Auswahlfragen

1 Aufgabe - code vorgabe

```
1 typedef struct _Node
2 {
3     int value;
4     Node* left;
5     Node* right;
6 } Node;
7
8 void print_tree_rek(Node* node){
9     printf("Besuche Node mit Wert %d\n", node->value);
10
11     if (node != NULL){
12         print_tree_rek(node->left);
13         printf("%d ", node->value);
14         print_tree_rek(node->right);
15     }
16 }
```

Man erhält Segmentation fault. wieso?

a) node hat kein attribute value
b) eine notwendige bibliothek wurde nicht eingebunden (erzeugt keinen seg. fault)
c) Zeile 13 dereferenziert NULL-Pointer
d) übergebener Parameter ist NULL und Zeile 9 dereferenziert NULL-Pointer

2 Aufgabe

```
1 void build_strukt(Strukt *aStrukt){
2     aStrukt->value = 0;
3     // ...
4 }
5
6 void free_strukt(Strukt *aStrukt){
7     // ...
8     free(aStrukt);
9 }
10
11 int main(int argc, char const *argv[])
12 {
13     Strukt aStrukt;
14
15     build_strukt(&aStrukt);
16
17     // ...
18
19     free_strukt(&aStrukt);
20
21     return 0;
22 }
```

Bei ausführung gibt es Fehler:

NULL(.....)malloc: *** error for object 0x....., pointer being freed was not allocated
und nochmal NULL malloc: *** set a breakpoint in malloc_error_break to debug, Job i, terminated by signal SIGABRT (abort)

was ist das Problem?

a) aStruct in main statisch allokiert aber free_struct dynamisch freigegeben
b) aStruct statt in main() in free_struct() freigegeben
c) build_struct() und free_struct() bekommen struct-strukturen statt Pointer zu struct-Strukturen
d) aStruct in main() statisch allokiert, danach build_struct()dynamisch allokiert

3 Aufgabe

```
char* str ="Hallo";
char* str2="Welt";
char str3 [12];
strcpy(str3, str1);
strcat(str3, str2);
```

(welt! nicht nur welt)
nach Aufruf strlen() welchen wert, wenn dabei Zeiger auf str2-Array übergeben wurde..?

a) 5
b) 12
c) 4
d) 10 (strlen ohne nullbyte, zusammengezählt)

4 Aufgabe

es sind nums.c nums.h und main.c gegeben, wie muss compiler aufgerufen werden um alles zu kompilieren und ausführbare Datei zu erzeugen?

a) clang -Wall -std=cli main.c nums.c nums.h
b) clang -Wall -std=cli main.c nums.h
c) clang -Wall -std=cli main.c -L nums.h

d) clang -Wall -std=cli main.c nums.c

5 Aufgabe

Gegeben sei eine einfach zyklisch verkettete Liste: Jedes Element zeigt auf das nächste Element, und das letzte Element zeigt auf das erste Element. Man verfügt nur über einen Pointer zum ersten Element. Was ist die kleinste obere Schranke für den asymptotischen Aufwand beim Einfügen eines Elements am Anfang der Liste?

a) O(n^2)
b) O(n)
c) O(1)
d) O(nlogn)

6 Aufgabe

Es werden nur unsortierte Arrays betrachtet, in denen jedes Element maximal zehn Stellen entfernt von der korrekten Position nach einer Sortierung liegt.
Was ist dann die kleinste obere Schranke der Worst-Case-Komplexität von Insertionsort?

a) O(n^2)
b) O(n)
c) O(n^2)
d) O(nlogn)

7 Aufgabe

Unter welcher Bedingung ist ein Sortieralgorithmus stabil?

a) Wenn eine beliebige aber endlich große Eingabe übergeben wird, hat nach der Sortierung jedes Element einen Wert kleiner als den Wert des dárauffolgenden Elements.
b) Wenn eine beliebige aber endlich große Eingabe übergeben wird, wird die Reihenfolge von gleichwertigen Elementen nach der Sortierung beibehalten.
c) Wenn eine beliebige aber endlich große Eingabe übergeben wird, terminiert der Sortieralgorithmus stets nach endlicher Zeit.

