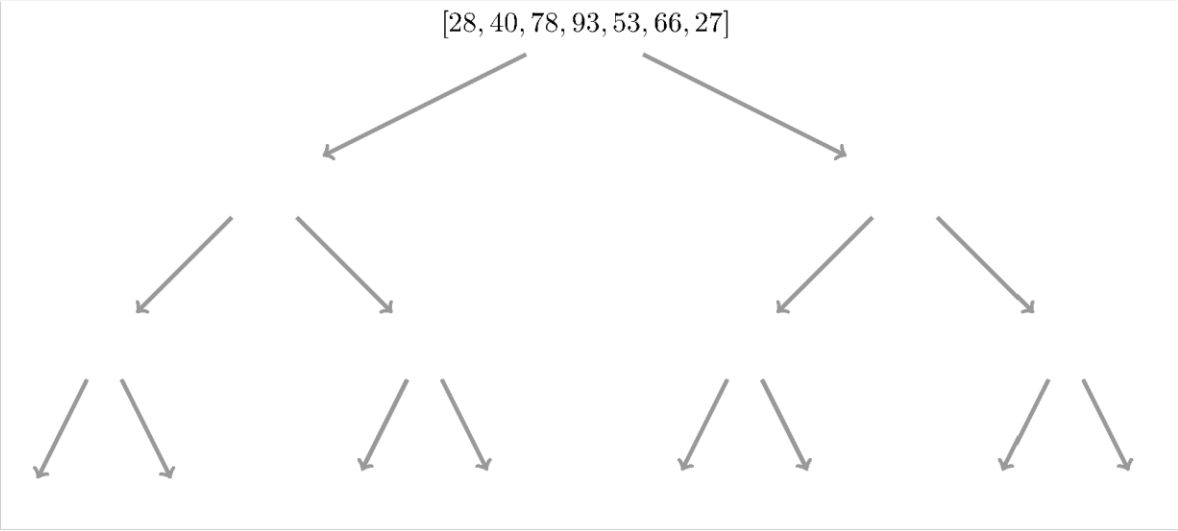
Abschlussprüfung IntroProg WS22/23

1. *In der Prüfung wurden mehrere Möglichkeiten für die einzelnen Lücken gegeben.*

In einem nichtleeren Array aus natürlichen Zahlen wollen wir den Index einer Zahl finden, welche größer ist als alle ihre Nachbarn im Array ist. Beispielsweise wäre im Array [10, 50, 20, 30] die Zahlen 50 und 30 mögliche Kandidaten. Dazu können wir den folgenden divide-and-conquer Algorithmus verwenden. _____ tritt bei jedem Array _____ ein, da die Lösung in konstanter Zeit ermittelt werden kann. Darum können wir für _____ Arrays das Problem _____. Dafür betrachten wir das mittlere Element des Arrays und dessen Nachbarn. Sind alle Nachbarelemente kleiner, haben wir eine Lösung gefunden. Andernfalls können wir das Problem _____ für ein Teilarray von einem größeren Nachbarelemente bis zum Ende des Arrays (so, dass das ursprünglich betrachtete Element nicht in diesem Teilarray liegt) lösen. Diese Lösung ist dann auch für das ursprüngliche Array korrekt, darum also geschieht _____ in konstanter Zeit.

Die Anzahl der Teilprobleme, in die wir in jedem Schritt zerlegen, ist also _____, jedes von jeweils _____ Größe. Insgesamt liegt die Laufzeit damit in _____.

2. **Vervollständigen Sie den Rekursionsbaum beim Mergesort-Aufruf auf dem Array in der Wurzel. Für Arrays mit ungerade vielen Elementen soll das rechte Teilarray ein Element mehr als das linke Teilarray enthalten. Jeder Knoten im Baum entspricht einem rekursiven Aufruf, und soll mit dessen Eingabearray gelabelt werden. Der rekursive Aufruf mit allen Elementen bis zur Mitte ist das linke Kind eines Knoten, der rekursive Aufruf mit den Elementen ab der Mitte ist das rechte Kind eines Knoten. Beispielsweise würde ein Knoten mit dem Label [1, 2, 3, 4] als linkes Kind [1, 2], und als rechtes Kind [3, 4] haben. Falls im Diagramm Kindknoten vorgesehen sind, aber der Knoten keine rekursiven Aufrufe erfordert, sollen – (Bindestrich) Label für die Kinder verwendet werden. Insbesondere sollen also alle BOXEN ausgefüllt werden.**



