

☐ Gruppe M. Hava☒ Gruppe J. HeinzelreiterName: Neuhold MichealAufwand [h]: 7☐ Gruppe P. Kulczycki

Feedback von: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (15P + 35 P)	100	100	100
2 (50 P)	95	95	90

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:

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

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.

Inhaltsverzeichnis

1. Sequences	2
1.1. <i>Lösungsidee</i>	2
1.1.1. Longest increasing run	2
1.1.2. Longest increasing subsequence	2
1.2. <i>Code</i>	3
1.3. <i>Testfälle</i>	7
2. Teambuilding	9
2.1. <i>Lösungsidee</i>	9
2.2. <i>Code</i>	9
2.3. <i>Testfälle</i>	12

1. Sequences

1.1. Lösungsidee

1.1.1. Longest increasing run

Den längsten monoton steigenden Lauf einer gegebenen Folge kann in linearer Zeit mit einem Schleifendurchlauf ermittelt werden. Dafür muss lediglich eine count- und eine previous_count Variable mitgeführt werden. Die Schleife durchwandert alle Elemente im Feld. Startindex ist 1, da immer auf das Element davor zugegriffen wird. In der Schleife wird geprüft, ob das vorherige Element kleiner ist als das derzeitige. Wenn dies der Fall ist, so wird die count Variable um eins erhöht. Sollte dies nicht der Fall sein, so wird geprüft, ob `count > previous_count` ist (`TRUE` → `previous count = count` || `FALSE` → `count = 1`).

1.1.2. Longest increasing subsequence

Als default kann für das erste Element als sub-sequence length immer 1 angenommen werden. Das erste Element besitzt außerdem keinen Vorgänger, daher kann hier der predecessor auf `INT_MIN` gesetzt werden. Grundsätzlich muss über alle Elemente der Sequence iteriert werden. Für jedes Element werden alle Elemente davor betrachtet und gecheckt, welches kleiner als das jetzige ist und die größte sub-sequence length hat. Im Anschluss wird max sequence length an der stelle i im Feld L gespeichert. Die Position vom predecessor wird ebenfalls in einem Feld abgelegt. Ist die Sequence fertig durchlaufen, so kann die Tabelle wie in der Angabe ausgegeben werden. Die sub-sequences können mit Hilfe des Feldes P rekonstruiert werden.

1.2. Code

Lis.c

```
#include "./subsequence.h"

int main() {

    /* ----- Testcase 1 ----- */
    int seq1[] = {9,5,2,8,7,3,1,6,4};
    int n1 = sizeof(seq1) / sizeof(int);
    Testing(seq1,n1);

    /* ----- Testcase 2 ----- */
    int seq2[] = {1,2,4,5,-1,0,3,7,8,11};
    int n2 = sizeof(seq2) / sizeof(int);
    Testing(seq2,n2);

    /* ----- Testcase 3 ----- */
    int seq3[] = {124,23,75,11,0,463,1,7426};
    int n3 = sizeof(seq3) / sizeof(int);
    Testing(seq3,n3);

    /* ----- Testcase 4 ----- */
    int seq4[] = {256,14,18,21,2,3,4,0,71};
    int n4 = sizeof(seq4) / sizeof(int);
    Testing(seq4,n4);

    return EXIT_SUCCESS;
}
```

Run.h

```
/* ----- longest increasing run ----- */
int longest_increasing_run (int const s [], int const n);
```

Run.c

```
#include "./run.h"

/* ----- longest increasing run ----- */
int longest_increasing_run (int const s [], int const n) {
    int count = 1;
    int prev_count = 0;

    for(int i = 1; i < n; i++) {
        if(s[i-1] < s[i]) count++;
        else {
            if(count > prev_count) prev_count = count;
            count = 1;
        }
    }
    return prev_count;
}
```

Sequence.h

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <limits.h>
#define MAX 100

/* ----- longest increasing subsequence ----- */
int longest_increasing_subsequence (int const s [], int const n);
```

Sequence.c

```
#include "./subsequence.h"
#include "./print_results.h"

