TECHNISCHE UNIVERSITÄT BERLIN

Fakultät IV – Elektrotechnik und Informatik
Open Distributed Systems
Prof. Manfred Hauswirth
Foucard / Kuehn / Seifert

Einführung in die Programmierung, WiSe 2019/2020
Klausur vom 1. Juli 2020

Bitte füllen Sie alle folgenden Felder aus:

Klausur-ID:	**001A**
Platznummer:	
ZECM-Login:	
Vorname:	
Nachname:	
Matrikelnummer:	
Studiengang:	
Hochschule:	

Durch meine Unterschrift bestätige ich die Korrektheit obiger Angaben!

Ort, Datum

Unterschrift

Beachten Sie folgende Hinweise!

- Ihre Klausur wird anonymisiert korrigiert, schreiben Sie Ihren Namen daher nur auf das Deckblatt, die Klausuren sind individuell nummeriert.
- Als Hilfsmittel ist nur **ein** handschriftlich doppelseitig beschriebenes DIN-A4 Blatt erlaubt.
- Schreiben Sie **nicht** mit roter Farbe, grüner Farbe oder Bleistift, diese Lösungen werden nicht bewertet!
- Benutzen Sie keinen 'Tintenkiller' oder Tipp-Ex sondern streichen Sie falsche Lösungen durch!
- Lösungen, die wegen undeutlicher Schrift nicht lesbar sind, können nicht bewertet werden.
- Geben Sie nur eine Lösung pro Aufgabe ab, streichen Sie alle alternativen Lösungsansätze auf Schmier-/Notizblättern vor der Abgabe.
- Notieren Sie Ihre Antworten nur auf dem Blatt (oder dessen Rückseite), auf dem auch die zugehörige Aufgabe steht, da die Aufgaben getrennt korrigiert werden!
- Benutzen Sie kein eigenes Schmierpapier! Am Ende der Klausur finden Sie ein Zusatzblatt.
- Zusätzliches Papier erhalten Sie bei der Aufsicht. Notieren Sie darauf Ihre persönliche Klausur-ID **001A**. Notieren Sie auch bei der entsprechenden Aufgabe die Verwendung des zusätzlichen Blattes! **Schreiben Sie jeweils nur die Lösung einer Aufgabe auf ein Zusatzblatt.**
- Sie haben **90 Minuten Bearbeitungszeit**!
- Insgesamt können in dieser Klausur **59 Punkte** erreicht werden.
- Diese Klausur besteht mit diesem Deckblatt aus den (nummerierten) Seiten **1** - **24**.
- Geben Sie alle Blätter des Tests und alle Zusatzblätter ab, auch wenn diese nicht beschrieben sind!
- Beschädigen oder überschreiben Sie nicht den Barcode am Ende der Seiten.

Zusatzblätter Nr.:	

Diese Seite ist absichtlich leer.

Aufgabe 1: Multiple-Choice-Fragen (/10 Punkte)

Bei den folgenden Multiple-Choice-Fragen, ist genau eine der vorgegebenen Aussagen wahr. Bitte kreuzen Sie diese Aussage an. Ein falsch gesetztes Kreuz kann durch Zeichnen eines leeren ◯-Symbols links neben der Aussage aufgehoben werden.

C-Verständnis

(a) (2 Punkte) Wir betrachten das folgende Programm:

```
1  #include<stdio.h>
2
3  int main()
4  {
5     int n, sum = 0, c, remainder;
6
7     scanf("%d", &n);
8
9     while(n != 0)
10    {
11       remainder = n%10;
12       sum += remainder;
13       n = n/10;
14    }
15
16    return 0;
17 }
```

Der User gibt die Zahl 5467 ein. Was ist der Wert von n am Ende des Programms?

◯	0
◯	7
◯	22
◯	840

C-Debugging

(b) (2 Punkte) Wir betrachten das folgende Programm:

```
1  #include <stdio.h>
2
3  int main()
4  {
5     int n, i, *ptr, sum = 0;
6
7     printf("\n\nEnter number of elements: ");
8     scanf("%d", &n);
9
10    // dynamic memory allocation using malloc()
11    ptr = (int *) malloc(sum*sizeof(int));
12
13    printf("\n\nEnter elements of array: \n\n");
14    for(i = 0; i < n; i++)
15    {
16       // storing elements at contiguous memory locations
17       scanf("%d", ptr+i);
18       sum = sum + *(ptr + i);
19    }
20
21    // printing the array elements using pointer to the location
22    printf("\n\nThe elements of the array are: ");
23    for(i = 0; i < n; i++)
24    {
25       printf("%d  ",ptr[i]);    // ptr[i] is same as *(ptr + i)
26    }
27
28    /*
29       freeing memory of ptr allocated by malloc
30       using the free() method
31    */
32    free(ptr);
33
34    return 0;
35 }
```

Ist die Allokierung von `ptr` richtig? Wie soll es lauten?