3. **Möglichkeit Aufgabe 2 als Foto hochzuladen falls Drag&Drop nicht funktioniert.**

4. **CleverSearch ist ein Suchalgorithmus der versucht geschickt zu raten, wo sich ein gesuchtes Element wahrscheinlich befindet:**

```
CleverSearch(A,b,p,r) // A Feld, b Element, p Feldanfang, r Feldende
    if (A[p] != A[r]) and (b >= A[p]) and (b <= A[r])
        q := p + abrunden((b - A[p]) * (r - p) / (A[r] - A[p]))
        if b < A[q]
            return CleverSearch(A, b, p, q)
        else if (b > A[q])
            return CleverSearch(A, b, q + 1, r)
        else
            return q
    else
        if b == A[p]
            return p
        else
            return -1
```

Arrays im Pseudocode sind 1-indiziert.

Tragen Sie in den folgenden Tabellen die Werte von p, q und r beim Auswerten von CleverSearch(A, 53, 1, 6) ein, für die folgenden drei Arrays A. Jede Zeile entspricht einem Funktionsaufruf. Gibt es mehr Zeilen als rekursive Aufrufe, können überzählige Zeilen leer bleiben, oder mit Nullen ausgefüllt werden. In Funktionsaufrufen in denen q nicht berechnet wird, müssen Sie q in der Tabelle auf 0 setzten.

A := [45, 49, 53, 57, 61, 65]

p	q	r
1		6

A := [45, 49, 57, 61, 65, 69]

p	q	r
1		6

A := [0, 53, 54, 55, 56, 57]

p	q	r
1		6

5. **Im Vergleich zu BinarySearch ist die asymptotische Best-Case Laufzeit von CleverSearch…**

○ geringer ○ höher ○ gleich

6. **Im Vergleich zu BinarySearch ist die asymptotische Worst-Case Laufzeit von CleverSearch…**

○ geringer ○ höher ○ gleich

Information:

Sie finden folgende C-Code Fragmente:

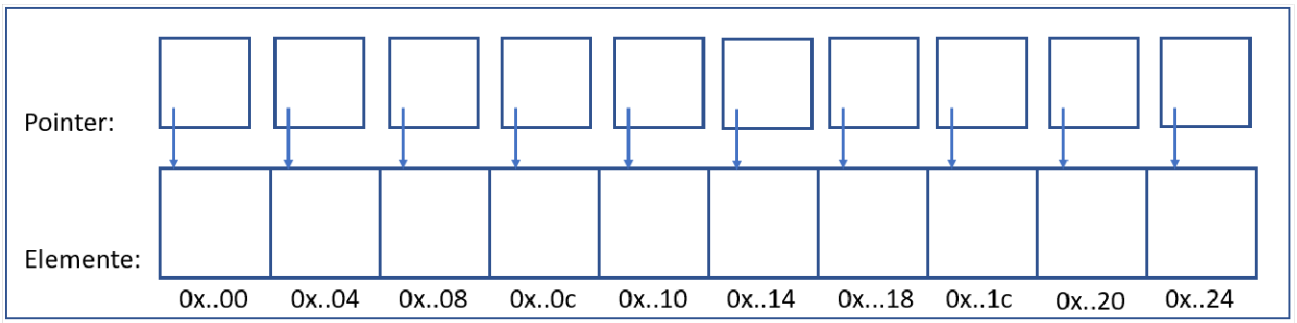
```c
typedef struct {
    int* data;
    int int1;
    int int2;
} mystruct;

void foo(mystruct *var1, int value)
{
    if(var1->int1 >= var1->int2){ exit(1); }
    var1->data[var1->int1] = value;
    (var1->int1)++;
    return;
}

int bar(mystruct *var1)
{
    if (var1->int1 < 1) { exit(1); }
    int value = var1->data[var1->int1 - 1];
    var1->data[var->int1 - 1] = 0;
    (var1->int1)--;
    return value;
}

/*
 * Erstelle die Datenstruktur und reserviere den nötigen Speicher
 * für capacity Elemente und setze die Hilfsvariablen int1 und int2
 */
mystruct* data_create(int capacity) {
    ...
}
```

Analysieren Sie den Code und beantworten Sie folgende Fragen. Gehen Sie davon aus, dass jeglicher Speicher erfolgreich allokiert werden konnte und keine Zugriffe auf ungültige Speicherbereiche stattfinden.