/* ----- longest increasing subsequence ----- */
int longest_increasing_subsequence (int const s [], int const n) {

    int max, pred;

    int l[MAX];
    int p[MAX];

    l[0] = 1;          // count subsequence length (first elemnt always 1)
    p[0] = INT_MIN;    // first element does not have a predecessor

    for(int i = 0; i < n; i++) {
        max = 0;
        pred = INT_MIN; // no predecessor as default

        for(int j = 0; j < i; j++) { // for each i --
> iterate over the previos elements
            if((s[j] < s[i]) && (max < l[j])) {
                // new max and store location of predecessor
                max = l[j];
                pred = j;
            }
            l[i] = max + 1; // subsequence max value
            p[i] = pred;
        }
    }

    // get the longest subsequence
    max = INT_MIN;
    for(int i = 1; i < n; i++) {
        if(max < l[i])
            max = l[i];
    }
    print_result(s,l,p,n);
    return max;
}
```

Print_result.h

```
#include <limits.h>

/* ----- print result functions ----- */
void print_line();
void print_header(char msg[]);
void print_table_row(int a[], int n);
void print_result(int s[], int l[], int p[], int n);

/* ----- Test Function ----- */
void Testing(int seq[], int n);
```

Print_result.c

```
#include "print_results.h"

/* ----- print result functions ----- */

// separator
void print_line() {
    printf("\n");
    for(int i = 0; i < 50; i++){
        printf("-");
    }
    printf("\n");
    return;
}

// print header
void print_header(char msg[]) {
    print_line();
    printf("%s", msg);
    print_line();
    return;
}

// iterate over array elements
void print_table_row(int a[], int n) {
    for(int i = 0; i < n; i++) {
        if(a[i] == INT_MIN) printf(" ");
        else printf("%d ", a[i]);
    }
    print_line();
}

// print result table
void print_result(int s[], int l[], int p[], int n) {

    // print index
    printf("n: \t");
    for(int i = 0; i < n; i++) printf("%d ", i);
    print_line();
}
```

```

// print original sequence
printf("s[i]: \t");
print_table_row(s, n);

// print subsequenece length
printf("l[i]: \t");
print_table_row(l, n);

// print index of predecessor
printf("p[i]: \t");
print_table_row(p, n);
}

/* ----- Test Function ----- */
void Testing(int seq[], int n) {
    // longest increasing run
    print_header("<<< longest increasing run >>>");
    printf("max. run: %d\n", longest_increasing_run(seq,n));
    // longest increasing subsequence
    print_header("<<< longest increasing subsequence >>>");
    printf("max. sub-
sequence length: %d\n\n", longest_increasing_subsequence(seq,n));
}

```


1.3. Testfälle

Es wurde 4 verschiedene Sequenzen getestet. Zu jeder dieser Sequenzen wird der längste Lauf und die längste Sub-Sequenz ermittelt. Bei der Subsequenz wird außerdem die Tabelle erstellt. Mit dem mitgeführten Feld p kann im nachhinein die längste Folge wieder ermittelt und ausgegeben werden.

```

<<< longest increasing run >>>
max. run: 2

<<< longest increasing subsequence >>>
n:      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
max. sub-sequence length: 3
n) {

<<< longest increasing run >>>
max. run: 4

<<< longest increasing subsequence >>>
n:      0 1 2 3 4 5 6 7 8 9
s[i]:   1 2 4 5 -1 0 3 7 8 11
l[i]:   1 2 3 4 1 2 3 5 6 7
p[i]:   0 1 2 4 1 3 7 8
max. sub-sequence length: 7

```

```

-----
max. sub-sequence length: 4
-----
<<< longest increasing run >>>
-----
max. run: 2
-----
-----
max. run: 7
-----
<<< longest increasing subsequence >>>
-----
n: 0 1 2 3 4 5 6 7
-----
s[i]: 124 23 75 11 0 463 1 7426
-----
l[i]: 1 1 2 1 1 3 2 4
-----
p[i]: 1 2 3 4 1 2 3 5 6 7
-----
max. sub-sequence length: 4
-----
max. sub-sequence length: 7
-----

