Inhaltsverzeichnis

[1. Hammingfolge 2](#_Toc22512892)

[1.1. Lösungsidee 2](#_Toc22512893)

[1.2. Code 2](#_Toc22512894)

[1.3. Testfälle 3](#_Toc22512895)

[1.3.1. Verschiedene Grenzwerte mit Printf 3](#_Toc22512896)

[1.3.2. Verschiedene Grenzwerte ohne Printf 3](#_Toc22512897)

[1.3.3. Terminalausgabe 3](#_Toc22512898)

[2. i-t größtes Element 5](#_Toc22512899)

[2.1. Lösungsidee 5](#_Toc22512900)

[2.1.1. Lösungsidee second\_largest 5](#_Toc22512901)

[2.1.2. Lösungsidee ith\_largest\_1 5](#_Toc22512902)

[2.1.3. Lösungsidee ith\_largest\_2 5](#_Toc22512903)

[2.2. Code 6](#_Toc22512904)

[2.3. Testfälle 11](#_Toc22512905)

[3. Sortieren ganzer Zahlen 12](#_Toc22512906)

[3.1. Lösungsidee 12](#_Toc22512907)

[3.2. Code 12](#_Toc22512908)

[3.3. Testfälle 15](#_Toc22512909)

[3.3.1. Eingabe weniger als 10 Werte 15](#_Toc22512910)

[3.3.2. Eingabe mehr als 10 Werte 15](#_Toc22512911)

[3.3.3. Eingabe keiner Werte 15](#_Toc22512912)

# Hammingfolge

## Lösungsidee

Das Programm hamming\_sequence bekommt als Übergabeparameter eine Zahl Z. Diese Zahl gibt jene Stelle an bis zu der die Hammingfolge berechnet werden soll. Sollte der Benutzer keinen Grenzwert beim Programmstart mitgeben, so wird eine Fehlermeldung ausgegeben.

Der Algorithmus besteht grundsätzlich aus einer Zählschleife, die bis zur eingegebenen Maximalgrenze alle Zahlen durchgeht. Zu jeder Zahl wird mit einer Funktion („hamming()“) ermittelt ob es sich um eine Zahl der Folge handelt oder nicht. Ist sie Teil der Folge, so wird sie direkt in der Konsole ausgegeben. Der Funktion „hamming“ wird immer eine Zahl übergeben. In der Funktion wird geprüft, ob sie durch 5, 3 und 2 ohne Rest teilbar ist solange bis sie 1 ist. Sollte dies der Fall sein, so ist die Zahl Teil der Folge und die Funktion liefert True zurück.

## Code

|  |
| --- |
| Hamming\_sequence.c |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  // simple standard error function  void PrintErr(char msg[]) {    printf("<< ERROR (%s) >>\n", msg);  }  // returns true if x is a member of the hamming sequence  int hamming(int x) {    while(x % 5 == 0)      x = x / 5;    while(x % 3 == 0)      x = x / 3;    while(x % 2 == 0)      x = x / 2;    return x == 1;  }  int main(int argc, char \*argv[]) {    if(argc == 2){      double start = clock();      for(int i = 1; i <= atoi(argv[1]); i++){        if(hamming(i)) { // checking numbers          printf("%d ", i);        }      }      printf("\ntime: %f ms \n", ((clock() - start) / CLOCKS\_PER\_SEC) \* 1000);    } else {      PrintErr("Wrong number of parameters!");    }      return 0;  } |