gleichwertige elemente =gleiche schlüssel usw

8 Aufgabe

Bei einer Variante von Quicksort wird das letzte Element als Pivotelement gewählt, und alle Elemente mit dem gleichen Wert wie das Pivotelement werden in die linke Liste
eingefügt. Ist diese Variante von Quicksort stabil?

a) Es ist nicht klar
b) Nein
c) Ja

9 Aufgabe

Bei einer Variante von Quicksort wird das Pivotelement zufällig gewählt. Alle Elemente mit dem gleichen Wert wie das Pivotelement, die davor vorkommen, werden in die rechte
Liste eingefügt. Alle Elemente mit dem gleichen Wert wie das Pivotelement, die danach vorkommen, werden in die linke Liste eingefügt.
ist diese Variante von Quicksort stabil?

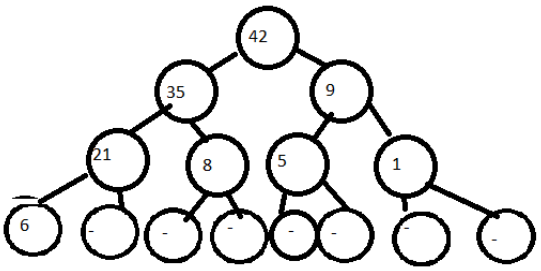
a) ja
b) unklar
c) nein

Datenstrukturen

Drag and drop für versch. Heaps/Bäume

10 Aufgabe

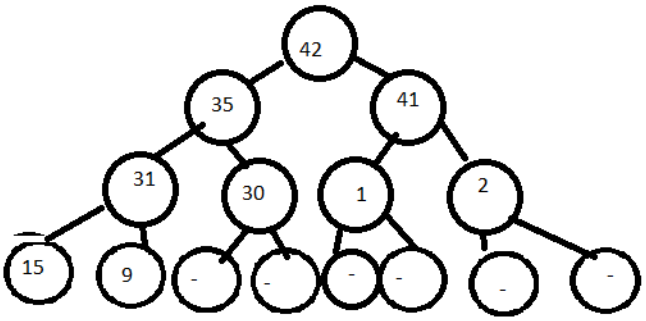
Gegeben ist ein Max-Heap in der Array-Notation: (42, 21, 9, 6, 8, 5, 1). Fügen Sie in diesen Heap mit der heapinsert-Funktion die Zahl 35 ein.
Geben Sie das Ergebnis in der unten stehenden Baumstruktur an, Besetzen Sie nicht benötigte Knoten mit einem Strich (-)



(11 und 13 platzhalter für alt. upload)

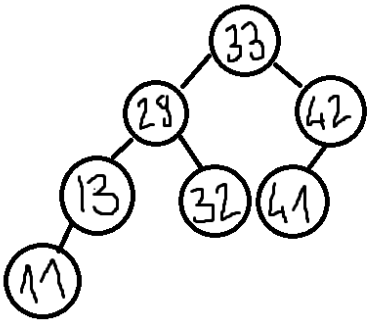
12 Aufgabe

Array [30,31,2,15,35,1,41,42,9] build_heap für Max-Heap ausführen:



14 Aufgabe

Ist der gegebene Binärbaum ein AVl Baum? Begründe!



Ja!
Höhe des linken & rechten Teilbaums unterscheidet sich max. um 1
Alle Elemente links kleiner als die des Knotens, alle rechts größer
Erfüllt Bedingungen für höhenbalancierten Suchbaum und somit AVL-Baum!