◯	`ptr = (int *) malloc(100*sizeof(int));`
◯	`ptr = (int *) malloc(sum*sizeof(int));`
◯	`ptr = (int *) malloc(i*sizeof(int));`
◯	`ptr = (int *) malloc(n*sizeof(int));`

Pseudocode → C

(c) (2 Punkt) Wir betrachten das folgende korrekte Programm in Pseudocode:

```
1 // array of length n assumed to exist
2 maximum = 0
3
4 for i = 1 to n
5    if array[i] > maximum
6       maximum = array[i]
```

Hier ist ein Versuch, dieses Programm in C zu übersetzen:

```
1  // n assumed to be declared and assigned to length of array
2  // array values assumed to be positive integers
3  int maximum, i;
4
5  for (i = 1; i <= n; i++)
6  {
7     if (array[i] > maximum) {
8        maximum = array[i];
9     }
10 }
```

Die C-Version beinhaltet Fehler. Wie muss Zeile 2-4 ersetzt werden, um die Fehler zu beheben?
A1:

```
2 int maximum = 0;
3
4 for (i = 0; i < n; i++)
```

A2:

```
2 int maximum;
3
4 for (i = 0; i <= n; i++)
```

A3:

```
2 int maximum = 5;
3
4 for (i = 0; i < n; i++)
```

A4:

```
2 int maximum = 0;
3
4 for (i = 1; i < n; i++);
```

◯	A1
◯	A2
◯	A3
◯	A4

Pointers

Die Funktion `delete_and_shift()` im folgenden Ausschnitt sollte in einem Array den Wert an einer gegebenen Position löschen und alle folgenden Werte nach links verschieben, so dass die Lücke verschwindet:

```
1  int MAX_ARR = 1000;
2
3  void delete_and_shift(int *arr, int len, int ind){
4     for (int i = ind; i < len; i++){
5        arr[i] = arr[i+1];
6     }
7     len = len - 1;
8  }
9
10 int main() {
11    int len = 4;
12    int arr[MAX_ARR];
13    arr[0] = 10;
14    arr[1] = 2;
15    arr[2] = 8;
16    arr[3] = 3;
17
18    delete_and_shift(arr, len, 1);
19
20    for (int i = 0; i < len; i++){
21       printf("arr[%d]=%d\n", i, arr[i]);
22    }
23
24    return 0;
25 }
```

Jedoch ist das angezeigte Ergebnis nicht wie erwartet:

```
arr[0]=10
arr[1]=8
arr[2]=3
arr[3]=3
```

Stattdessen sollte die Ausgabe wie folgt aussehen:

```
arr[0]=10
arr[1]=8
arr[2]=3
```

Der Code soll korrigiert werden.

(d) (2 Punkte) Wie soll man den Aufruf von `delete_and_shift` (Zeile 18) korrigieren?

◯	Man soll `len` durch `&len` ersetzen.
◯	Man braucht nichts zu ändern.
◯	Man soll `len` durch `*len` ersetzen.

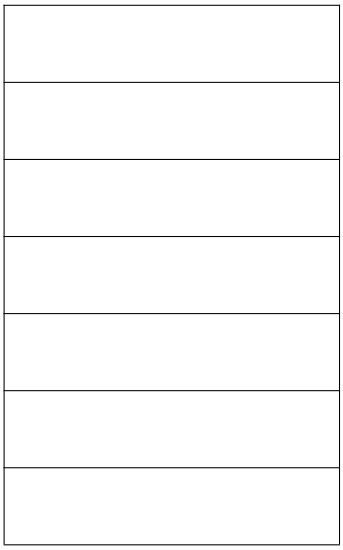
(e) (2 Punkte) Wie soll man die Verwendung von `len` in der Definition von `delete_and_shift()` (Zeilen 3-8) korrigieren?

◯	Man soll an Zeilen 3, 4 und 7 `len` durch `&len` ersetzen.
◯	Man braucht nichts zu ändern.
◯	Man soll an Zeilen 3, 4 und 7 `len` durch `*len` ersetzen.

Aufgabe 2: Datenstrukturen (/6 Punkte)

(a) (3 Punkte) Fügen Sie unter Verwendung der (in der Vorlesung vorgestellten) `push`-Funktion die folgenden Werte in der folgenden Reihenfolge in einen initial leeren Stack ein:
`12, 3, 2, 9, 42`.

Tragen Sie Ihr Ergebnis in den abgebildeten leeren Stack ein. Sie müssen keine Zwischenergebnisse angeben. Markieren Sie zusätzlich *nach dem Einfügen aller Werte* das oberste Element ("top of Stack") mit einem Pfeil.



Geben Sie die Werte an, die zurückgegeben werden, wenn Sie die in der Vorlesung vorgestellte `pop`-Funktion 5 mal auf den Stack anwenden.

Geben Sie die Worst-Case-Laufzeit einer `push`-Operation auf einem Stack mit n Elementen in $\mathcal{O}$-Notation an:

$$\mathcal{O}\left(\qquad\right)$$

Geben Sie die Worst-Case-Laufzeit einer `pop`-Operation auf einem Stack mit n Elementen in $\mathcal{O}$-Notation an:

$$\mathcal{O}\left(\qquad\right)$$

(b) (3 Punkte) Fügen Sie in einen initial leeren Suchbaum unter Verwendung der (in der Vorlesung vorgestellten) `Einfügen`-Funktion die folgenden Werte in der folgenden Reihenfolge ein:
`4, 2, 1, 5, 10, 3`.

