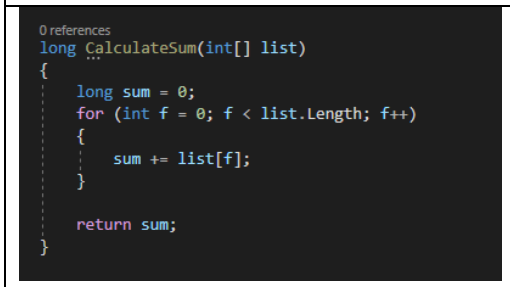
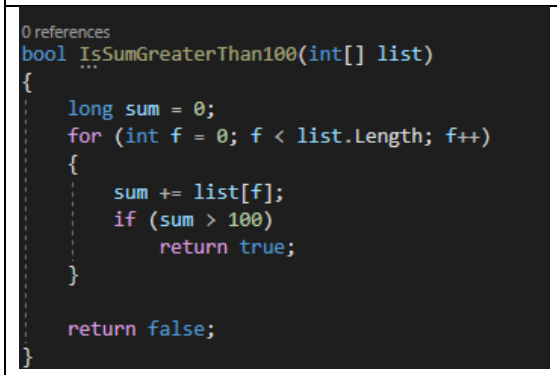
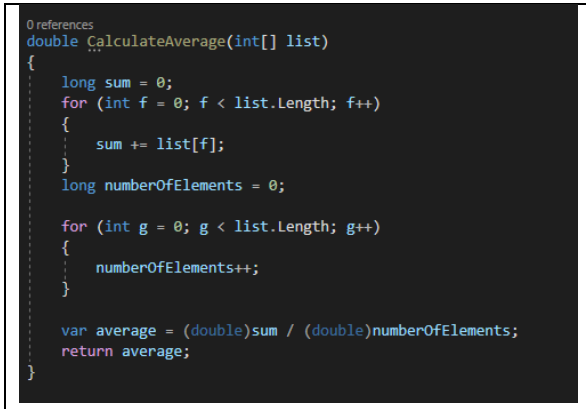
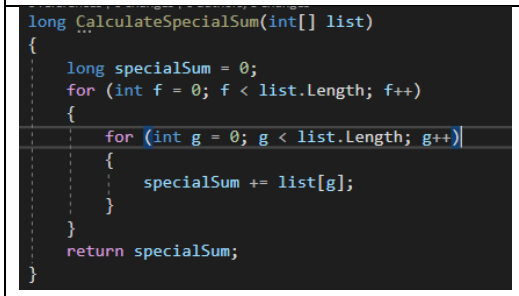
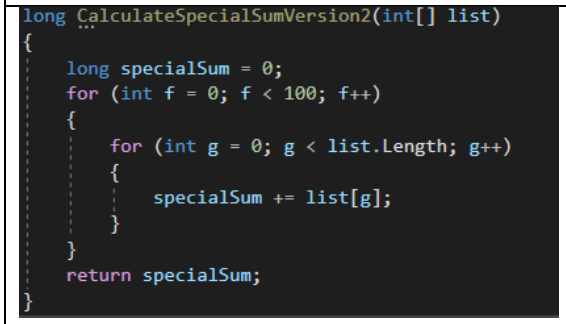
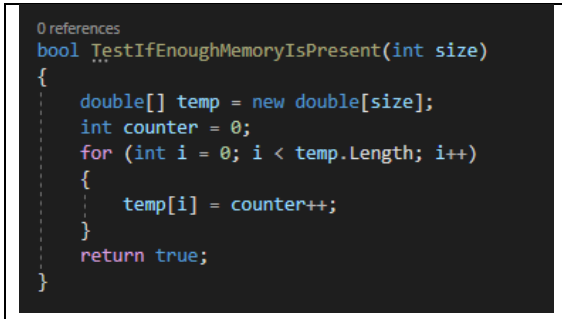
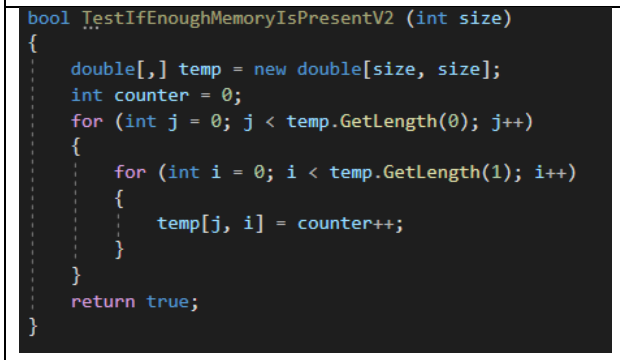
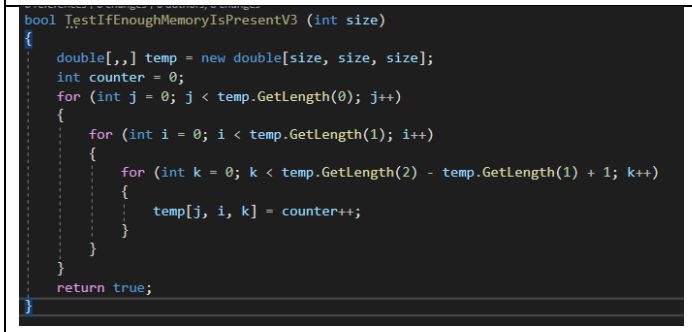
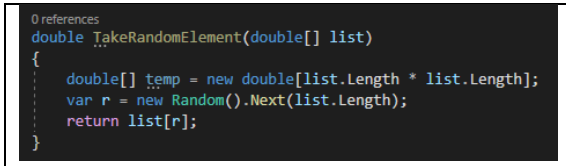
Datastructures, Oefeningen Algoritmes en Complexity

