SWO3	**Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3**	**WS 2019/20, ÜZ 4**

☐ Gruppe M. Hava

☒ Gruppe J. Heinzelreiter Name: Neuhold Micheal Aufwand [h]: 7

☐ Gruppe P. Kulczycki Feedback von: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (15P + 35 P)			
2 (50 P)			

Beispiel 1: Longest Increasing Subsequence (`src/lis/`)

Gegeben sei eine beliebig lange, unsortierte Folge von positiven ganzen Zahlen und die Anzahl `n` der Elemente dieser Folge, in C dargestellt durch folgende Definitionen (mit Initialisierungen, die nur als Beispiel dienen):

```
#define MAX 100

                  // 0, 1, 2, 3, 4, 5, 6, 7, 8
int const s [MAX] = {9, 5, 2, 8, 7, 3, 1, 6, 4};
int const n       = 9;   // number of elements in s
```

Der *Longest Increasing Run* (LIR) ist die Länge der längsten zusammenhängenden Teilfolge (engl. *run*) von Elementen, deren Werte streng monoton steigend (engl. *increasing*) sind. Für das obige Beispiel mit den Werten `9,5,2,8,7,3,1,6,4` ergibt sich `2` (die beiden Läufe sind unterstrichen). Dieses Problem kann man in linearer Zeit lösen.

Die Berechnung der *Longest Increasing Subsequence* (LIS) besteht darin, die Länge der längsten nicht zusammenhängenden Teilfolge (engl. *subsequence*) monoton steigender Elemente zu ermitteln. Für das obige Beispiel `9,5,2,8,7,3,1,6,4` ergibt sich hierfür 3, wobei es (rein zufällig) wieder zwei solcher längsten Teilfolgen gibt, nämlich `2,3,6` und `2,3,4`.

Mit dynamischer Programmierung lässt sich auch dieses Problem lösen. Die zentrale Idee: Wenn die LIS für alle Anfänge der Gesamtfolge, also z.B. für `s[1..i-1]` bekannt ist, so ergibt sich die LIS für die um `1` längere Anfangsfolge durch Hinzunahme des Elements `s[i]`, indem die Länge der bisher längsten Teilfolge (also deren Maximum) um `1` erhöht wird, wenn diese mit einem Element `s[j] < s[i]` geendet hat. Diese Erkenntnis kann man in einem Feld `l` für alle Längen der Anfangsfolgen abbilden, dessen Elemente iterativ (für `i = 1..n`) wie folgt berechnet werden:

$$l_i = \max_{1 \le j < i} l_j + 1 \qquad \text{mit } s_j < s_i$$

Um die Elemente der längsten Teilfolge später auch rekonstruieren zu können bietet es sich an, neben dem Feld `l` für auch ein Feld `p` für den Index des jeweiligen Vorgängers (engl. *predecessor*) mitzuführen. Folgende Tabelle zeigt das Ergebnis für obiges Beispiel:

i	0	1	2	3	4	5	6	7	8
s[i]	9	5	2	8	7	3	1	6	4
l[i]	1	1	1	2	2	2	1	3	3
p[i]	-	-	-	1	1	2	-	5	5

Entwickeln Sie nun ein C-Programm, das die beiden Funktionen

```
int longest_increasing_run         (int const s [], int const n); // Bsp. 1a
int longest_increasing_subsequence (int const s [], int const n); // Bsp. 1b
```

implementiert. Die `main`-Funktion stellt im Wesentlichen einen Testreiber dar, teilen Sie Ihr C-Programm sinnvoll auf mehrere Module auf.

Beispiel 2: Teambuilding (`src/team/`)

Eine Disziplin in der Leichtathletik ist die 4 x 100-m-Staffel. Für Wettkämpfe (z.B. bei den olympischen Spielen) stellt jedes Teilnehmerland eine Staffel zusammen, die aus den besten LäuferInnen dieses Lands besteht. Gibt es genügend qualifizierte TeilnehmerInnen, so kann es auch noch eine zweite und eine dritte Staffel geben.

Sie sind Bundestrainer einer Leichtathletik-Nation und im Trainingscamp für die 4 x 100-m-Staffel zuständig. Damit Sie Ihre Viererteams auch gegeneinander unter interessanten Bedingungen antreten lassen können, möchten Sie aus den n TeilnehmerInnen (mit bekannten Bestzeiten über 100 m) $n/4$ Viererteams so zusammenstellen, dass die Siegchancen dieser Teams in den Trainingsläufen möglichst gleich sind. Dazu wird das arithmetische Mittel der Bestzeiten der vier Teammitglieder berechnet. Ziel ist es, die Standardabweichung der durchschnittlichen Bestzeiten aller Teams zu minimieren.

Entwickeln Sie einen Exhaustionsalgorithmus mit Optimierung und realisieren Sie diesen in einem C-Programm, das die Gruppeneinteilung errechnet und ausgibt.