Zeichnen Sie den resultierenden binären Suchbaum. Sie müssen keine Zwischenergebnisse angeben.

Geben Sie die Werte in der Reihenfolge an, in der sie durch die in der Vorlesung vorgestellten `Inorder-Tree-Walk`-Funktion ausgegeben werden.

Geben Sie die Worst-Case-Laufzeit der `Baumsuche`-Funktion auf einem binären Suchbaum in Abhängigkeit der Anzahl an Elementen n in $\mathcal{O}$-Notation an:

$$\mathcal{O}\left(\qquad\right)$$

Geben Sie die Worst-Case-Laufzeit der `Einfügen`-Operation auf einem binären Suchbaum Abhängigkeit der Anzahl an Elementen n in $\mathcal{O}$-Notation an:

$$\mathcal{O}\left(\qquad\right)$$

Aufgabe 3: Korrektheitsbeweis (/8 Punkte)

Der untenstehende Algorithmus berechnet den euklidischen Abstand zweier Vektoren A und B der Länge n. Dabei werden die Vektoren als Arrays übergeben: das Array $[4, 1, -2]$ entspricht dem Vektor $(4, 1, -2) \in \mathbb{R}^3$.

Die euklidische Distanz berechnet sich als $\sqrt{\sum_{i=1}^{n} \Big(A[i] - B[i]\Big)^2}$. Bei Übergabe von $A = [4, 1, -2]$ und $B = [2, 3, -1]$ berechnet sich die Distanz also zu $\sqrt{(4-2)^2 + (1-3)^2 + (-2-(-1))^2} = \sqrt{9} = 3$.
Wichtig: Nehmen Sie an, dass die übergebenen Arrays A und B immer die gleiche Länge $n \geq 1$ besitzen.

```
1  EuklidDistanz(Array A, Array B, Laenge n)
2      dist ← 0
3      for i ← 1 to n do
4          dist ← dist + (A[i] − B[i])²
5      dist ← √dist
6      return dist
```

(a) (3 Punkte) Geben Sie für den Algorithmus EuklidDistanz die Korrektheitsaussage gemäß der Antwortmöglichkeiten aus *Abbildung 2* an. Ihre Antwort muss aus genau einer Quantifizierung sowie genau einer Aussage bestehen. Quantifizierung ☐ Aussage ☐
Begründen Sie kurz (i) Ihre Wahl von Quantifizierung und (ii) Ihre Wahl von Aussage.

[(i) Quantifizierung]

[(ii) Aussage]

(A) (keine Quantifizierung)
(B) $\forall k \in \{0, \ldots, i-1\}$
(C) $\forall k \in \{0, \ldots, i\}$
(D) $\forall k \in \{0, \ldots, i+1\}$
(E) $\forall k \in \{1, \ldots, i-1\}$
(F) $\forall k \in \{1, \ldots, i\}$
(G) $\forall k \in \{1, \ldots, i+1\}$
(H) $\forall k \in \{2, \ldots, i-1\}$
(I) $\forall k \in \{2, \ldots, i\}$
(J) $\forall k \in \{2, \ldots, i+1\}$
(K) $\forall k \in \{0, \ldots, n-1\}$
(L) $\forall k \in \{0, \ldots, n\}$
(M) $\forall k \in \{0, \ldots, n+1\}$
(N) $\forall k \in \{1, \ldots, n-1\}$
(O) $\forall k \in \{1, \ldots, n\}$
(P) $\forall k \in \{1, \ldots, n+1\}$
(Q) $\forall k \in \{2, \ldots, n-1\}$
(R) $\forall k \in \{2, \ldots, n\}$
(S) $\forall k \in \{2, \ldots, n+1\}$

(1) EuklidDistanz(A,B,n) $= \sum_{j=1}^{n-1} \Big(A[j] - B[j]\Big)^2$
(2) EuklidDistanz(A,B,n) $= \sum_{j=1}^{n} \Big(A[j] - B[j]\Big)^2$
(3) EuklidDistanz(A,B,n) $= \sum_{j=1}^{n+1} \Big(A[j] - B[j]\Big)^2$
(4) EuklidDistanz(A,B,n) $= \sum_{j=k}^{n-1} \Big(A[j] - B[j]\Big)^2$
(5) EuklidDistanz(A,B,n) $= \sum_{j=k}^{n} \Big(A[j] - B[j]\Big)^2$
(6) EuklidDistanz(A,B,n) $= \sum_{j=k}^{n+1} \Big(A[j] - B[j]\Big)^2$
(7) EuklidDistanz(A,B,n) $= \sqrt{\sum_{j=1}^{n-1} \Big(A[j] - B[j]\Big)^2}$
(8) EuklidDistanz(A,B,n) $= \sqrt{\sum_{j=1}^{n} \Big(A[j] - B[j]\Big)^2}$
(9) EuklidDistanz(A,B,n) $= \sqrt{\sum_{j=1}^{n+1} \Big(A[j] - B[j]\Big)^2}$
(10) EuklidDistanz(A,B,n) $= \sqrt{\sum_{j=k}^{n-1} \Big(A[j] - B[j]\Big)^2}$
(11) EuklidDistanz(A,B,n) $= \sqrt{\sum_{j=k}^{n} \Big(A[j] - B[j]\Big)^2}$
(12) EuklidDistanz(A,B,n) $= \sqrt{\sum_{j=k}^{n+1} \Big(A[j] - B[j]\Big)^2}$

Abbildung 2: Mögliche Quantifizierungen (links) und Aussagen (rechts) für die Korrektheitsaussage.