## Testfälle

### Verschiedene Grenzwerte mit Printf

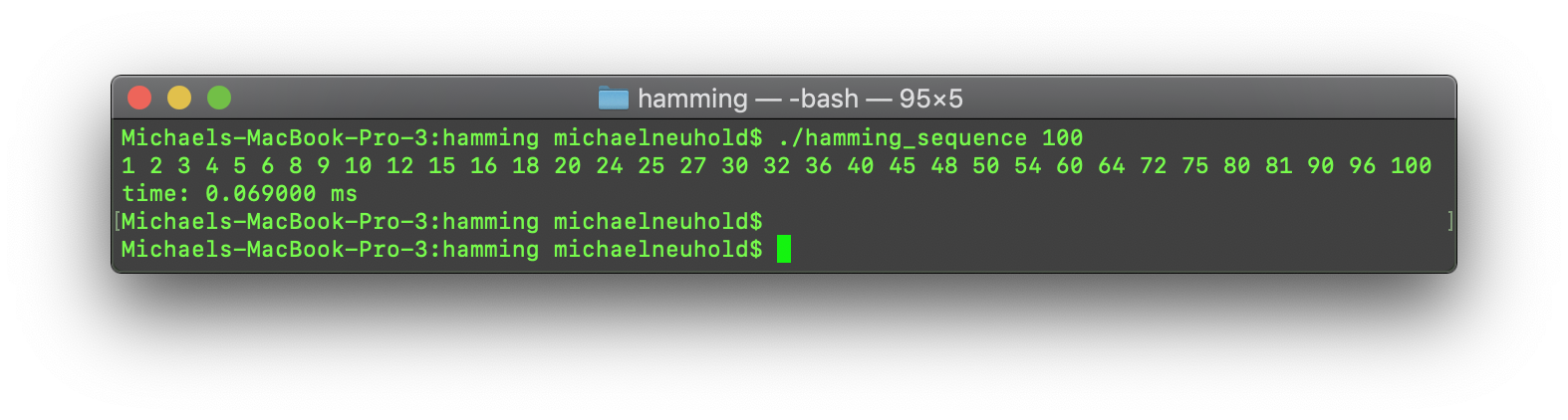
|  |  |
| --- | --- |
| 10 | 0.036 ms |
| 100 | 0.092 ms |
| 1000 | 0.148 ms |
| 5000 | 0.529 ms |
| 10000 | 1.023 ms |
| 1000000 | 102.522 ms |