<<< longest increasing run >>>
-----
max. run: 3
-----
<<< longest increasing subsequence >>>
-----
n: 0 1 2 3 4 5 6 7 8
-----
s[i]: 256 14 18 21 2 3 4 0 71
-----
l[i]: 1 1 2 3 1 2 3 1 4
-----
p[i]: 2 3 1 2 3 1 4 3
-----
max. sub-sequence length: 4

```

2. Teambuilding

2.1. Lösungsidee

Für dieses Beispiel ist es notwendig alle möglichen Kombinationen der Läufer in Teams zu generieren. Dies geschieht mit einer rekursiven Funktion. Immer wenn eine Teamkombination zusammengesetzt wurde (in der Rekursion), wird geprüft ob die diese Kombination eine kleinere Standardabweichung hat als die Vorgänger. Ist dies der Fall, so wird die globale Variable „minStd“ (im Modul) durch den neuen Wert ersetzt. Immer Wenn ein Team vollständig mit Läufern angefüllt wurde (in der rekursiven Funktion), so wird der Startindex wieder zurückgesetzt, damit keine Läufer bei den Kombinationen ausgesetzt werden. Zusätzlich zu dem gibt es ein Feld im Modul, in dem die derzeitige Teamkonstellation mitgeführt wird. Ein Feld „is_used“ mit dem Elementdatentyp BOOLEAN wird verwendet, um zu checken, dass ein Läufer in einer Teamkonstellation nicht in mehreren Teams vorkommt.

Ist die Rekursion vollständig durchlaufen, so kann mit Hilfe einer Funktion die minStd (minimale Standardabweichung) ausgegeben werden. Dies geschieht mit einer Funktion, da die Variable minStd in einem Modul gekapselt ist und nicht von außen direkt erreichbar ist.

2.2. Code

Team.c

```
#include "teamlib.h"

int main() {

    /* ----- testcases ----- */
    double laeufer1[] = { 10.3 , 12.5 , 13.7 , 9.9 , 10.1 , 11.0 , 12.4 , 10.3 , 9.
7 , 10.7 , 11.2 , 20.9 , 20.6 };
    int n1 = sizeof(laeufer1) / sizeof(laeufer1[0]);
    double laeufer2[] = { 10 , 12 , 14 , 16 , 18 , 20 , 22 , 30 };
    int n2 = sizeof(laeufer2) / sizeof(laeufer2[0]);
    double laeufer3[] = { 18.3 , 14.2 , 9.1 , 8.0 , 9.13 , 13.3 , 22.4 , 29.1, 30.9
};
    int n3 = sizeof(laeufer3) / sizeof(laeufer3[0]);

    /* ----- find perfect solution ----- */
    combinations(laeufer1, n1, 0);
    printstd();

    combinations(laeufer2, n2, 0);
    printstd();

    combinations(laeufer3, n3, 0);
    printstd();
}
```

```
    return EXIT_SUCCESS;
}
```

Teamlib.h

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdio.h>
#include <math.h>
#include <limits.h>
#include <float.h>

#define TEAM_SIZE 4
#define MAX_RUNNERS 40
#define MAX_TEAMS MAX_RUNNERS / TEAM_SIZE

/* ----- help functions ----- */
double avg(double elem[],int start,int n);
double std_variation(double teams[], int n);

/* ----- find all combinations and update min. standard variation -----
- */
void combinations(const double runners[], const int n, int start);

/* ----- print final standard variation ----- */
void printstd(void);
```

