Aufgabe:

Ein Casinobesitzer bat uns bei seinem neuesten Spiel um Hilfe. Sein Spiel funktioniert wie folgt: Jeder Teilnehmer zahlt eine Einstiegsgebühr von $25 und nennt eine Glückszahl zwischen 1 und 1000. Der Casinobesitzer nennt im Anschluss 10 Zahlen. Allen Teilnehmenden wird die Differenz zwischen ihrer genannten Glückszahl und der nächstgelegenen Zahl des Casinobesitzers ausgezahlt.

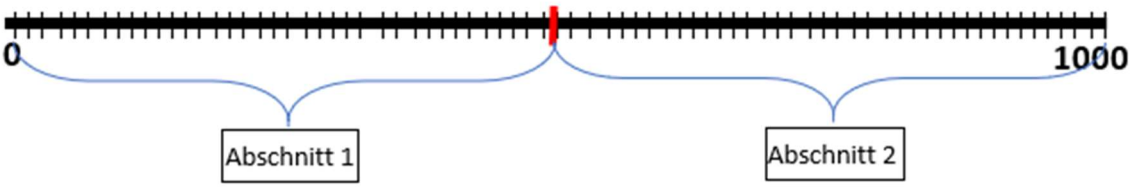
Unser Programm soll die Glückszahlen aller Teilnehmer einlesen und daraufhin die Zahlen identifizieren, bei denen die Summe dieser Differenzen am kleinsten ist, der Gewinn des Casinobesitzers also am Größten.

Kleines Beispiel: Ein Teilnehmer nennt die Zahl 10 und die nächstgelegene Zahl des Casinobesitzers ist die 15, dann werden dem Teilnehmer $5 ausgezahlt.

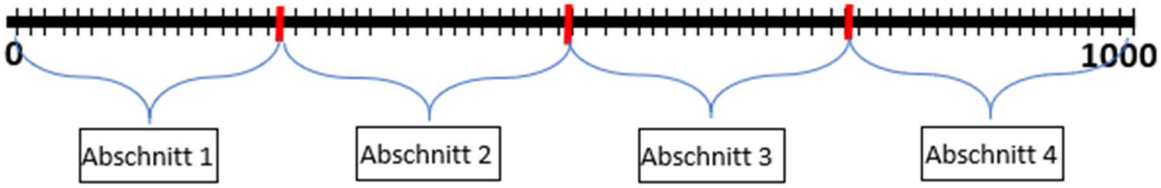
Lösungsidee:

Die Zahl, die am nächsten an allen Zahlen in einer Liste ist, ist der Durchschnitt aller Zahlen in dieser Liste. Wir sollen aber nicht nur eine Zahl finden, sondern zehn. Die erste Aufgabe besteht daher darin, die Liste in zehn Abschnitte einzuteilen, deren Durchschnitte den größtmöglichen Gewinn ergeben.

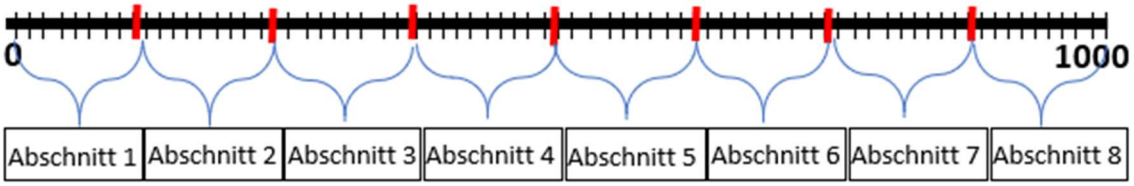
Unser Ausgangspunkt ist der Durchschnitt aller Glückszahlen. Dieser Durchschnitt bestimmt die ersten beiden Abschnitte. Im zweiten Schritt berechnen wir die Durchschnitte von Abschnitt eins und Abschnitt zwei.



Jetzt haben wir aber immer noch nur zwei Abschnitte, von denen wir die Durchschnitte nehmen können. Also teilen wir einfach noch einmal:



Wenn wir noch einmal teilen, dann erhalten wir acht Abschnitte:

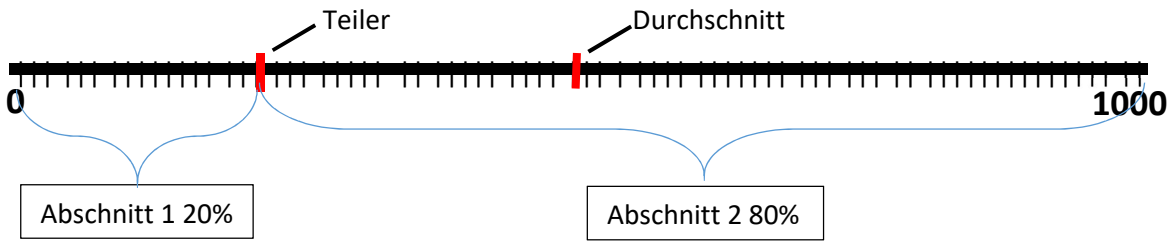


Das sind aber immer noch zwei Abschnitte zu wenig um zehn Zahlen errechnen zu können. Was aber, wenn wir diese Prozedur nur auf 80% der Zahlen anwenden? Und die restlichen 20% nur einmal teilen?