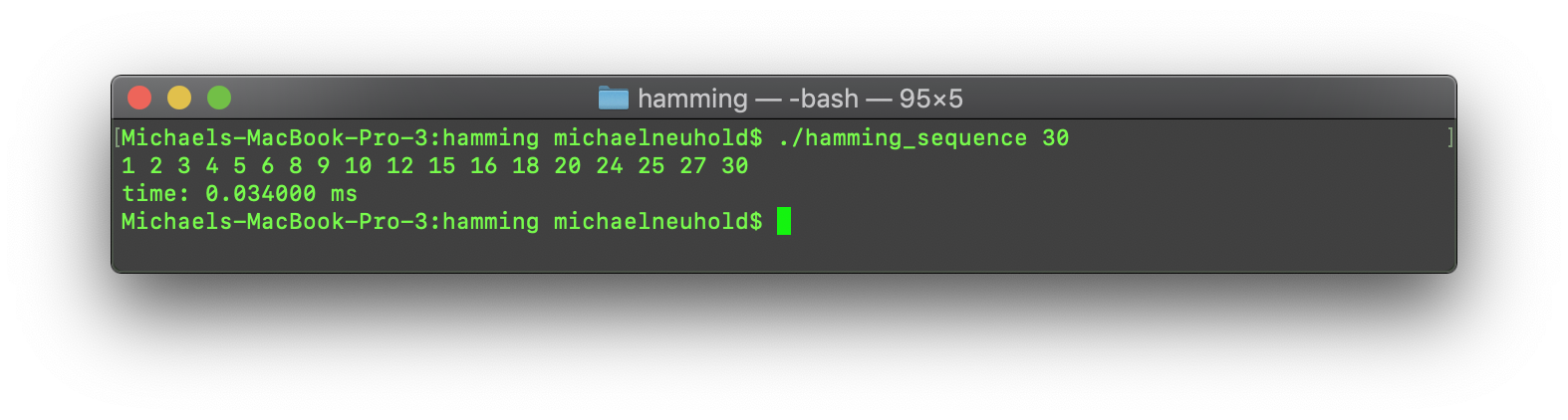
### Verschiedene Grenzwerte ohne Printf

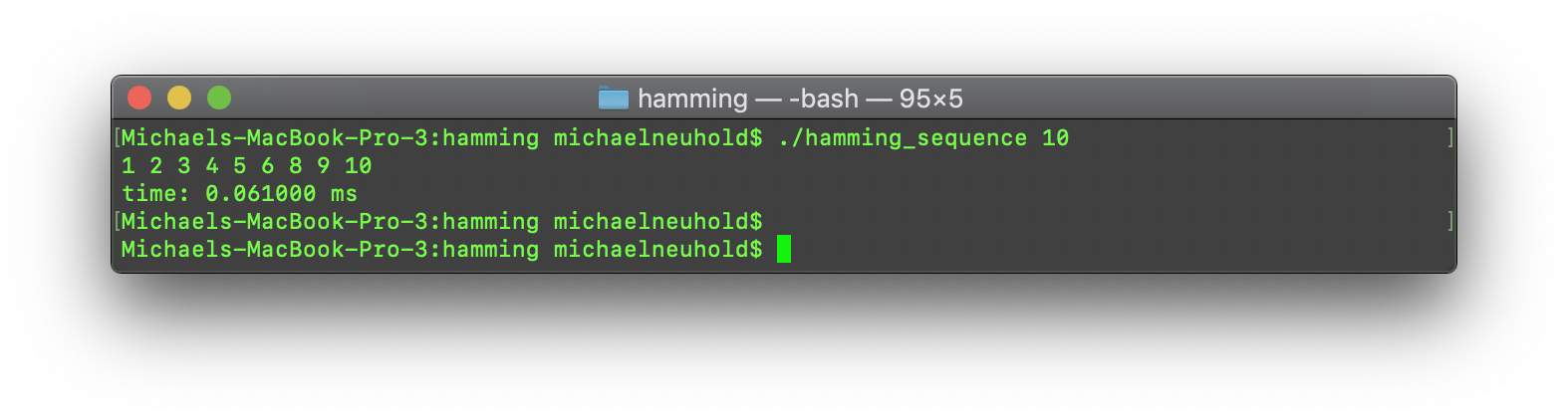
|  |  |
| --- | --- |
| 10 | 0.014 ms |
| 100 | 0.020 ms |
| 1000 | 0.096 ms |
| 5000 | 0.393 ms |
| 10000 | 0.941 ms |
| 1000000 | 86.974 ms |

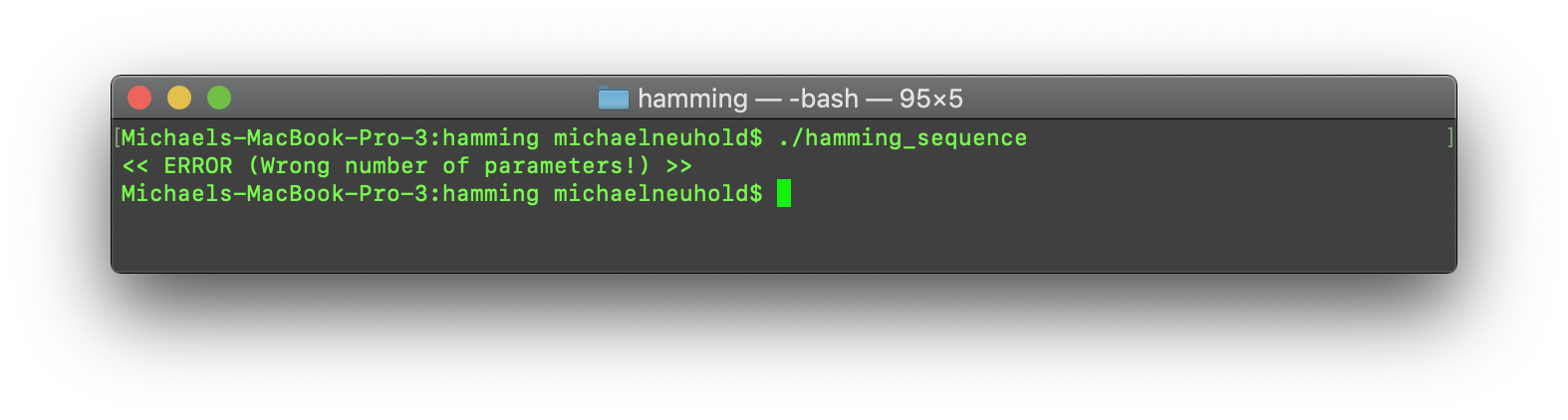
### Terminalausgabe

Die folgenden Screenshots zeigen einige Terminalausgaben des Programms.