15 Aufgabe

Die Telefon-Hotline eines Gesundheitsamts ist während der Corona-Pandemie chronisch unterbesetzt. Es wird daher ein System in Auftrag gegeben, welches dafür sorgen soll, dass freiwerdenden Angestellten jeweils die zu diesem Zeitpunkt am längsten wartende Person vermittelt wird.
Frage: in welcher Datenstruktur werden die (Pointer zu den) entsprechenden Anrufvorgänge(n) am besten abgelegt? Nennen Sie die für die Anwendung notwendigen Funktionen auf der Datenstruktur.
Datenstruktur: FIFO queue
Funktionen: tail, head pointer setzen (queue start und ende, eventuell length nutzen um durch zu sagen wie lange die wartezeit ist)
(habe es nicht gut genug erläutert weil ich dachte queue versteht sich von selbst also Ergänzung:)
Tail und head pointer sind beinhaltet in der queue um Start und Ende anzugeben. Zudem kann durch die position in der queue der Platz gesagt werden. FIFO bedeutet, erste Anrufer werden auch zuerst aufgerufen. Push, Pop, empty als standard funktionen. (unklar wie “Funktionen” Bezug haben)

16 Aufgabe

An einer Universität werden alle Leistungen der Studierenden in einem Universitäts-Informations-System gespeichert. Nach der Eingabe der Matrikelnummer können Angestellte des Prüfungsamtes die zugehörigen Noten eintragen oder Zeugnisse erstellen (vielleicht auch mal rechtzeitig). Wegen einer Sicherheitslücke muss angenommen werden, dass die Daten kürzlich von Dritten manipuliert wurden. Es existiert ein rechtsgültiger Ausdruck des Datensatzes als nach Matrikelnummer aufsteigend sortierte Liste, Da unbedingt Zeugnisse ausgestellt werden müssen, gehen die Angestellten die Liste von oben nach unten durch und tragen die Datensätze in dieser Reihenfolge in das frisch aufgesetzte System ein, Nachdem der Betrieb wieder aufgenommen wurde, stellen die Angestellten jedoch fest, dass der Zugriff auf fast alle Datensätze nun deutlich länger dauert als zuvor. Datensätze die als letztes in das neue System eingespielt wurden, benötigen jetzt beim Zugriff die meiste Zeit. Dabei wurden auch schon vor der Sicherheitslücke neue Datensätze hinzugefügt, ohne dass solch ein Problem jemals auftrat,
Fragen: in welcher Datenstruktur werden die Matrikelnummern (mit dem Pointer zu den Noten) vermutlich abgelegt? Mit welcher besser geeigneten Datenstruktur wäre das Problem nicht aufgetreten? Erläutern Sie das auftretende Problem und begründen Sie Ihre Wahl der besser geeigneten Datenstruktur.

Bisherige Datenstruktur: einfach verknüpfte Liste
Besser geeignete Datenstruktur: doppelt verknüpfte liste (pointer in beide richtungen)
Erläuterung/Begründung: erleichtert zugriff und ist nicht extrem aufwändig als Änderung einzuführen, schneller Zugriff auf alle elemente, nicht gesamte Liste durchgehen.
wenn man zB: noch middle pointer setzt, noch schneller auslesen möglich

Diese Antworten waren im nachhinein betrachtet nicht ganz sinnig. Abgesehen von dem schlechten Praxis-Beispiel, das sehr unwahrscheinlich in C sondern eher über Instanzen in Java geregelt werden würde, sollte es sich wahrscheinlich auch um einen nicht ausgeglichenen Suchbaum handeln. (SUchbaum degenerierte zu einfach verknüpfter liste) Die Lösung wäre dementsprechend das wieder gleichmäßige Wichten um einen korrekten AVL-Suchbaum zu schaffen, AVL rebalanciert.
Ich persönlich hing bei der Formulierung zu sehr an “Liste” und mir war unklar, dass es vor dem Angriff dieselbe (?) Struktur zu sein schien... Also genau lesen!