(b) (5 Punkte) Geben Sie für die Schleife des Algorithmus EuklidDistanz (Zeilen 3-4) eine gültige Schleifeninvariante gemäß der Antwortmöglichkeiten aus *Abbildung 3* auf Seite 10 an.

Wir folgen der Konvention, dass eine Summe über eine leer Menge von Indizes den Wert 0 hat.

Ihre Antwort muss aus genau einer Quantifizierung sowie genau einer Aussage bestehen.

Quantifizierung ☐ Aussage ☐

Zeigen Sie (i) die initiale Gültigkeit Ihrer Invariante, (ii) die Erhaltung der Gültigkeit der Invariante, und (iii) dass aus der Gültigkeit der Invariante Ihre Korrektheitsaussage aus Aufgabe (a) folgt.

[(i) Initiale Gültigkeit]

[(ii) Erhalt der Gültigkeit]

[(iii) Ableitung allgemeiner Korrektheitsaussage]

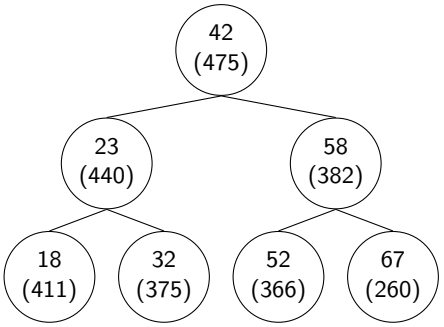
(A) (keine Quantifizierung)
(B) $\forall k \in \{0, \ldots, i-1\}$
(C) $\forall k \in \{0, \ldots, i\}$
(D) $\forall k \in \{0, \ldots, i+1\}$
(E) $\forall k \in \{1, \ldots, i-1\}$
(F) $\forall k \in \{1, \ldots, i\}$
(G) $\forall k \in \{1, \ldots, i+1\}$
(H) $\forall k \in \{2, \ldots, i-1\}$
(I) $\forall k \in \{2, \ldots, i\}$
(J) $\forall k \in \{2, \ldots, i+1\}$
(K) $\forall i \in \{0, \ldots, k-1\}$
(L) $\forall i \in \{0, \ldots, k\}$
(M) $\forall i \in \{0, \ldots, k+1\}$
(N) $\forall i \in \{1, \ldots, k-1\}$
(O) $\forall i \in \{1, \ldots, k\}$
(P) $\forall i \in \{1, \ldots, k+1\}$
(Q) $\forall i \in \{2, \ldots, k-1\}$
(R) $\forall i \in \{2, \ldots, k\}$
(S) $\forall i \in \{2, \ldots, k+1\}$

(1) $\text{dist} = \sum_{j=k}^{n-1} \Big(A[j] - B[j]\Big)^2$
(2) $\text{dist} = \sum_{j=k}^{n} \Big(A[j] - B[j]\Big)^2$
(3) $\text{dist} = \sum_{j=k}^{n+1} \Big(A[j] - B[j]\Big)^2$
(4) $\text{dist} = \sum_{j=1}^{i-1} \Big(A[j] - B[j]\Big)^2$
(5) $\text{dist} = \sum_{j=1}^{i} \Big(A[j] - B[j]\Big)^2$
(6) $\text{dist} = \sum_{j=1}^{i+1} \Big(A[j] - B[j]\Big)^2$
(7) $\text{dist} = \sqrt{\sum_{j=1}^{n-1} \Big(A[j] - B[j]\Big)^2}$
(8) $\text{dist} = \sqrt{\sum_{j=1}^{n} \Big(A[j] - B[j]\Big)^2}$
(9) $\text{dist} = \sqrt{\sum_{j=1}^{n+1} \Big(A[j] - B[j]\Big)^2}$
(10) $\text{dist} = \sqrt{\sum_{j=1}^{i-1} \Big(A[j] - B[j]\Big)^2}$
(11) $\text{dist} = \sqrt{\sum_{j=1}^{i} \Big(A[j] - B[j]\Big)^2}$
(12) $\text{dist} = \sqrt{\sum_{j=1}^{i+1} \Big(A[j] - B[j]\Big)^2}$

Abbildung 3: Mögliche Quantifizierungen (links) und Aussagen (rechts) für die Invariante.

