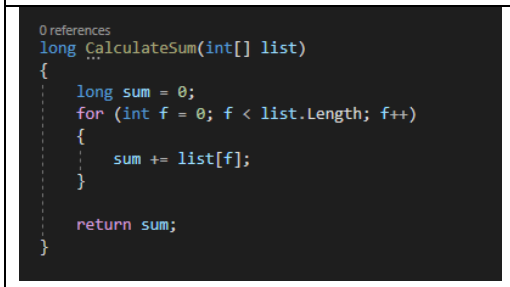
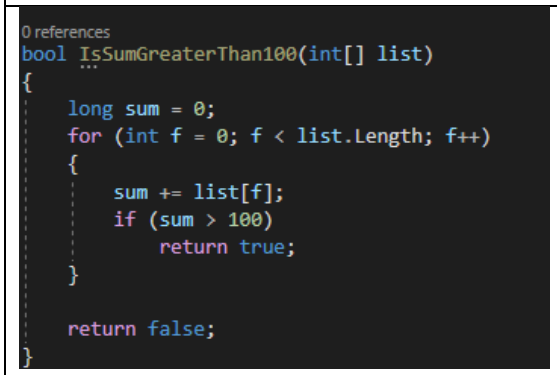
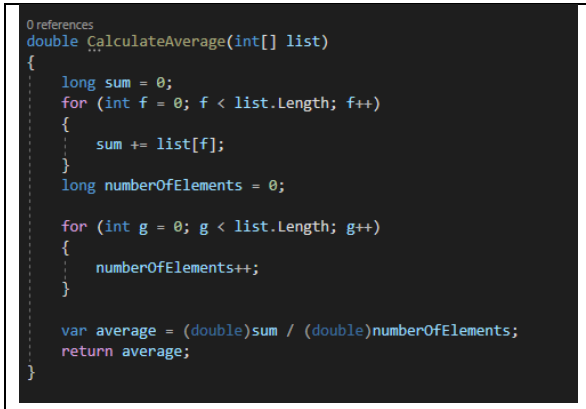
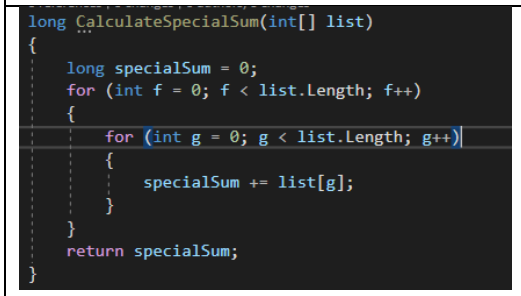
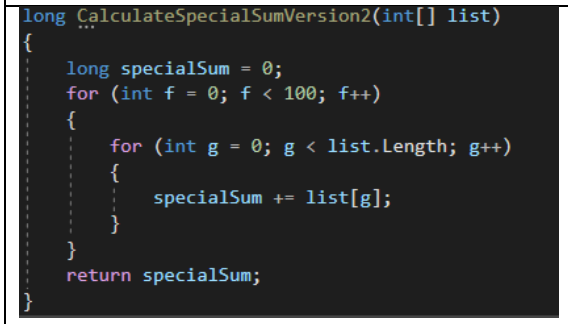
Datastructures, Oefeningen Algoritmes en Complexity

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	Timecomplexity	Geheugen verloop	Space complexity
				
				

```
0 references
long CalculateSum(int[] list)
{
    long sum = 0;
    for (int f = 0; f < list.Length; f++)
    {
        sum += list[f];
    }

    return sum;
}
```

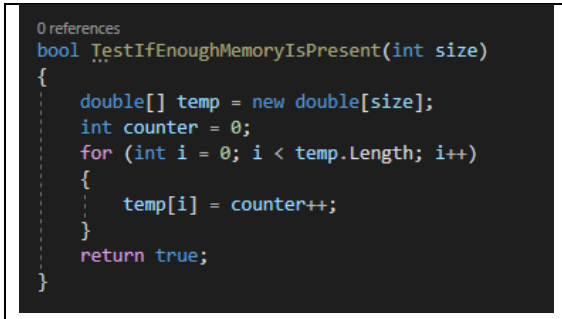
```
0 references
bool IsSumGreaterThan100(int[] list)
{
    long sum = 0;
    for (int f = 0; f < list.Length; f++)
    {
        sum += list[f];
        if (sum > 100)
            return true;
    }

    return false;
}
```