7. **folgender Programmcode ist gegeben.**

```c
int main(int argc, char** args)
{
    mystruct* data = data_create(8);
    foo(data, 9);
    foo(data, 52);
    foo(data, 32);
    foo(data, 10);
    bar();
    foo(data, 10);
    foo(data, 9);
    /* Zustand ? */
    printf("Ich bin fertig.\n");
}
```

Geben Sie den Zustand der Datenstruktur an, nachdem Zeile 10 ausgeführt wurde, aber das Programm noch ausgeführt wird. Achten Sie darauf, wie die Speicheradresse wachsen. Fügen Sie die Zahlen für den Wert der in den Elementen der Datenstruktur (untere Reihe) gespeichert ist per Drag&Drop ein. Bennen Sie die Pointer oberhalb der Elemente. Falls Elemente nicht belegt oder Pointer nicht beschrieben sind, nutzen Sie die "-" Elemente.



8. **Möglichkeit Aufgabe 7 als Foto hochzuladen falls Drag&Drop nicht funktioniert.**

9. **Der oben gezeigte C-Code implementiert folgende abstrakte Datenstruktur:_____**

10. **Der oben gezeigte C-Code benutzt folgende Implementierung der Datenstruktur:_____**

11. **Programmier-aufgabe Teil 1: Einfache Verschlüsselung**

12. **Programmier-aufgabe Teil 2: Bessere Verschlüsserung**

Information:

Gegeben ist folgender Pseudocode: dieser ist 1:1 von den Vorlesungsfolien zu Heaps entnommen und ist fehlerfrei. Sie benötigen diesen zur Beantwortung der Fragen unten und können diesen vorerst überspringen.

```
1  Heapify(A, i)
2      l ← left(i)
3      r ← right(i)
4      if l ≤ heap-size(A) and A[l] > A[i] then largest ← l
5      else largest ← i
6      if r ≤ heap-size(A) and A[r] > A[largest] then largest ← r
7      if largest ≠ i then
8           A[i] ↔ A[largest] // swap(A[i], A[largest]
9           Heapify(A, largest)
10
11 Build-Heap(A)
12     heap-size(A) ← length(A)
13     for i ← ⌊length(A)/2⌋ downto 1 do // 1 inklusiv
14          Heapify(A, i)
15
16 Heap-Extract-Max(A)
17     if heap-size(A) < 1 then error "Kein Element vorhanden!"
18     max ← A[1]
19     A[1] ← A[heap-size(A)]
20     heap-size(A) ← heap-size(A)-1
21     Heapify(A, 1)
22     return max
23
24 Heap-Insert(A,key)
25     heap-size(A) ← heap-size(A)+1
26     i ← heap-size(A)
27     while i>1 and A[Parent(i)] < key do
28          A[i] ← A[Parent(i)]
29          i ← Parent(i)
30     A[i] ← key
31
32 Heapsort(A)
33     Build-Heap(A)
34     for i ← length(A) downto 2 do
35         A[1] ← A[i]
36         heap-size(A) ← heap-size(A)-1
37         Heapify(A, 1)
38
39 Left(i)
40     return 2*i
41
42 Right(i)
43     return (2*i)+1
44
45 Parent(i)
46     return ⌊i/2⌋
```

Analysieren Sie folgende Fragmente einer Implementierung des hier vorgestellten Algorithmus und geben Sie an, in welcher Zeilennummer der Implementierung sich ein Fehler eingeschlichen hat. Oberhalb der Implementierung finden Sie eine Angabe, welcher Teil des Pseudocodes programmiert wurde. Geben Sie dabei an in welcher Zeile der

Implementierung der Fehler ist. Geben Sie die Korrektur des Fehlers an in dem Sie die komplette korrigierte Zeile ohne Tabulatoren eingeben. Bennen Sie die Art des Fehlers.