1. Geef aan welke time en space complexity deze algoritmes hebben. Ga voor de time complexity ervan uit dat elke instructie evenveel tijd in beslag neemt. Is het combinatie van meerdere verlopen, neem dan steeds de hoogste complexity.

Algoritme	tijdsverloop	Time complexity	Geheugen verloop	Space complexity
	Lineair (for lus duurt langer als in de input vergroot)	O(n)	Constant (2 var , type int)	O(1)
	Lineair (vergeet niet dat O handelt over "worst case", in het slechtste geval wordt telkens de volledige lijst doorlopen)	O(n)	Constant	O(1)

	**Lineair** (er zijn 2 for lussen die vergroten in functie van de input, maar het blijft een lineair verloop omdat ze na elkaar komen)	**O(n)**	**Constant** (5 variabelen van een basic type , de hoeveelheid geheugen verbruik wordt bepaald door het type en staat los van de input)	**O(1)**
	**Kwadratisch** (er zijn 2 for lussen in elkaar -nested loops – dus de tijd loopt length * length op)	**O(n^2)**	**Constant**	**O(1)**
	**Lineair** (2 nested for loops, maar de buitenste heeft een constante tijd, ze gaat steeds 100 iteraties doen, dus enkel de binnenste gaat verhogen)	**O(n)**	**Contant**	**O(1)**

	Lineair	O(n)	Lineair (er wordt een array aangemaakt die groter wordt als de input waarde verhoogt)	O(n)
	Kwadratisch (2 nested for loops)	$O(n^2)$	Kwadratisch (de temp array is een 2-dim array die een grootte (size*size) krijgt en dus kwardatisch groter wordt in functie van de input)	$O(n^2)$
	Kwadratisch verloop (3 nested loops, waarvan er echter maar 2 verlengen in functie van de input, de 3e heeft een constant tijdsverloop)	$O(n^2)$	Derde machtsverloop(cubic) Aangezien er een 3-dim array wordt aangemaakt die vergroot met de input volgens size*size*size	$O(n^3)$

	**Constant** verloop	O(1)	**Kwadratisch** verloop	$O(n^2)$
Bekijk de complexity van jouw stack versie 1, • Methode push • Methode pop	Beide een **constant** verloop	O(1)	Beide een **constant** verloop	O(1)
Bekijk de complexity van jouw stack versie 2 • Methode push • Methode pop	Push: **lineair** Pop: **constant**	O(n) O(1)	Push: **lineair** Pop: **constant**	O(n) O(1)
Bekijk de complexity van jouw lineaire queue versie 1 • Enqueue • Dequeue	Enqueue: **constant** Dequeue: **lineair**	O(1) O(n)	Beide **Constant**	O(1)
Bekijk de complexity van jouw lineaire queue versie 2 • Enqueue • Dequeue	Enqueue: **lineair** Dequeue: **lineair**	O(n) O(n)	Enqueue: **lineair** Dequeue: **constant**	O(n) O(1)
Bekijk de complexity van jouw circulaire queue versie 1 • Enqueue • Dequeue	Beide **Constant**	O(1)	Beide **Constant**	O(1)
Bekijk de complexity van jouw circulaire queue versie 2 • Enqueue • Dequeue	Enqueue : **Lineair** Dequeue: **Constant**	O(n) O(1)	Enqueue: L**ineair** Dequeue: C**onstant**	O(n) O(1)