Inhaltsverzeichnis

[1. Sequences 2](#_Toc23753816)

[1.1. Lösungsidee 2](#_Toc23753817)

[1.1.1. Longest increasing run 2](#_Toc23753818)

[1.1.2. Longest increasing subsequence 2](#_Toc23753819)

[1.2. Code 3](#_Toc23753820)

[1.3. Testfälle 7](#_Toc23753821)

[2. Teambuilding 9](#_Toc23753822)

[2.1. Lösungsidee 9](#_Toc23753823)

[2.2. Code 9](#_Toc23753824)

[2.3. Testfälle 12](#_Toc23753825)

# Sequences

## Lösungsidee

### 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. Sollt dies nicht der Fall sein, so wird geprüft, ob count > previous\_count ist (TRUE 🡪 previous count = count || FALSE 🡪 count = 1).

### 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.

## 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

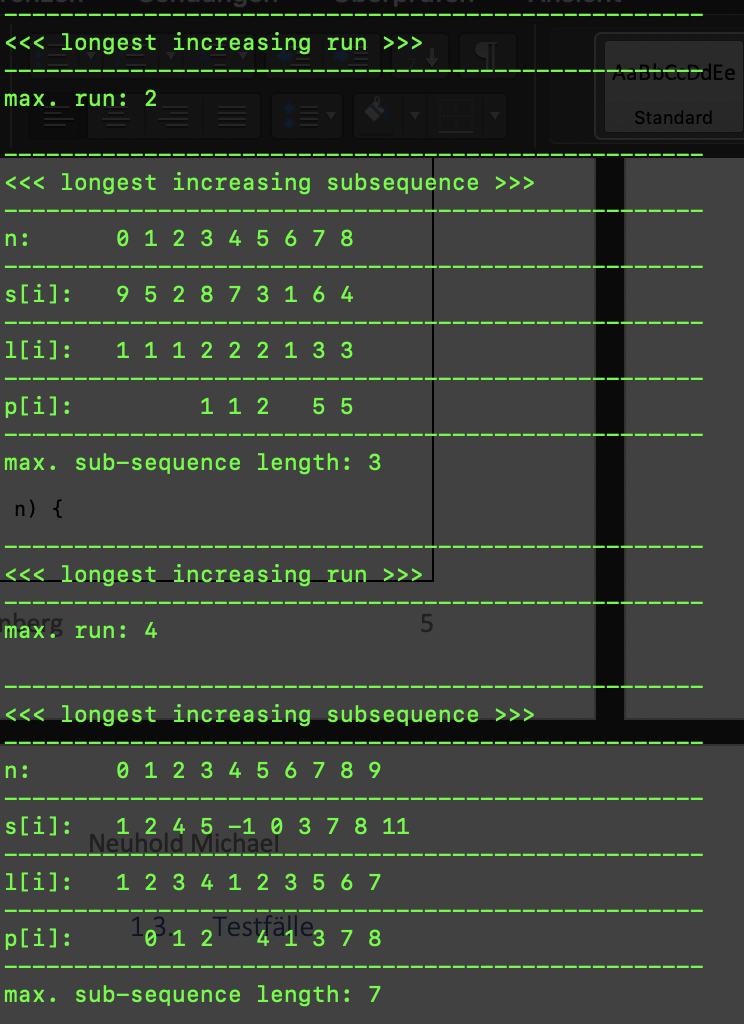
|  |
| --- |
| #include <limits.h>  /\* --------- print result functions --------- \*/  void print\_line();  void print\_header(char msg[]);  void print\_table\_row(int a[], int n);  void print\_result(int s[], int l[], int p[], int n);  /\* --------- Test Function --------- \*/  void Testing(int seq[], int n); |

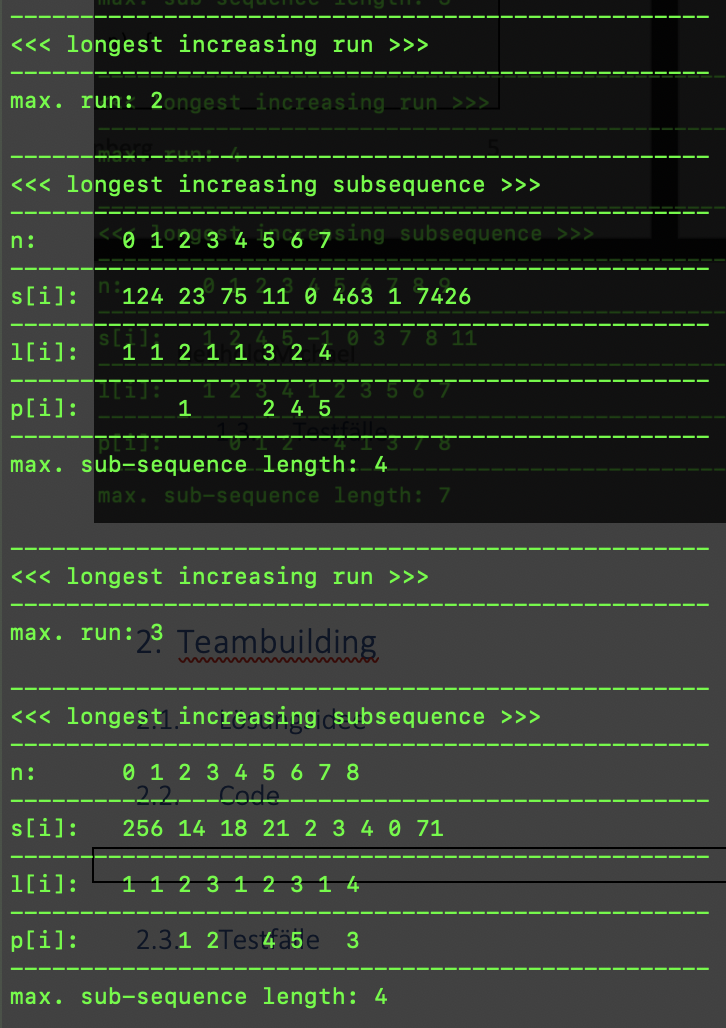
Print\_result.c

|  |
| --- |
| #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 subsequnece 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));  } |

## 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 die Tabelle erstellt. Mit dem mitgeführten Feld p kann im nachhinein die längste Folge wieder ermittelt und ausgegeben werden.





# Teambuilding

## Lösungsidee

Für dieses Beispiel ist es notwenigen 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.

## 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;  } |

## Testfälle

Es wurden 3 „normale“ Testfälle erstellt, mit verschieden 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 werde, dies wurde allerdings aus Zeitnot nicht mehr implementiert.