Aufgabe 4: Treap in C (/11 Punkte)

Ein Treap ist eine Mischung aus binärem Suchbaum und Heap. In jedem Knoten werden zwei Werte `key` (oben) und `prio` (in Klammern unten) abgelegt (siehe Abbildung rechts). Betrachtet man die Werte von `key`, erfüllt der Treap die Eigenschaft eines binären Suchbaums. Betrachtet man die Werte von `prio`, so erfüllt der Treap die Max-Heap-Eigenschaft: Die `prio` eines Knotens ist immer größer als die `prio` seiner Kinder.
Um den Suchbaum zu balancieren, wird das Feld `prio` für jeden Knoten *zufällig* gewählt. Dies führt dazu, dass der resultierende Baum mit hoher Wahrscheinlichkeit balanciert ist.



Die Einfügeoperation besteht aus drei Phasen:

(1) Der neue Knoten wird erstellt und initialisiert.
(2) Der Knoten wird gemäß `key` in den binären Suchbaum eingefügt.
(3) Die Max-Heap-Eigenschaft wird nach `prio` wiederhergestellt.
Dazu wird der neue Knoten so lange hoch rotiert, bis die Heap-Eigenschaft wiederhergestellt ist: Ist der Knoten rechtes Kind, wird eine Linksrotation auf dem Elternknoten durchgeführt; ist der Knoten linkes Kind, wird eine Rechtsrotation auf dem Elternknoten durchgeführt.

Sie finden folgende Implementierung vor.

```
1  #include <stdlib.h>
2  #include <assert.h>
3  #include <stdio.h>
4
5  typedef struct treap treap_t;
6  typedef struct treapnode treapnode_t;
7
8  struct treap {
9     treapnode_t *root;
10    size_t nodes;
11 };
12
13 struct treapnode {
14    int key;
15    unsigned int prio;
16    treapnode_t *parent, *left, *right;
17 };
18
19 /* Vorabdeklaration von Hilfsfunktionen zum Einfügen in einen Treap */
20 treapnode_t *_treap_insert_initnode(int key);
21 void _treap_insert_bintree(treap_t *treap, treapnode_t *new_item);
22 void _treap_insert_fix_heap(treap_t *treap, treapnode_t *new_item);
23
24 /* Erzeugen eines leeren Treaps: Speicher allozieren und initialisieren */
25 treap_t *treap_create() {
26    treap_t *treap = calloc(1, sizeof(treap_t));
27    return(treap);
28 }
29
30 /* Einfügen Hauptfunktion */
31 void treap_insert(treap_t *treap, int key) {
32    assert(treap != NULL);
33    treapnode_t *new_item = _treap_insert_initnode(key);
34    assert(new_item != NULL);
35    _treap_insert_bintree(treap, new_item);
36    _treap_insert_fix_heap(treap, new_item);
37 }
```

```
1  /*
2   * Einfügen Hilfsfunktion:
3   * Neuen Knoten allozieren und initialisieren
4   *
5   * Gehen Sie davon aus, dass diese Funktion korrekt implementiert ist
6   * und einen validen Pointer zurückgibt
7   */
8  treapnode_t *_treap_insert_initnode(int key) { .... }
9
10 /*
11  * Einfügen Hilfsfunktion:
12  * Einfügen in binären Suchbaum gemäß Key
13  *
14  * Gehen Sie davon aus, dass diese Funktion korrekt implementiert ist
15  * und das new_item als linkes bzw. rechtes Kind in dem Binärbaum
16  * eingefügt wird und der Pointer auf den neuen Elternknoten korrekt
17  * gesetzt ist
18  */
19 void _treap_insert_bintree(treap_t *treap, treapnode_t *new_item) { ... }
20
21 /* just for testing */
22 int main(int argc, char** argv)
23 {
24    treap_t *treap = treap_create();
25    treap_insert(treap, 10);
26    treap_insert(treap, 15);
27    treap_insert(treap, 90);
28    treap_insert(treap, 60);
29    treap_insert(treap, 30);
30    treap_insert(treap, 23);
31    treap_insert(treap, 17);
32    treap_insert(treap, 64);
33
34    treap_print(treap->root);
35    printf("\n");
36 }
```

(a) (5 Punkte) Schreiben Sie die Funktion `treap_print`, die alle Keys unterhalb eines Knotens `t` (einschließlich `t`) in *absteigender* Reihenfolge ausgibt (siehe Seite 12, Zeile 34 für den Aufruf).

```
1  /* Treap in absteigender Reihenfolge ausgeben */
2  void treap_print(treapnode_t *t)
3  {
4
5
6
7
8
9
10
11 }
```

(b) (6 Punkte) Ergänzen Sie die Rechtsrotation in der Funktion `_treap_insert_fix_heap` (*siehe Seite 11, Zeile 36*). Geben Sie in den Kommentaren in Zeile 19 und 27 auf der gestrichelten Linie der Lösung an, welche Rotation Sie dort implementiert haben. Tip: Das hängt davon ab, welche Bedingung Sie in Zeile 18 einfügen.