Sie haben folgende header-Datei definiert:

```
1  typedef struct heap heap;
2
3  struct heap {
4      int* elements;
5      size_t size;
6      size_t capacity;
7  };
8
9  heap* heap_create(size_t capacity);
10 void build_heap(heap* h);
11 void heapify(heap* h, size_t i);
12 int heap_extract_max(heap* h);
13 int heap_insert(heap* h, int key);
14 void heap_free(heap* h);
15 void swap(heap* h, int i, int j);
16 size_t left(size_t i);
17 size_t right(size_t i);
18 size_t parent(size_t i);
```

Gehen Sie dabei davon aus, dass alle anderen Implementierungsteile entsprechend der header-Datei korrekt implementiert wurden, d.h., die hier angegebenen Implementierungsschnipsel sind voneinander unabhängig.

Beispiellösung:

```
1  int main(int argc, char** args)
2  {
3      return0;
4  }
```

Zeile	Korrektur	Fehlertyp
3	return 0;	Syntaxfehler

13. Pseudo-Code Zeile 42-43

```
1  size_t right(size_t i) {
2      return 2*j+2;
3  }
```

Zeile	Korrektur	Fehlertyp

14. Pseudo-Code Zeile 24-30

```
1  int heap_insert(heap* h, int key) {
2      if (h->size >= h->capacity)
3          return -1;
4
5      h->size = h->size + 1;
6      int i = h->size;
7      while (i>0 && h->elements[parent(i)] < key) {
8          h->elements[i] = h->elements[parent(i)];
9          i = parent(i);
10     }
11     h->elements[i] = key;
12     return 0;
13 }
```

Zeile	Korrektur	Fehlertyp

15. Pseudo-Code Zeile 24-30

```
1  int heap_insert(heap* h; int key) {
2      if (h->size >= h->capacity)
3          return -1;
4
5      h->size = h->size + 1;
6      int i = h->size-1;
7      while (i>0 && h->elements[parent(i)] < key) {
8          h->elements[i] = h->elements[parent(i)];
9          i = parent(i);
10     }
11     h->elements[i] = key;
12     return 0;
13 }
```

Zeile	Korrektur	Fehlertyp

Information:

Ein Mitarbeiter nimmt die Liste der Studierenden, die die Klausur in ISIS geschrieben haben, und sortiert sie nach Nachnamen aufsteigend in alphabetischer Reihenfolge. Danach geht er der Reihe nach durch seine Liste und trägt jeweils pro Studierende/n eine Punktzahl für ihre/seine Abgabe ein. Die Liste wird von ISIS exportiert und die Punktzahlen werden entsprechend des Notenschlüssels fehlerfrei in Noten übersetzt. Der Mitarbeiter will danach die Noten in eine andere Platform, MTS, per Hand übertragen. Er öffnet im MTS die Liste von den für die Klausur angemeldeten Studierenden und er sortiert die Liste der Studierenden nach Nachnamen aufsteigend in alphabetischer Reihenfolge. Danach geht er der Reihe nach durch seine Liste im MTS und liest jedesmal den Nachnamen in der MTS-Liste, springt zur nächsten Person im ISIS-Export mit diesem Nachnamen und schreibt die Note entsprechend der Punktzahl. Der Mitarbeiter benutzt beide Male zur Sortierung eine Variante von QuickSort, die den Pivot zufällig auswählt. Zur Vereinfachung wird davon ausgegangen, dass alle angemeldeten Studierenden die Klausur geschrieben haben und dass keine Studierenden die Klausur geschrieben haben, ohne angemeldet zu sein. Die Eintragung wäre korrekt, wenn alle Studierenden die Note bekommen würden, die ihnen in ISIS zugewiesen wurde.