17 Aufgabe

Eine wirklich schlecht gewichtete aufgabe mit zu wenig Punkten, für zu viel Aufwand. (Ein Punkt für 8x4 Einträge.) **Eingabe a= [12,24,11,42] hand simulieren wobei Array a, variable x in zeile 4 angegeben**

```
algoi(Array a);
    x = 1;
    while x <= length(a) do
    //Hier
    if ( x== 1 or a[x] <= a [x-1]) then
        x =x+1;
    else
        swap(a[x], a[x-1])
        x=x-1;
```

	a[1]	a[2]	a[3]	a[4]	x
1	12	24	11	42	1
2	12	24	11	42	2
3	12	24	11	42	3
4	12	11	24	42	2
5					

```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void num_out(int *nums, int len) {
    for (int i = 1; i < len; i++)
        printf("%d, ", nums[i]);
    printf("%d\n", nums[len]);
}

int main() {
    int a[] = {-1, 12, 24, 11, 42, 1};
    int a_len = 5;
    int x = 1;
    while (x <= a_len) {
        num_out(a, a_len);
        printf("%d\n", x);
        if (x == 1 || a[x] >= a[x-1]) {
            x += 1;
        } else {
            swap(&a[x], &a[x-1]);
            x -= 1;
        }
    }
}
```

Korrekter Code für Aufgabe 17 ergibt:
12, 24, 11, 42, 1
12, 24, 11, 42, 2
12, 24, 11, 42, 3
12, 11, 24, 42, 2
11, 12, 24, 42, 1
11, 12, 24, 42, 2
11, 12, 24, 42, 3
11, 12, 24, 42, 4

18 Welcher Algorithmus ist das?

Ein Sortieralgorithmus -> Insertionsort aber suboptimal

19 Cases n=length(a)

Worst case: O(n^2)
best case: O(n^2)

20 Aufgabe

Der gegebene Algorithmus ist in seiner Lautzeitkasse ineffizient, Beschreiben Sie mit wenigen Worten eine Optimierung des Algorithmus, welche die Effaienz steigert, aber die Laufzeitklasse nicht ändert, Mit welchem aus der Vorlesung bekannten Algorithmus ist dieser dann vergleichbar?

Beschreibung der optimierung:
- Der Index (x bei Aufgabe 17) muss nach der erfolgreichen Insertion nicht vom Einfügungs-Ort langsam zurück iterieren, sondern wird direkt auf den Index des nächsten uneingefügten Elements gesetzt.
- Statt die Elemente einzeln zu swap()en, was das einzufügende Element mehrmals bewegen würde, werden bei deinem Code zuerst alle anderen Elemente eins nach hinten bewegt, und dann das einzufügende Element nur einmal bewegt.

Vergleichbarer Algorithmus: Insertionsort (beide verahlten sich so aber pseudo ist nicht wie in vl)

```c
void insertionSort(int arr[], int n)
/*Vergleiche das aktuelle Element (Schlüssel) mit seinem Vorgänger*/
{
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        /* Move elements of arr[0..i-1], that are
          greater than key, to one position ahead
          of their current position */
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Laufzeit: O(n) best case, O(n^2) worst case

21 Aufgabe

Konkatenation von Arrays z.B. [0,1] o [1,6,1] = [0,1,1,6,1,]
A sei Eingabe Array. A' Ausgabe Array, n Länge Eingabe Array und m Anzahl Indizes Werte Array.