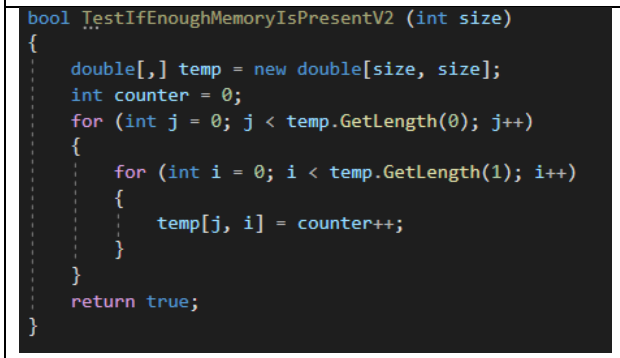
				
				
				

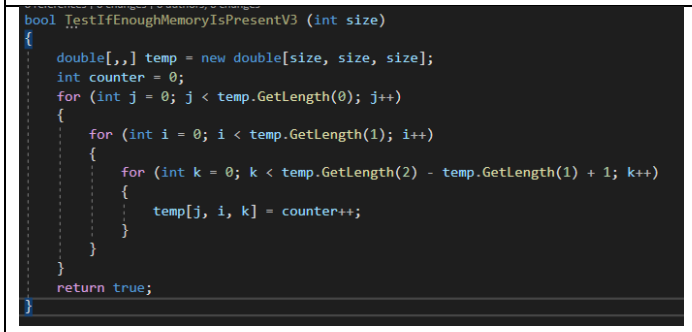
0 references
double CalculateAverage(int[] list)
{
long sum = 0;
for (int f = 0; f < list.Length; f++)
{
sum += list[f];
}
long numberOfElements = 0;
for (int g = 0; g < list.Length; g++)
{
numberOfElements++;
}
var average = (double)sum / (double)numberOfElements;
return average;
}

long CalculateSpecialSum(int[] list)
{
long specialSum = 0;
for (int f = 0; f < list.Length; f++)
{
for (int g = 0; g < list.Length; g++)
{
specialSum += list[g];
}
}
return specialSum;
}

long CalculateSpecialSumVersion2(int[] list)
{
long specialSum = 0;
for (int f = 0; f < 100; f++)
{
for (int g = 0; g < list.Length; g++)
{
specialSum += list[g];
}
}
return specialSum;
}

 0 references
bool TestIfEnoughMemoryIsPresent(int size)
{
double[] temp = new double[size];
int counter = 0;
for (int i = 0; i < temp.Length; i++)
{
temp[i] = counter++;
}
return true;
}				

 bool TestIfEnoughMemoryIsPresentV2 (int size)
{
double[,] temp = new double[size, size];
int counter = 0;
for (int j = 0; j < temp.GetLength(0); j++)
{
for (int i = 0; i < temp.GetLength(1); i++)
{
temp[j, i] = counter++;
}
}
return true;
}				

 bool TestIfEnoughMemoryIsPresentV3 (int size)
{
double[,,] temp = new double[size, size, size];
int counter = 0;
for (int j = 0; j < temp.GetLength(0); j++)
{
for (int i = 0; i < temp.GetLength(1); i++)
{
for (int k = 0; k < temp.GetLength(2) - temp.GetLength(1) + 1; k++)
{
temp[j, i, k] = counter++;
}
}
}
return true;
}				

0 references
`double TakeRandomElement(double[] list)`
`{`
`  double[] temp = new double[list.Length * list.Length];`
`  var r = new Random().Next(list.Length);`
`  return list[r];`
`}`				

Bekijk de complexity van jouw stack versie 1, • Methode push • Methode pop				
Bekijk de complexity van jouw stack versie 2 • Methode push • Methode pop				
Bekijk de complexity van jouw lineaire queue versie 1 • Enqueue • Dequeue				
Bekijk de complexity van jouw lineaire queue versie 2 • Enqueue • Dequeue				
Bekijk de complexity van jouw circulaire queue versie 1 • Enqueue • Dequeue				
Bekijk de complexity van jouw circulaire queue versie 2 • Enqueue • Dequeue				