16. **Ist dieser Ablauf korrekt? Wenn nicht, warum ist er inkorrekt und was ist die Eigenschaft der verwendeten Sortierverfahren, weswegen dieser Ablauf inkorrekt ist?**

    _____
    _____
    _____

17. **Für welche Daten können Fehler erfolgen? D.h., unter welcher Bedingung bzgl. Eigenschaften der Daten können Fehler erfolgen?**

    _____
    _____
    _____

18. Wie kann man die Auswahl des Pivots so ändern, dass der Ablauf korrekt wird?

_____
_____
_____

Information:

Eine Variante von CountSort in Pseudocode folgt. Der Code dieser Implementierung unterscheidet sich von der in der Vorlesung vorgestellten Version, zeigt jedoch das gleiche Verhalten.

CountSort(Array A)

```
1  // C: 0-gefülltes Array der Länge m (m groß genug)
2  // n: Länge von A
3  for i ← 1 to n do // n inklusiv
4      C[A[i]]++
5  k ← 1
6  for v ← 1 to m do // m inklusiv
7      while C[v] ≠ 0
8          A[k] ← v
9          k++
10         C[v] ← C[v]-1
```

Die Korrektheit dieser Varianten von CountSort ist mit vollständiger Induktion zu beweisen.

19. $\mathbb{N}$ bezeichnet die Menge an streng positiven Ganzzahlen. Es wird angenommen, dass alle Werte in A in $\mathbb{N}$ enthalten sind. \| bezeichnet der Kardinalitätsoperator, d.h., wieviele Elemente es in der Menge gibt, bspw. \|{1, 5, 2}\| = 3. j und v sind freie Laufindexe im Array A bzw. Array C. => bezeichnet eine Induktion, bspw. (v > 1) => (v > 0). i ist der Laufindex in A bei der ersten For-Schleife.

Welche Invariante gilt bei der ersten For-Schleife (wo das Array C befüllt wird)? Unterscheide zwischen den einzelnen Antwortmöglichkeiten sind fett markiert.

- $(1 \leq v \leq m) \Rightarrow (C[v] = |\{j \in \mathbb{N}, \mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n}. A[j] = \boldsymbol{v}\}|)$
- $(1 \leq v \leq m) \Rightarrow (C[v] = |j \in \mathbb{N}, \mathbf{1} \leq \boldsymbol{j} < \boldsymbol{i}: A[j] = \boldsymbol{v} + \mathbf{1}\}|)$
- $(\mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{i} - \mathbf{1}) \Rightarrow (\boldsymbol{A[j]} \leq \boldsymbol{A[j+1]})$
- $(1 \leq v \leq m) \Rightarrow (C[v] = |\{j \in \mathbb{N}, \mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{i}: A[j] = \boldsymbol{v}\}|)$
- $(1 \leq v \leq m) \Rightarrow (C[v] = |\{j \in \mathbb{N}, \mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{i}: A[j] = \boldsymbol{v}\}|)$

20. Um die Schreibweise zu vereinfachen, wird in den Formeln davon ausgegangen, dass k (laufender Index bei der Befüllung von A) streng größer als 1 ist. v ist der Laufindex in C bei der zweiten For-Schleife.