# i-t größtes Element

## Lösungsidee

Für Testzwecke wird eine Feld fixer Größe mit der Funktion rand() befüllt. Im Anschluss werden die folgenden Funktionen darauf angewandt:

### Lösungsidee second\_largest

Um das zweitgrößte Element aus einem unsortierten Feld herauszufinden wird lediglich eine Zählschleife benötig, die das gesamte Feld einmal durchwandert. Zwei Variablen mit first\_largest and second\_largest werden mit dem kleinsten Integer Value initialisiert. Wenn In der Schleife dann ein Wert des Arrays betrachtet wird der größer als first\_largest ist, so wird der Wert von first\_largest der Variable second\_largest zugewiesen und first\_largest bekommt den neuen Wert vom aktuellen Feldelement zugewiesen. Ist jedoch die gerade betrachtete Zahl kleiner als first\_largest und größer als second\_largest, so wird lediglich second\_largest aktualisiert. Ist das Feld fertig durchlaufen, so kann einfach die Variable second\_largest auf der Konsole ausgegeben werden.

### Lösungsidee ith\_largest\_1

Für das Ausgeben des ith-größten Elements eines unsortierten Feldes, ist es möglich zunächst das Feld absteigend zu sortieren und anschließend einfach das Element mit dem Index i-1 auszugeben. Um das Feld zu sortieren wurde (wie in der Angabe gefordert) der Merge-Sort aus Beispiel 3 verwendet. Jedoch wurde statt aufwärts, abwärts sortiert. Dies wurde durch den Tausch des größer-kleiner Operators erzielt ;)

### Lösungsidee ith\_largest\_2

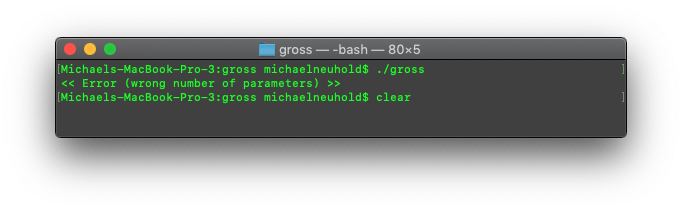
Zunächst wird vom Algorithmus geprüft, ob i = 1 ist. Wenn dem so ist, dann muss das größte Element im Feld zurückgegeben werden. In diesem Fall kann einfach eine lineare Suche durch das Feld verwendet werden. Sollte dies nicht der Fall sein, so kann man das Prinzip des Teilens heranziehen. Dafür wird ein sogenanntes Pivot-Element benötigt. Grundsätzlich gibt es einige Möglichkeiten ein Pivot-Element zu finden. Ich habe mich dafür entschieden, das erste, das letzte und das mittige Element des Feldes zu betrachten und diese im Anschluss in aufsteigender Reihenfolge zu sortieren. Retourniert wird im Anschluss das Mittige. Im Anschluss müssen die restlichen Werte auf der richtigen Seite im Feld „platz nehmen“. Auf diese Weise kann rekursiv vorgegangen werden, bis die Abbruchbedingung i == 1 erfüllt ist.