Eine fixe Teilung bei 200 oder 800 wäre unabhängig von der tatsächlichen Verteilung der Glückszahlen und würde daher nicht das beste Ergebnis für den Casinobesitzer liefern. Dies stellt uns vor das Problem, den besten Teiler zu finden.

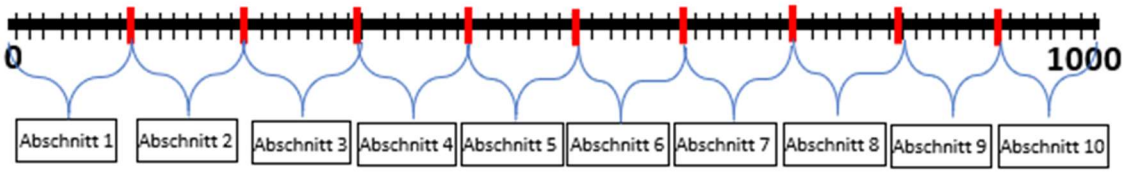
Bei einer Normalverteilung wäre die beste Teilstelle für das weitere Verfahren bei 200. Da wir aber von der tatsächlichen Verteilung ausgehen, verschieben wir den diese Teilstelle gemäß dem Durchschnitt der tatsächlich gewählten Glückzahlen: Wir teilen den Durchschnitt durch 5 und multiplizieren mit zwei.

Wenn wir dort teilen, erhalten wir, basierend auf dem Durchschnitt, im ersten Abschnitt ungefähr 20% der Glückszahlen.



Diesen ersten Abschnitt teilen wir nun, entsprechend dem oben geschilderten Verfahren, am Durchschnitt in zwei Abschnitte. Die Durchschnitte dieser beiden Abschnitte sind die ersten zwei der gesuchten zehn Zahlen.

Zur Identifikation der restlichen Zahlen teilen wir nun den zweiten Abschnitt wie oben beschrieben in acht Abschnitte. Deren Durchschnitte liefern die gesuchten Zahlen.



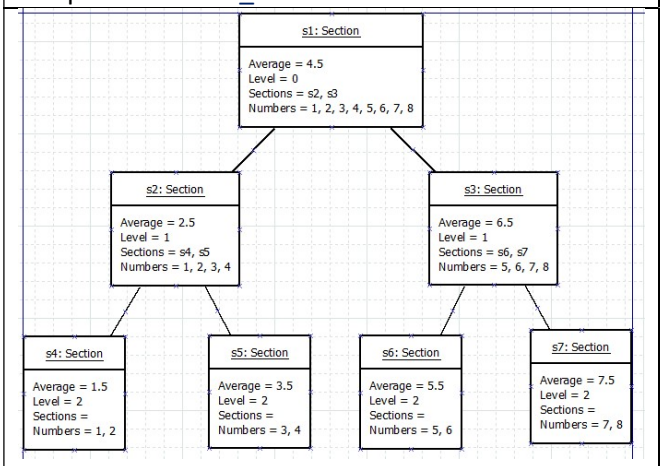
Umsetzung:

Für die Umsetzung in Quellcode haben wir uns für C# entschieden, da wir mit Hilfe der objektorientierten Programmierung einen für den Menschen natürliche Programmstruktur verwirklichen konnten. Auch ist die Benennung unserer Variablen und Objekte in Englisch (Abschnitt wird zu Section)

Für die Verwaltung unserer Abschnitte verwenden wir Objekte. Diese Objekte nennen wir Sections. Eine Section speichert eine List mit Integern, den Durchschnitt aller Zahlen in der List, und einen Integer ‚Level'.

Im Konstruktor von Section Objekten erstellen diese zwei weitere Section Objekte, denen aber jeweils nur die Hälfte der List übergeben wird (geteilt beim durchschnitt). In der ‚Level' Variable wird gespeichert wie oft sich ein Objekt schon geteilt hat (Siehe Beispiel 1). Abgebrochen wird dieses teilen, sobald die ‚Level' Variable den Wert von ‚MAX_DEPTH' erreicht hat, worauf diese Objekte ihren Durchschnitt in die statische List ‚Marker' speichern.