Teamlib.c

```
#include "teamlib.h"

int team_count = 0;
int teams[MAX_RUNNERS] = { 0 };
double minStd = DBL_MAX;
bool is_used[MAX_RUNNERS] = { 0 };
double solution[MAX_RUNNERS] = { 0 };

/* ----- help functions ----- */

// returns avg with start and end index
double avg(double elem[],int start,int n) {
    double sum = 0;
    for(int i = start; i < start + n; i++){
        sum += elem[i];
    }
    return sum / n;
}

// calculate
```

```

double std_variation(double teams[], int n) {
    double mean = avg(teams,0,n);

    double tmp = 0;

    for(int i = 0; i < (n / TEAM_SIZE); i++ ) {
        tmp += pow((avg(teams, i * TEAM_SIZE, TEAM_SIZE) - mean),2) ;
    }
    tmp = tmp / (n / TEAM_SIZE);
    return sqrt(tmp);
}

/* ----- find all combinations and update min. standard variation -----
- */
void combinations(const double runners[], const int n, int start) {
    if((team_count % TEAM_SIZE == 0) && (n - team_count) < TEAM_SIZE) {
        double team_time[MAX_RUNNERS];
        for (int i = 0; i < team_count; i++)
        {
            team_time[i] = runners[teams[i]];
        }
        if(minStd > std_variation(team_time,team_count)) {
            minStd = std_variation(team_time,team_count);
        }
        return;
    } else if((team_count % TEAM_SIZE == 0) ) {
        start = 0;
    }

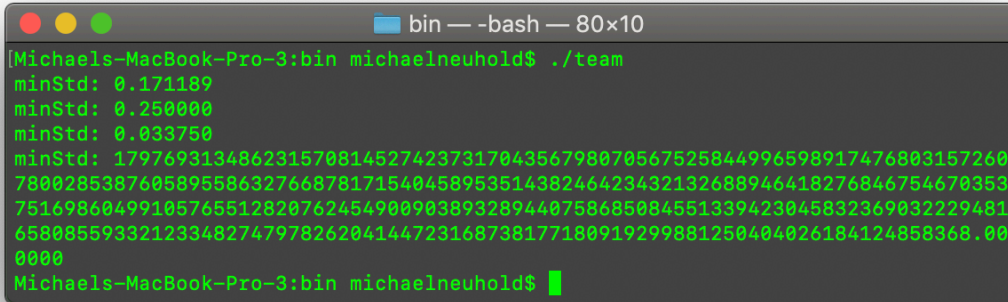
    for (int i = start; i < n; i++) {
        if(!is_used[i]) {
            teams[team_count++] = i;
            is_used[i] = true;
            combinations(runners, n, i + 1);
            team_count--;
            is_used[i] = false;
        }
    }
    return;
}

/* ----- print final standard variation ----- */
void printstd(){
    printf("minStd: %f\n", minStd);
    minStd = DBL_MAX;
}

```

2.3. Testfälle

Es wurden 3 „normale“ Testfälle erstellt, mit verschiedenen vielen Zeiten im Feld. Ausgegeben wird immer die minimale Standardabweichung der besten Teamkonstellation. Beim vierten Testfall wurde ein Feld ohne Elemente getestet. Hier wird der Wert von DBL_MAX ausgegeben, da minStd mit diesem Wert initialisiert wurde. Dieser Fehlerfall kann leicht mit einem IF ($n < 4$) → ERROR abgefangen werden, dies wurde allerdings aus Zeitnot nicht mehr implementiert.

A terminal window titled 'bin — -bash — 80x10' showing the execution of the './team' command. The output displays four 'minStd' values: 0.171189, 0.250000, 0.033750, and a very large number representing DBL_MAX (179769313486231570814527423731704356798070567525844996598917476803157260780028538760589558632766878171540458953514382464234321326889464182768467546703537516986049910576551282076245490090389328944075868508455133942304583236903222948165808559332123348274797826204144723168738177180919299881250404026184124858368.0000000000). The prompt returns to the shell.

```
bin — -bash — 80x10
[Michaels-MacBook-Pro-3:bin michaelneuhold$ ./team
minStd: 0.171189
minStd: 0.250000
minStd: 0.033750
minStd: 179769313486231570814527423731704356798070567525844996598917476803157260
78002853876058955863276687817154045895351438246423432132688946418276846754670353
75169860499105765512820762454900903893289440758685084551339423045832369032229481
65808559332123348274797826204144723168738177180919299881250404026184124858368.00
0000
Michaels-MacBook-Pro-3:bin michaelneuhold$
```