## Code

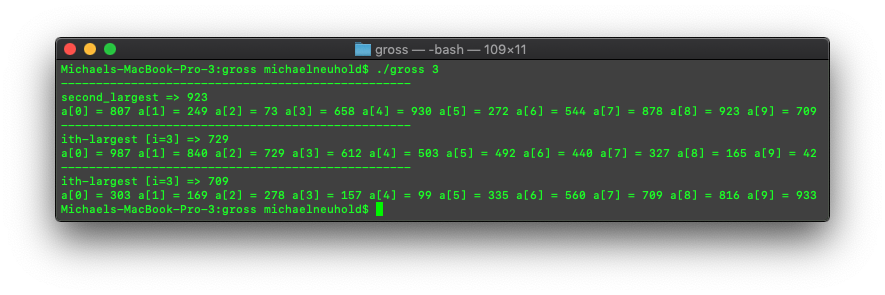
|  |
| --- |
|  |
| #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #define MAX 10  void printArr(int a[], int n) {    for(int i = 0; i < n; i++){      printf("a[%d] = %d ", i, a[i]);    }    printf("\n");    return;  }  void Line() {    for(int i = 0; i < 50; i++)      printf("-");    printf("\n");    return;  }  void InitArr(int a[], int n) {    for(int i = 0; i < n; i++){      a[i] = rand() % 1000;    }  }  // print error msg  void printErr(char msg[]) {    printf("<< Error (%s) >>\n", msg);    return;  }  /\* ================================================= \*/  // 2.1.  int second\_largest(int a[], int n) {    int f = INT\_MIN;  // first    int s = INT\_MIN;  // second    if(n >= 2) {      for(int i = 0; i < n; i++){        if(a[i] > f){          s = f;          f = a[i];        } else if (a[i] > s){          s = a[i];        }      }    } else {      printf("ERROR\n");    }    return s;  }  /\* ================================================= \*/  // 2.2.  void merge(int a[], int from, int mid, int to){    int tmp[10];    int k = 0;    int ai = from;    int aj = mid+1;    while(ai <= mid && aj <= to) {      if(a[ai] >= a[aj]){        tmp[k++] = a[ai++];      } else {        tmp[k++] = a[aj++];      }    }    // copy rest of first subarray    while(ai <= mid){      tmp[k++] = a[ai++];    }    // copy rest of second subarray    while(aj <= to){      tmp[k++] = a[aj++];    }      int i,j;    for(i=from,j=0;i<=to;i++,j++)      a[i]=tmp[j];  }  void merge\_sort\_rec(int a[], int from, int to) {    int mid;    if (from < to) {      mid = (from + to) / 2;      merge\_sort\_rec(a,from,mid);      merge\_sort\_rec(a,mid+1,to);      merge(a,from,mid,to);    } else {      return;    }  }  void merge\_sort (int a[], int n) {    merge\_sort\_rec(a,0,n-1);    return;  }  int ith\_largest\_1(int a[], int n, int i) {    merge\_sort(a,n);    return a[i-1];  }  /\* ================================================= \*/  // 2.3.  // swap values in array  void swap(int \*a, int \*b) {    int h = \*a;    \*a = \*b;    \*b = h;  }  // return selected pivo element  int pivot\_elem(int a[], int left, int right) {    // pc => pivot candidate    int pc[3] = {      a[left],      a[(left+right)/2],      a[right]    };    // sort candidates    if (pc[0] > pc[1]) {      swap(&pc[0], &pc[1]);    }    if (pc[1] > pc[2]) {      swap(&pc[1], &pc[2]);    }    if (pc[0] > pc[1]) {      swap(&pc[0], &pc[1]);    }    return pc[1];  }  // brings elements to the correct side of pivot  int split(int a[], int left, int right, int pivot) {    // split    do {      while (a[left] < pivot) {        left++;      }      while (a[right] > pivot) {        right--;      }      swap(&a[left], &a[right]);    } while (left < right);      // return position of pivot    if (a[left] == pivot) return left;    return right;  }  // search largest element in array (linear)  int get\_largest(int a[], int left, int right) {    int max\_value = a[left];    left++;    for (int i = left; i <= right; i++) {      if (max\_value < a[i]) {        max\_value = a[i];      }    }    return max\_value;  }  // serach ith-largest element  int search\_ith\_largest(int a[], int left, int right, int i) {    if(i == 1) {      return get\_largest(a, left, right);    }      // get value of pivot element (candidates: left, middle, right)    int pivot = pivot\_elem(a, left, right);    // split elements of array --> < / > than pivot and return pivot pos    int pivot\_pos  = split(a, left, right, pivot);    int right\_size = right - pivot\_pos + 1;    if (right\_size >= i) {      return search\_ith\_largest(a, pivot\_pos, right, i);    } else {      return search\_ith\_largest(a, left, pivot\_pos-1, i-right\_size);    }  }  int ith\_largest\_2(int a[], int n, int i) {    return search\_ith\_largest(a, 0, n-1, i);  }  /\* ================================================= \*/  int main (int argc, char \*argv[]) {      int a[MAX];      if(argc == 2) {      int ith = atoi(argv[1]);      // 1.      Line();      InitArr(a,MAX); // init with random numbers      printf("second\_largest => %d\n", second\_largest(a,MAX));      printArr(a,MAX);      // 2.      Line();      InitArr(a,MAX); // init with random numbers      printf("ith-largest [i=%d] => %d\n",ith, ith\_largest\_1(a,MAX,ith));      printArr(a,MAX);      // 3.      Line();      InitArr(a,MAX); // init with random numbers      printf("ith-largest [i=%d] => %d\n",ith, ith\_largest\_2(a,MAX,ith));      printArr(a,MAX);    } else {      printErr("wrong number of parameters");    }    return 0;  } |