Beispiel 1 bei MAX_DEPTH = 2



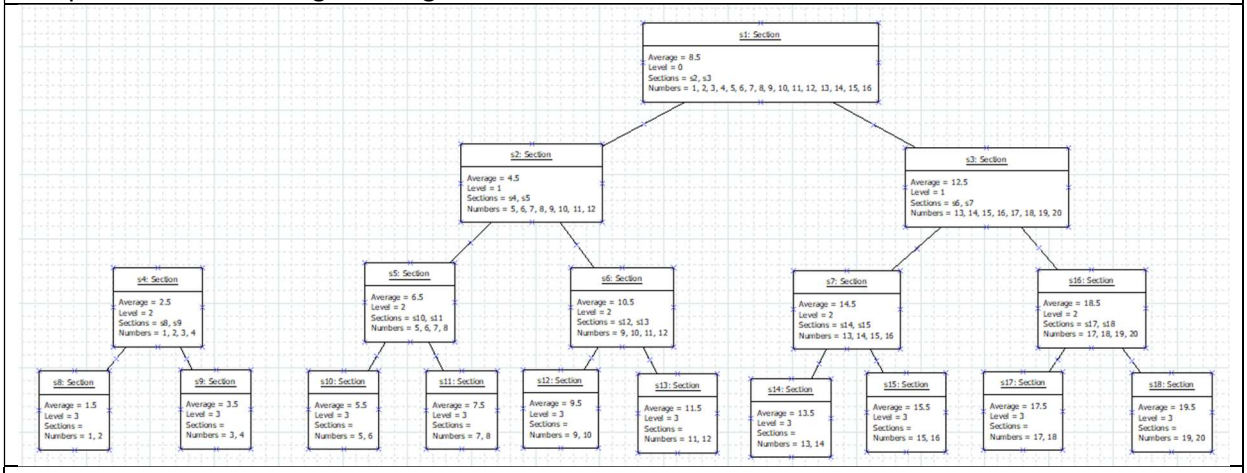
In der Marker Liste sind hier 1.5, 3.5, 5.5 und 7.5

In unserem Programm erstellen wir am Anfang zwei Abschnitte und setzen MAX_DEPTH auf 3.

Abschnitt 1 besteht aus 80% der Zahlen und beginnt bei einem Level von 0.

Abschnitt 2 geben wir 20%. Der zahlen und einen Level von 2.

Beispiel 2 bei Anwendung des Programms auf eine Liste mit 20 zahlen im Abstand von 1.



In der Marker Liste sind 1.5, 3.5, 5.5, 7.5, 9.5, 11.5, 13.5, 15.5, 17.5 und 19.5
Bzw. gerundet 2, 4, 6, 8, 10, 12, 14, 16, 18 und 20

Beispielaufgaben:

Beispiele	Ergebnisse										Gewinn
Beispiel 1	50	147	247	347	447	547	647	747	847	947	$25
Beispiel 2	46	134	267	366	432	549	695	783	868	942	$222
Beispiel 3	66	154	249	344	441	527	610	707	818	937	$161

Quellcode:

Section.cs

```
using System;
using System.Collections.Generic;

namespace BWINF_Aufgabe_3
{
    internal class Section
    {
        #region Attributes

        private readonly double Average;
                // Individual base value for the specific section marker
        private readonly byte Level;
                //Depth of "Split's"
        private const int MAX_DEPTH = 3;

        private readonly List<int> Numbers;
                //Stores all numbers in this section
        private readonly Section[] Sections = new Section[2];
                // Each Section hold 2 more Sections
        public static List<int> Markers { get; private set; } = new List<int>();
               // Average of the final Sections

        #region Konstruktor
        public Section(List<int> Numbers, byte Level)
        {
            this.Numbers = Numbers;
            Average = CalcAvg(Numbers);

            if (Level != MAX_DEPTH)
            {
                Tuple<List<int>, List<int>> tuple = Split();
                Sections[0] = new Section(tuple.Item1, (byte)(Level + 1));
                Sections[1] = new Section(tuple.Item2, (byte)(Level + 1));
            }
            else
            {
                Markers.Add((int)Math.Round(Average, 0));
            }
        }


        #region Methods

        /// <summary>
        /// Entry point for a chain reaction of creating 10 sections
        /// </summary>
        /// <param name="numbersList"></param>
        internal static void Create(List<int> numbersList)
        {
            Tuple<List<int>, List<int>> tuple = CutAt(numbersList, 20);
            Section Robin = new Section(tuple.Item1, 2); //Creates 2 Sections  (20%)
            Section Batman = new Section(tuple.Item2, 0); //Creates 8 Sections (80%)
        }

        private static Tuple<List<int>, List<int>>
            CutAt(List<int> numbersList, int v)
        {
            List<int> a = new List<int>();
            List<int> b = new List<int>();

            foreach (int num in numbersList)
            {
                if (num < (CalcAvg(numbersList) / 50) * v)
                {
```

```
                    a.Add(num);
                }
                else
                {
                    b.Add(num);
                }
            }
            return Tuple.Create(a, b);
        }

        private Tuple<List<int>, List<int>> Split()
        {
            List<int> a = new List<int>();
            List<int> b = new List<int>();

            foreach (int num in Numbers)
            {
                if (num < Average)
                {
                    a.Add(num);
                }
                else
                {
                    b.Add(num);
                }
            }

            return Tuple.Create(a, b);
        }

        /// <summary>
        /// Calculates the average of the given numbers in the list
        /// </summary>
        /// <param name="numbers"></param>
        /// <returns></returns>
        private static double CalcAvg(List<int> numbers)
        {
            if (numbers.Count != 0) // Else possible [Divide by Zero]
            {
                int sum = 0;
                foreach (int n in numbers)
                {
                    sum += n;
                }

                return sum / numbers.Count;
            }

            return 0;
        }

        #endregion Methods
    }
}
```