Fangen Sie ungültige Werte für die Funktionsparameter mit `assert(<Bedingung>)` ab. `assert(<Bedingung>)` beendet das Programm, falls die Bedingung nicht zutrifft.

```
1  /* Einfügen Hilfsfunktion: Heapeigenschaft gemäß prio wiederherstellen*/
2  void _treap_insert_fix_heap(treap_t *treap, treapnode_t *new_item) {
3     /* Abfangen von ungültigen Parametern */
4
5
6
7     /* Solange die Heapeigenschaft verletzt ist, müssen wir reparieren */
8     treapnode_t *parent =
9
10    while(                                                        ) {
11       /* new_item eine Ebene im Baum hoch rotieren */
12
13
14
15
16
17       /* Unterbäume durch Rotation reparieren */
18       if(                                              ) {
19          /** Rotation :  _____________     */
20
21
22
23
24
25
26       } else {
27          /** Rotation :  _____________     */
28
29
30
31
32
33
34       }
35
36       /* Zeiger vom neuen Elternknoten auf new_item setzen*/
37       if (                                             ) {
38
39
40       } else if (                                          ) {
41
42
43       } else if (                                          ) {
44
45
46       }
47
48       parent =
49    }
50 }
```

Auf diese Seite darf nur die Aufgabe, welche auf der Vorderseite begonnen wurde, fortgesetzt werden. Für andere Aufgaben lassen Sie sich bitte ein weiteres Zusatzblatt geben.

Aufgabe 5: Dateisystem (/4 Punkte)

Sie sind auf einem Rechner mit unixoiden Betriebssystem als Benutzer `student` angemeldet und der Gruppe `studenten` zugeordnet. Damit haben Sie keine `root` Rechte.
Sie haben den Befehl `ls -lah` ausgeführt und erhalten folgende Ausgabe des Verzeichnisinhaltes:

```
1     -1-    -2-     -3-          -4-         -5-        -6-      -7-
2
3  drwxr-xr-x  4      student     studenten   4,0K Dez 24 19:10 .
4  drwxr-xr-x 11      student     studenten   4,0K Dez 24 19:10 ..
5  -rwxr-xr-x  1      root        root         12K Jan 18 19:43 file1
6  ----r-x---  1      student     studenten    12K Jan 18 19:43 file2
```

Bei den Dateien `file1` und einer Kopie `file2` in Ihrem Verzeichnis handelt es sich um fehlerfrei kompilierte Programme.

(a) (0.5 Punkte) Ist `file1` durch Sie ausführbar? Beantworten und begründen Sie.

(b) (0.5 Punkte) Ist `file2` durch Sie ausführbar? Beantworten und begründen Sie.

(c) (3 Punkte) Erklären Sie vollständig die Bedeutung der Spalten 1 und 3-7.

Auf diese Seite darf nur die Aufgabe, welche auf der Vorderseite begonnen wurde, fortgesetzt werden. Für andere Aufgaben lassen Sie sich bitte ein weiteres Zusatzblatt geben.

Aufgabe 6: Sorting (/20 Punkte)

Bei den folgenden Multiple-Choice-Fragen, ist genau eine der vorgegebenen Aussagen wahr. Bitte kreuzen Sie diese Aussage an. Ein falsch gesetztes Kreuz kann durch Zeichnen eines leeren ◯-Symbols links neben der Aussage aufgehoben werden.
Wenn eine Schranke abgefragt wird, nur eine möglichst genaue Schranke wird als Antwort akzeptiert.

(a) (2 Punkte) QuickSort: Kreuzen Sie die zutreffende Aussage an.

◯	Quicksort hat eine Best-Case-Komplexität von $O(n^2)$
◯	Quicksort hat eine Average-Case-Komplexität von $O(n\,log(n))$
◯	Wenn das Pivot-Element immer das größte oder kleinste Element im Array ist, hat Quicksort eine Worst-Case-Komplexität von $O(n^3)$.
◯	Wenn das Pivot-Element immer das größte oder kleinste Element im Array ist, hat Quicksort eine Best-Case-Komplexität von $O(n)$.

(b) (2 Punkte) MergeSort: Kreuzen Sie die zutreffende Aussage an.

◯	MergeSort hat eine Average-Case-Komplexität von $O(n^2\,log(n))$
◯	Mit MergeSort kann man die Position des Pivot-Elements anpassen.
◯	MergeSort kann sowohl rekursiv als auch iterativ implementiert werden.
◯	MergeSort hat eine Worst-Case-Komplexität von $O(n)$.

(c) (2 Punkte) InsertionSort: Kreuzen Sie die zutreffende Aussage an.

◯	InsertionSort hat eine Average-Case-Komplexität von $O(n\,log(n))$
◯	Die Invariante von InsertionSort sagt aus, dass nach jeder Iteration alle Zahlen vor dem Key (Key ausgeschlossen) sortiert sind.
◯	InsertionSort hat eine Best-Case-Komplexität von $O(\sqrt{n})$.
◯	InsertionSort hat eine Best-Case-Komplexität von $O(n\,log(n))$.

(d) (2 Punkte) Gegeben sei ein Array mit vielen Elementen (Zahlen) und einem kleinen Wertebereich. Dieses muss sortiert werden. Bewerten Sie CountSort, MergeSort und BubbleSort hinsichtlich ihrer Effizienz für diese Problemstellung.

◯	CountSort ist am effizientesten, weil es wenige Elemente gibt.
◯	MergeSort ist am effizientesten, weil das Array vorsortiert ist.
◯	BubbleSort ist am effizientesten, weil der Wertebereich klein ist und es viele Elemente gibt.
◯	CountSort ist am effizientesten, weil der Wertebereich klein ist und es viele Elemente gibt.