## Testfälle

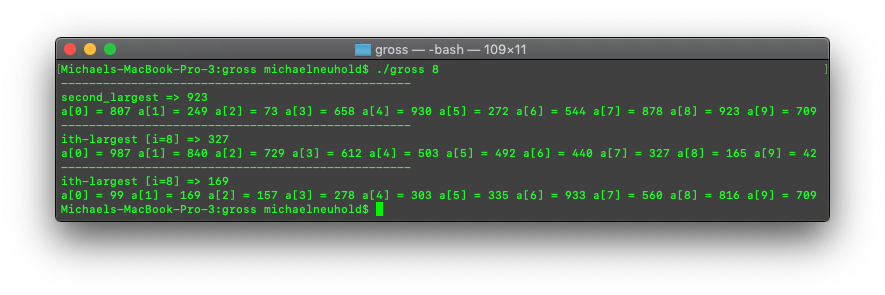
Dieser Testfall zeigt die den Programmaufruf ohne Parameterübergebe. Es ist jedoch ein Parameter notwendig für das ith größte Element. Somit wird eine Fehlermeldung ausgegeben.



Dieser Testfall zeigt eine valide Parameterübergabe. Es wird somit bei den Unterpunkten 2 & 3 jeweils das 3 größte Element der Felder ausgegeben. (Die Felder werden immer zwischen den Funktionsaufrufen mit zufälligen Zahlen initialisiert.



Der folgende Screenshot zeigt nur noch ein weiteres Beispiel mit einem anderen Parameterwert.



# Sortieren ganzer Zahlen

## Lösungsidee

Eingelesen werden die Parameter direkt über die Parameterliste beim Programmaufruf. Es sind lediglich 10 Elemente erlaubt. Wurden mehr als 10 Parameter eingegeben, so wird das Feld mit den ersten 10 Zahlen befüllt.

Um das Feld mit Merge-Sort zu sortieren ist eine Rekursive Funktion notwendig, die jeweils das übergebene Feld in zwei Teile zerlegt. Im Anschluss müssen die Teilfelder natürlich wieder zusammengefügt werden. Das Zusammenfügen basiert prinzipiell auf dem Reißverschlussprinzip. Es werden immer ein Element vom einen Teilfeld mit einem des anderen Teilfeld verglichen. Das kleinere Element wird in ein temporäres Array geschrieben. Der Index des Feldes, von dem das kleinere Element stammt wird im Anschluss um eins erhöht. Dieser Vorgang wird solange wiederholt, bis eines der Teilfelder fertig durchlaufen ist. Im Anschluss müssen noch die restlichen Elemente (falls welche übergeblieben sind) in das temporäre Array geschrieben werden. Zu guter Letzt wird das temporäre Array in das originale Feld (an der richtigen Stelle) zurückgeschrieben.

Die Prototypen (Funktionsdeklarationen) wurden in ein Header-File ausgelagert.