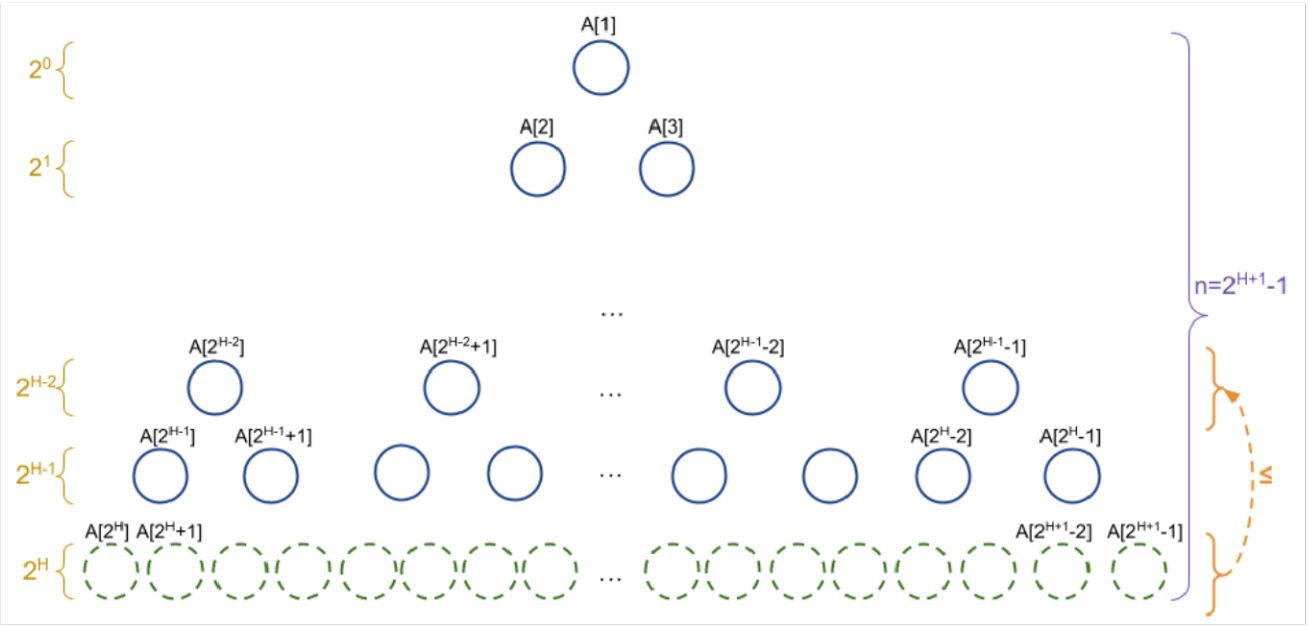
Welche Invariante gilt bei der zweiten For-Schleife (wo das Array A befüllt wird)?

- $A[1] \leq \cdots \leq A[k-1]\ und\ A[k-1] < v$
- $A[1] \leq \cdots \leq A[k+1]\ und\ A[k+1] < v$
- $A[1] \leq \cdots \leq A[k]\ und\ A[k] = v$
- $A[1] \leq \cdots \leq A[k]\ und\ A[k] < v$
- $A[1] \leq \cdots \leq A[k-1]\ und\ A[k-1] = v$

21. Gegeben sei ein Heap von n items. Was ist die engste obere Schranke der Laufzeitkomplexität, um herauszufinden, ob ein gegebenes Element im Heap ist?

- O(log(n))
- $O(n^2)$
- O(n)
- O(n log(n))
- O(1)

22. Gegeben ist ein linksvoller Binärbaum (wie in der Vorlesung beschrieben) der Höhe H mit $n = 2^{H+1}-1$ Knoten (damit sind alle Ebenen gefüllt). Mit Ausnahme der Blattknoten hat jeder Knoten einen Wert kleiner(-gleich) dem Wert des Elternknotens, d.h., der Teilbaum von allen Ebenen bis zur vorletzten (inklusive) ist ein Max-Heap. Der Wert jedes Blattknoten ist lediglich kleiner(-gleich) dem Wert des jeweiligen "Großelternknoten" (Elternknoten vom Elternknoten).
Sei A ist die Array-Repräsentation des gegebenen Baumes. A hat also die Länge $n = 2^{H+1}-1$ und das Teilarray $A[1 \dots 2^H-1]$ ist die Array-Repräsentation eines Max-Heap.
Die Beziehung der Blätter zu ihren Großelternknoten wird wie folgt formalisiert:
$\forall i_1, i_2 \in \mathbb{N}, 2^{H-2} \leq i_i < 2^{H-1}, 4i_1 \leq i_2 < 4i_1 + 4, A[i_2] \leq A[i_1]$.
$i_1$bezeichnet das Index von einem beliebigen Knoten v in der vorvorletzten Ebene. Für einen Knoten v in der vorvorletzten Ebene bezeichnet i_2 das Index von einem Knoten in der letzten Ebene, welches ein Enkelkind von v ist. $\mathbb{N}$ bezeichnet die Menge der streng positiven Ganzzahlen.
Eine schematische Abbildung des Baums folgt.



Es wird jetzt Build_Heap() auf A aufgerufen. Was ist die maximale Anzahl an Swap-Operationen (Tausch von zwei Knoten), die im Array A durchgeführt werden?

- $2^{H-2} - 1$
- 0
- 2^{H-2}
- n
- 2^{H-1}
- 2^H
- H
- $2^{H-1} - 1$
- $2^H - 1$
- $2^{H-1} - 1$