(e) (2 Punkte) Das Array $[1, 6, 3, 12, 5, 4]$ muss aufsteigend sortiert werden. Welche der folgenden Aussagen trifft zu?

◯	Nach einer Ausführung der inneren Schleife von InsertionSort ergibt sich das Array $[1, 6, 3, 12, 5, 4]$.
◯	Nach zwei Ausführungen der inneren Schleife von InsertionSort ergibt sich das Array $[1, 4, 3, 12, 5, 6]$.
◯	Nach einer Ausführung der inneren Schleife von BubbleSort ergibt sich das Array $[1, 5, 3, 6, 5, 4, 12]$.
◯	Nach einer Ausführung der inneren Schleife von SelectionSort ergibt sich das Array $[4, 3, 6, 5, 12, 1]$.

(f) (2 Punkte) Das Array $[4, 6, 3, 12, 5, 1]$ muss aufsteigend sortiert werden. Welche der folgenden Aussagen trifft zu?

◯	Nach einer Ausführung der inneren Schleife von InsertionSort ergibt sich das Array $[4, 6, 3, 12, 5, 1, 45]$.
◯	Nach drei Ausführungen der inneren Schleife von InsertionSort ergibt sich das Array $[3, 4, 6, 12, 5, 1]$.
◯	Nach einer Ausführung der inneren Schleife von BubbleSort ergibt sich das Array $[4, 3, 6, 5, 12, 1]$.
◯	Nach einer Ausführung der inneren Schleife von SelectionSort ergibt sich das Array $[1, 3, 6, 5, 12, 4]$.

(g) (1 Punkt) Wir betrachten in folgendem C-Code die Funktion `sort_version_1()`.

```
1  /*
2   * sortiere die gegebene Liste
3   * input_array enthält eine unsortierte Liste von ganzen Zahlen
4   * len gibt die Anzahl der Elemente in der Liste an
5   */
6  void sort_version_1(int input_array[], int len) {
7      int key;
8      int j;
9      for (int i = 0; i < len ; i++) {
10         /* Debugpunkt 1 */
11         key = input_array[i];
12         for (j = i; j > 0 && key < input_array[j-1]; j--) {
13             input_array[j] = input_array[j-1];
14         }
15         input_array[j] = key;
16     }
17     /* Debugpunkt 2 */
18 }
```

Um welchen Sortieralgorithmus handelt es sich?

◯	Insertion Sort
◯	Bubble Sort
◯	Quick Sort
◯	Count Sort
◯	Merge Sort
◯	Selection Sort
◯	Heap Sort
◯	Random Sort
◯	Slow Sort
◯	Bucket Sort
◯	Radix Sort
◯	Keines der angebenen Verfahren

(h) (1.5 Punkte) Wir bleiben bei dem letzten Beispiel `sort_version_1()`. Betrachten Sie als Eingabe das Array $[21, 42, 10, 30, 19]$. Der Wert in `len` ist auf 5 gesetzt. Geben Sie die Einzelzustände des Arrays `input_array` während des Programmablaufes an. Dabei gehen Sie von dem Zeitpunkt aus, an dem das Programm die Punkte `Debugpunkt 1` und `Debugpunkt 2` im Code erreicht hat. Geben Sie ebenso den Inhalt der Variablen `i` an, sofern existent. Ist `i` nicht existent, schreiben Sie "*n.a.*" an Stelle des Inhaltes von `i`.

i	*input_array[0]*	*input_array[1]*	*input_array[2]*	*input_array[3]*	*input_array[4]*

(i) (1 Punkt) Wir betrachten in folgendem C-Code die Funktion `sort_version_2()`.

```
1  /*
2   * Datenvorverarbeitung
3   */
4  void sort_version_2_prepare(int input_array[], int len, int array[]) {
5      for(int i = 0; i <= MAX_VALUE; i++) {
6          array[i] = 0;
7      }
8
9      for(int i = 0; i < len; ++i) {
10         array[input_array[i]] ++;
11     }
12 }
13
14 /*
15  * sortiere die gegebene Liste
16  * input_array enthält eine unsortierte Liste von ganzen Zahlen
17  * len gibt die Anzahl der Elemente in der Liste an
18  * array ist eine Liste der Länge MAX_VALUE
19  */
20 void sort_version_2(int output_array[], int input_array[], int len,
21         int array[]) {
22     int index_output = 0;
23     sort_version_2_prepare(input_array, len, array);
24
25     for(int i = 0; i <= MAX_VALUE; i++) {
26         while(array[i] > 0) {
27             output_array[index_output] = i;
28             index_output++;
29             array[i]--;
30         }
31     }
32 }
```

Um welchen Sortieralgorithmus handelt es sich?