## Code

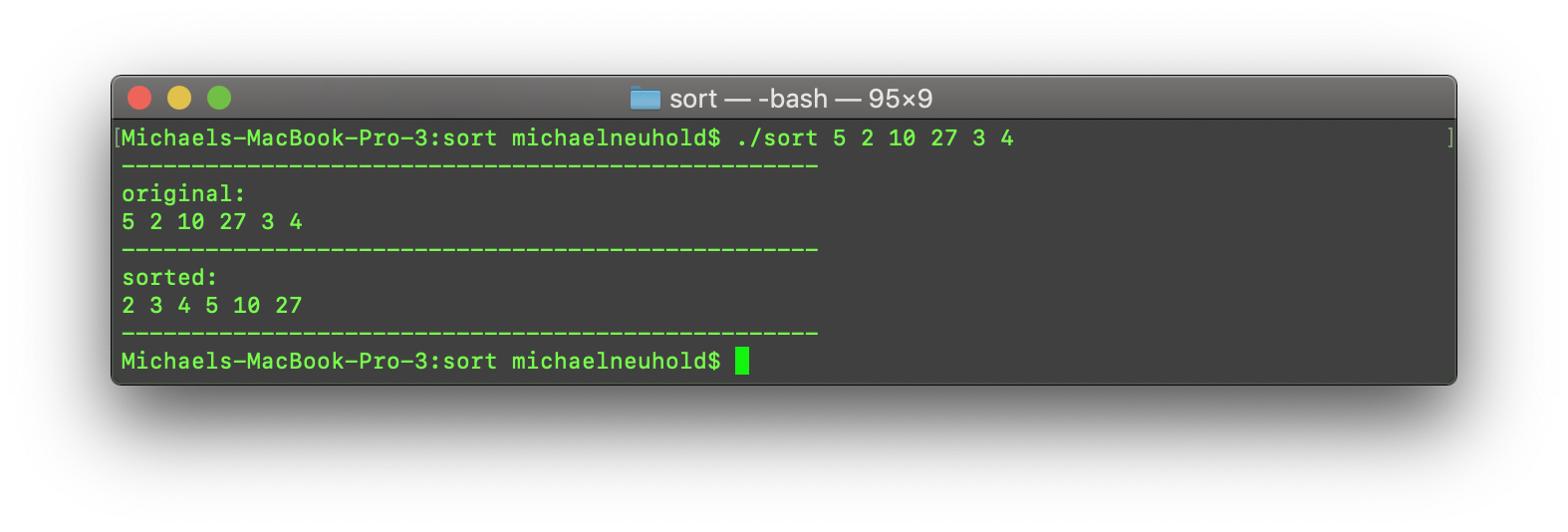
|  |
| --- |
| Sort.h |
| #ifndef sort\_h  #define sort\_h  // prototypes  void line(void);  void printArr(int a[], int n);  void printErr(char msg[]);  void merge(int a[], int from, int mid, int to);  void merge\_sort\_rec(int a[], int from, int to);  void merge\_sort (int a[], int n);  #endif |

|  |
| --- |
| Sort.c |
| #include <stdio.h>  #include <stdlib.h>  #include "sort.h"  #define MAX 10  // print error msg  void printErr(char msg[]) {    printf("<< Error (%s) >>\n", msg);    return;  }  // Separator  void line() {    for(int i = 0; i < 50; i++)      printf("-");    printf("\n");    return;  }  // simply print elements of array separated by spaces  void printArr(int a[], int n) {    for(int i = 0; i < n; i++)      printf("%d ", a[i]);    printf("\n");    return;  }  // merge two subarrays  void merge(int a[], int from, int mid, int to){    int tmp[MAX];    int k = 0;    int ai = from;  // start pos. first subarray    int aj = mid+1; // start pos. second subarray    while(ai <= mid && aj <= to) {      if(a[ai] <= a[aj]){        tmp[k++] = a[ai++];      } else {        tmp[k++] = a[aj++];      }    }    // copy rest of first subarray    while(ai <= mid){      tmp[k++] = a[ai++];    }    // copy rest of second subarray    while(aj <= to){      tmp[k++] = a[aj++];    }      // copy tmp into original array    int i,j;    for(i=from,j=0;i<=to;i++,j++)      a[i]=tmp[j];    return;  }  // mergesort  void merge\_sort\_rec(int a[], int from, int to) {    if (from < to) {      int mid = (from + to) / 2;      merge\_sort\_rec(a,from,mid);      merge\_sort\_rec(a,mid+1,to);      merge(a,from,mid,to);    }    return;  }  // call recursive mergesort  void merge\_sort (int a[], int n) {    merge\_sort\_rec(a,0,n-1);    return;  }  int main(int argc, char \*argv[]) {    int n;    int a [MAX] = {0};      if(argc > 1) {      // to set n to the actual number of values in a      if(argc-1 <= 10){        n = argc-1;      } else {        n = 10;      }      // code to read a maximum of MAX values from argv to a and      for(int i = 0; i < n; i++){        a[i] = atoi(argv[i+1]);      }        // code to display the unsorted array a      line();      printf("original:\n");      printArr(a,n);      // call sorting algorithmus      merge\_sort(a, n);      // code to display the sorted array a      line();      printf("sorted:\n");      printArr(a,n);      line();    } else {      printErr("there is nothing to sort!");    }    return EXIT\_SUCCESS;  } |