```
1  Countsort1(A)
2      for i = 1 to n do
3          w = A[i]
4          j = w+1
5          counts[j]++
6      for j = 1 to m do
7          w= j-1
8          for k = 1 to counts[j] do
9              A' = A' o [w]
10     return A'
11
```

Countsort(1) als Variante von Countsort:
Was ist die kleinste obere Schranke (Laufzeit)?

a) O(n)
b) O(n^2)
c) O(n+m)
d) O(m)

22 Aufgabe

Die Funktion pos () gibt die Position eines Wertes im Array zurück, z.B. pos(17, A) = 3 für A = (5, 2, 17, 20) (es wird angenommen, dass alle Werte unterschiedlich sind). **Welche Aussage trifft nach der Terminierung von CountSort1() für beliebige i, i' aus dem Intervall (1, n) zu?**

a) (A[i']) < A[i]) ⇔ (pos(A[i], A') < pos(A[i'], A'))
b) (A[i] < A[i']) ⇔ (pos(A[i], A')> pos(A[i'], A'))
c) (A[i] > A[i']) ->(pos(A[i], A') = pos(A[i'], A')) - nur einfache implikation
d) (A[i] < A[i']) ⇔ (pos(A[i], A') < pos(A[i'], A'))

d da ai kleiner ai', also aufwärts sortiert - einzige äquivalenz die aussagt dass array A' sortiert ist x<- A[i], y<- A[i'] also: x,y element A: (x<y) ⇔ (pos(x,A') <pos(y,A'))

23 Aufgabe

```
1  Countsort2(A)
2      for i = 1 to n do
3          w = A[i]
4          w' = f(w)
5          j = w'+1
6          werte[j] = werte[j] o [w]
7      for j = 1 to m do
8          werte[j] = MergeSort(werte[j])
9              A' = A' o werte[j]
10     return A'
11
```

Es Seien n1... nm die Längen der Arrays an Stellen 1...m im Werte-Array. **Wie lautet die kleinste obere Schranke von CountSort2()?**

a) o(n^2+m)
b) O(n+m)
c) O(n1log(n1)+...+nmlog(nm)+m)
d) O(n1+...+nm+m)

mehrere verkettete listen also c (?)

24 Aufgabe

Ein Sortieralgorithmus ist korrekt, wenn für eine beliebige Eingabe deren Werte in richtiger Sortierung ausgegeben werden. **Ist CountSort2() ein korrekter Sortieralgorithmus?**

a) nein
b) ja
c) ungewiss

25 Aufgabe

Die Funktion pos () gibt wieder die Position eines Wertes im Array zurück, z.B. pos(17, A) = 3 für A = (5, 2,17, 20). S ist ein Array an einer bestimmten Stelle j im Werte-Array (S = werte [j]). Welche Aussage trifft nach der Terminierung von **CountSort2() für beliebige i, i' aus dem Intervall (1, n) unter der Bedingung zu, dass A[i], A[i'] E S.**

a) (A[i] > A*) -> (pos (A[i], MergeSort(S)) = pos(A[i'] , MergeSort(S)))
b) (A[i]] < A(i'l) <-> (pos (A[i], MergeSort(S)) < pos(A[i'] , MergeSort(S)))
c) (f(A[i]) < S(Ali')) <-> (pos (A[i], MergeSort(S)) < pos (A[i'] , MergeSort(S)))
d) (A[i] ≤ A[i']) <-> (pos(A[i], MergeSort(S)) >= pos(A[i']), MergeSort(S)))

c war bait antwort, auf rechter seite steht merge sort, antwort b

26 Aufgabe

Welche weitere Aussage trifft nach Terminierung von CountSort2() für beliebige i, i' aus intervall [1,n] zu?

a) (A'[i']) < A'[i]) ⇔ (pos(A'[i], A') < pos (A'[i']), A'))
b) (A'[i]) < A'[i']) ⇔pos (A'[i], A') < pos(A'[i'], A'))
c) (A'[i'])> A'[i']) -> (pos(A'[i], A') < pos(A'[i'], A'))
d) (A'[i']) <A'[i']) ⇔ (pos (A'[i], A') > pos(A'[i'], A'))

a bait b korrekt, sortiereigenschaft

27 Aufgabe

Pseudocode korrigieren und syntax-/semantikfehler finden und identifizieren.

28 Aufgabe

und folgende; viel zu Umfangreiche Programmieraufgabe.
Vorgaben herunterladen, umgebung öffnen.
String-Bibliothek implementieren.
Radix-Sort implementieren.