◯	Insertion Sort
◯	Bubble Sort
◯	Quick Sort
◯	Count Sort
◯	Merge Sort
◯	Selection Sort
◯	Heap Sort
◯	Random Sort
◯	Slow Sort
◯	Bucket Sort
◯	Radix Sort
◯	Keines der angebenen Verfahren

(j) (1 Punkt) Wir betrachten in folgendem C-Code die Funktion `sort_version_3()`.

```
1  /*
2   * sortiere die gegebene Liste
3   * input_array enthält eine unsortierte Liste von ganzen Zahlen
4   * len gibt die Anzahl der Elemente in der Liste an
5   */
6  void sort_version_3(int input_array[], int len) {
7      for(int j = 0; j < len; j++) {
8          int min = j;
9          for(int i = 0; i < j + 1; i++){
10             if(input_array[i] < input_array[min]){
11                 min = i;
12             }
13
14             int tmp = input_array[min];
15             input_array[min] = input_array[j];
16             input_array[j] = tmp;
17         }
18     }
19 }
```

Um welchen Sortieralgorithmus handelt es sich?

◯	Insertion Sort
◯	Bubble Sort
◯	Quick Sort
◯	Count Sort
◯	Merge Sort
◯	Selection Sort
◯	Heap Sort
◯	Random Sort
◯	Slow Sort
◯	Bucket Sort
◯	Radix Sort
◯	Keines der angebenen Verfahren

(k) (1 Punkt) Wir betrachten in folgendem C-Code die Funktion `sort_version_4_start()`.

```
1  /*
2   * sortiere die gegebene Liste
3   * input_array enthält eine unsortierte Liste von ganzen Zahlen
4   * len gibt die Anzahl der Elemente in der Liste an
5   */
6  void sort_version_4_start(int input_array[], int len){
7      sort_version_4(input_array, 0, len - 1);
8  }
9
10 void sort_version_4(int input_array[], int p, int r){
11     if(p < r){
12         int q = (p + r) / 2;
13         sort_version_4(input_array, p, q);
14         sort_version_4(input_array, q + 1, r);
15         sort_version_4b(input_array, p, q, r);
16     }
17 }
18
19 void sort_version_4b(int input_array[], int p, int q, int r) {
20     int help_len = r - p;
21     int help[help_len];
22
23     int i = p, j = q + 1, b = 0;
24
25     while(i <= q && j <= r) {
26         if(input_array[i] <= input_array[j]){
27             help[b] = input_array[i];
28             i++;
29         }
30         else {
31             help[b] = input_array[j];
32             j++;
33         }
34
35         b++;
36     }
37
38     while(i <= q) {
39         help[b] = input_array[i];
40         b++;
41         i++;
42     }
43
44     while(j <= r) {
45         help[b] = input_array[j];
46         b++;
47         j++;
48     }
49
50     for(int k=p, c=0; k <= r; k++, c++){
51         input_array[k] = help[c];
52     }
53 }
```

Um welchen Sortieralgorithmus handelt es sich?

◯	Insertion Sort
◯	Bubble Sort
◯	Quick Sort
◯	Count Sort
◯	Merge Sort
◯	Selection Sort
◯	Heap Sort
◯	Random Sort
◯	Slow Sort
◯	Bucket Sort
◯	Radix Sort
◯	Keines der angebenen Verfahren

(l) (1 Punkt) Wir betrachten in folgendem C-Code die Funktion `sort_version_5()`.

```
1  /*
2   * sortiere die gegebene Liste
3   * input_array enthält eine unsortierte Liste von ganzen Zahlen
4   * len gibt die Anzahl der Elemente in der Liste an
5   */
6  void sort_version_5(int input_array[], int len) {
7      for(int j = len - 1; j >= 0; j--){
8          /* Debugpunkt */
9          for(int i = 0; i < j; i++){
10             if(input_array[i] > input_array[i+1]){
11                 int tmp = input_array[i];
12                 input_array[i] = input_array[i + 1];
13                 input_array[i + 1] = tmp;
14             }
15         }
16     }
17 }
```

Um welchen Sortieralgorithmus handelt es sich?

◯	Insertion Sort
◯	Bubble Sort
◯	Quick Sort
◯	Count Sort
◯	Merge Sort
◯	Selection Sort
◯	Heap Sort
◯	Random Sort
◯	Slow Sort
◯	Bucket Sort
◯	Radix Sort
◯	Keines der angebenen Verfahren

(m) (1.5 Punkte) Wir bleiben bei dem letzten Beispiel `sort_version_5()`. Betrachten Sie als Eingabe das Array $[21, 42, 35, 30, 19]$. Der Wert in `len` ist auf 5 gesetzt. Geben Sie die Einzelzustände des Arrays `input_array` während des Programmablaufes an. Dabei gehen Sie von dem Zeitpunkt aus, an dem das Programm den `Debugpunkt` im Code erreicht hat. Geben Sie auch den Inhalt der Variablen `j` an, sofern existent. Ist `j` nicht existent, schreiben Sie "*n.a.*" an Stelle des Inhaltes von `j`.

j	*input_array[0]*	*input_array[1]*	*input_array[2]*	*input_array[3]*	*input_array[4]*

Klausur-ID:	001A	**Aufgabe:**	

Auf diese Seite darf nur die Aufgabe, welche auf der Vorderseite begonnen wurde, fortgesetzt werden. Für andere Aufgaben lassen Sie sich bitte ein weiteres Zusatzblatt geben.