## Testfälle

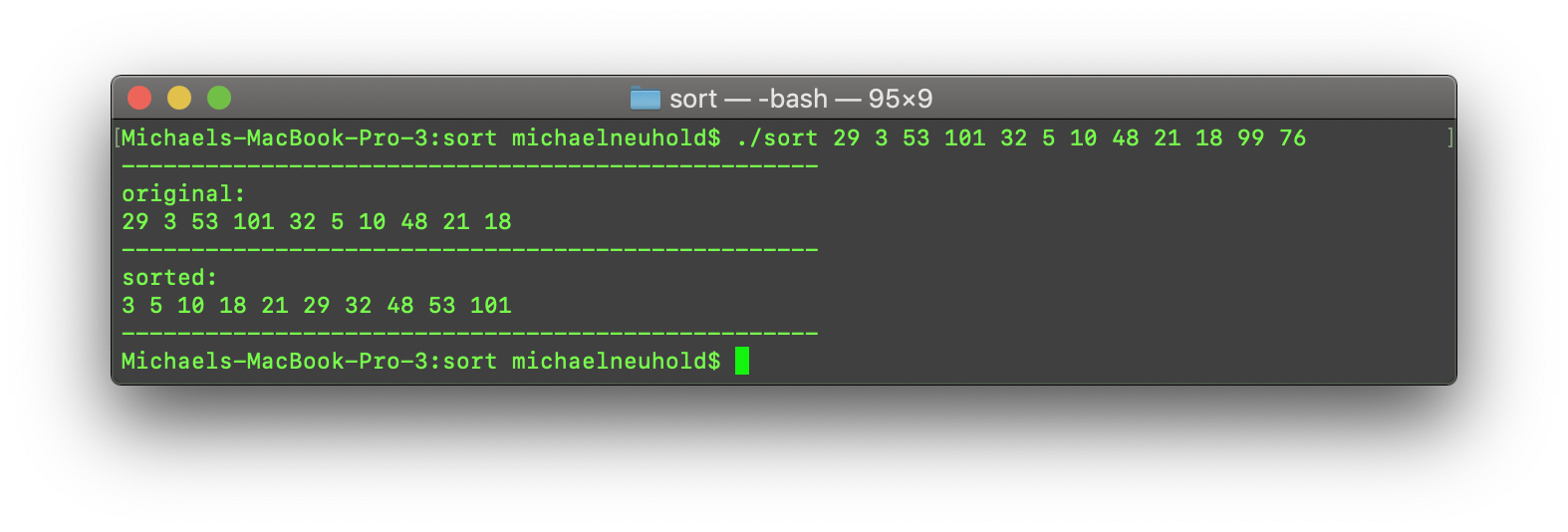
### Eingabe weniger als 10 Werte

Dieser Testfall zeigt, die Übergabe von (n < 10) Parametern. Somit werden alle eingegeben Werte sortiert.



### Eingabe mehr als 10 Werte

Dieser Testfall zeigt, die Übergabe von (n > 10) Parametern. Somit werden lediglich die ersten 10 Werte sortiert.



### Eingabe keiner Werte

Dieser Testfall zeigt einen Programmaufruf ohne Parameter. In diesem Fall wird eine Fehlermeldung ausgegeben.

