SWO3	Übung zu Softwareentwicklung mit klassischen Sprachen und Bibliotheken 3	WS 2018/19, Angabe 2

☐ Gruppe 1 (J. Heinzelreiter)

☐ Gruppe 2 (M. Hava) Name: Niklas Vest Aufwand [h]: 7

☐ Gruppe 3 (P. Kulczycki) Übungsleiter/Tutor: _____ Punkte: _____

Beispiel	Lösungsidee (max. 100%)	Implement. (max. 100%)	Testen (max. 100%)
1 (30 P)	100%	100%	100%
2 (5+10+20 P)	50%	100%	80%
3 (35 P)	50%	100%	100%

Beispiel 1: Hammingfolge (`src/hamming/`)

Die *Folge der regulären Zahlen* $\langle H_1, H_2, H_3, \dots \rangle$, in der Informatik *Hammingfolge* genannt (OEIS-Nummer A051037), ist wie folgt definiert:

1. Es gilt $H_1 = 1$.
2. Sei $H_i, i \in \mathbb{N}$ eine Zahl der Folge. Dann sind auch $2 \cdot H_i$, $3 \cdot H_i$ und $5 \cdot H_i$ Zahlen der Folge.

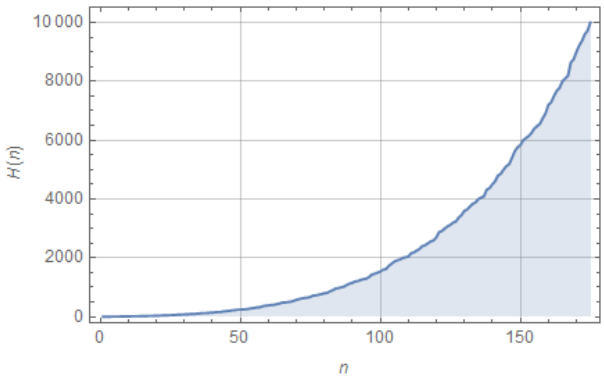
Gesucht ist nun ein möglichst kurzes, einfaches und schnelles C-Programm `hamming_sequence`, welches als Kommandozeilenparameter einen Wert Z nimmt und die ersten n Zahlen der Hammingfolge mit $H_n \leq Z$ aufsteigend sortiert und ohne mehrfaches Vorkommen gleicher Zahlen ausgibt.

Ein Beispiel: Der Aufruf von `hamming_sequence` mit $Z = 30$ liefert die ersten $n = 18$ Zahlen der Hamming-folge:

```
1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30
```

Geben Sie auch die Laufzeit (in Millisekunden) Ihres Algorithmus für verschiedene Werte für Z an. Verwenden Sie dafür die Funktion `clock` aus der Headerdatei `time.h`.

Hinweis: Die Zahlen der Hammingfolge wachsen exponentiell: $H_n \in \mathcal{O}(b^n), b > 1$. Es wäre also keine gute Idee, mit einem Feld der Größe H_n zu arbeiten.



Beispiel 2: *i*-t größtes Element (`src/gross/`)

Es ist einfach, das größte (oder kleinste) Element in einem unsortierten Feld (z. B. ganzer Zahlen) mit einem Durchlauf, also in $\mathcal{O}(n)$, zu ermitteln. Auch das zweit- (oder dritt-)größte Element kann noch in linearer Zeit einfach ermittelt werden.

Hinweis: Sie dürfen im Folgenden davon ausgehen, dass die zu durchsuchenden Felder keine mehrfach vorkommenden Zahlen enthalten.

(a) Implementieren Sie eine C-Funktion

```
int second_largest (int a [], int n);
```

die das zweitgrößte Element in einem unsortierten Feld a ganzer Zahlen mit n Elementen in einem Durchlauf ermittelt.

(b) Ist man allerdings an dem *i*-t größten Element interessiert, ist es am einfachsten, das Feld absteigend zu sortieren und dann das *i*-te Element herauszugreifen. Implementieren Sie eine C-Funktion

```
int ith_largest_1 (int a [], int n, int i);
```

nach diesem Konzept, wobei Sie zum Sortieren Ihre Funktion `merge_sort` aus Beispiel 3 verwenden müssen.

(c) Der Algorithmus `ith_largest_1` hat eine asymptotische Laufzeitkomplexität von $\mathcal{O}(n \cdot \log n)$. Es geht aber auch in linearer Zeit. Erinnern Sie sich zurück an Quick-Sort, der das zu sortierende Feld nach einem Pivotelement in zwei Teilfelder zerlegt (*divide*), beide Teilfelder wieder mittels Quick-Sort sortiert (*conquer*) und damit das gesamte Feld (sogar ganz ohne *combine*) sortiert hat. Implementieren Sie nach diesem Muster eine Funktion

```
int ith_largest_2 (int a [], int n, int i);
```

die zwar mittels Pivotelement eine Zerlegung des Feldes durchführt, dann aber nur jenes Teilfeld weiter betrachtet, in dem das gesuchte Element liegt.

Beispiel 3: Sortieren ganzer Zahlen (`src/sort/`)

Bauen Sie den folgenden Quelltext zu einem voll funktionsfähigen C-Programm aus:

```
#define MAX 100

void merge_sort (int a [], int n) {
// code to sort a[0] .. a[n - 1] using merge sort
}

int main (int argc, char * argv []) {
   int n       = 0;
   int a [MAX] = {0};

// code to read a maximum of MAX values from argv to a and
// to set n to the actual number of values in a

// code to display the unsorted array a

   merge_sort (a, n);

// code to display the sorted array a

   return EXIT_SUCCESS;
}
```