayList<Integer> aList = new ArrayList<>();
```

2.1

Beschrijf een situatie waarin je het best voor anArray kunt kiezen, en één waarin aList efficiënter is. Onderbouw je keuzes.

anArray: weinig overhead, lengte vooraf bekend, kan niet automatisch groeien

aList: iets meer overhead, groeit automatisch. Kun je ook eenvoudig vervangen door LinkedList

Opgave 3 - Hash (max. 10 punten)

3.1

Wat is de returnwaarde van de default Hash-methode van de klasse Object in Java?

De functie hashCode() geeft als default het adres van het object terug.

3.2

Waarom is deze niet goed bruikbaar?

Hoewel dit altijd verschillend is voor verschillende objecten (goed) is het ook verschillend voor objecten die qua inhoud hetzelfde zijn (fout). Voorbeeld: twee strings met dezelfde inhoud op een verschillend geheugenadres moeten dezelfde hashCode moeten geven.

3.3

Wat is de rekencomplexiteit van de methode() containsKey van de klasse HashMap? Verklaar je antwoord.

Zoeken naar een key in een hashmap die goed verdeeld is (dus zonder extreem lange buckets) is O(1). De positie in het array kan immers uitgerekend worden met hash/modulo, die in snelheid onafhankelijk is van de waarde van N. Maar “worst case” staat alles in één bucket en is de O() afhankelijk van de datastructuur die voor de buckets is gekozen, afhankelijk van de versie van Java of C#.

Opgave 4 - Generics (max. 10 punten)

Bekijk de volgende declaraties:

```
A) public Shape handleList(List<Shape> shapes);

B) public <T extends Shape> T handleList(List<T> shapes);

C) public Shape handleList(List<? extends Shape> shapes);

public class Circle extends Shape { ... }
```

4.1

Met welke van de drie kun je een lijst van het type List<Circle> verwerken? (meerdere antwoorden mogelijk). Leg per klasse uit waarom wel of waarom niet.
Deze hebben we in de les behandeld. Probeer het zelf nog eens.

4.2

Leg het verschil uit tussen de declaraties B en C.
B en C accepteren alle bij een lijst van Shape of een afgeleide klasse. Het verschil is dat C altijd een Shape retourneert; B geeft een object terug van de klasse van de elementen in de lijst die je als parameter meegeeft.

Opgave 5 - Bomen (max. 10 punten)

5.1

Beschrijf in woorden hoe je de diepte van een boom kunt bepalen. Schrijf eventueel voorbeeldcode ter verduidelijking.

De diepte van een boom is het langste pad van de wortel naar een blad. Je Zou dit recursief kunnen bepalen: de diepte van een boom = 1 + de diepte van zijn diepste kind