

Klausur Grundgebiete der Informatik 1

SS 08

(BPO07/DPO04)

Datum: 09.09.2008

Keine offizielle Lösung!

Name:
	<i>(in Druckschrift)</i>
Matr.Nr.:
Unterschrift:

Keinerlei Gewähr auf Richtigkeit!

Aufgabe	Max. Punkte	Korrektur		Einsicht	
		Punkte	Kürzel	Punkte	Kürzel
1	8				
2	9				
3	9				
4	18				
5	8				
6	10				
7	18				
8	20				
Σ	100				

Aufgabe 1 – Wissensfragen

Bitte beachten Sie, dass bei den folgenden Teilaufgaben - sofern in der Aufgabenstellung nicht anders verlangt - eine oder mehrere Antworten richtig sein können.

- (a) Nehmen Sie an, `p` sei ein Zeiger auf `int`. Das C-Code Fragment `p=(int*)malloc(8)` reserviert Speicher auf
- dem Heap
 - dem Stack
 - sowohl dem Heap, als auch dem Stack
 - dem Dateisystem
- (b) Welche der folgenden Aussagen sind korrekt?
- Ein (gerichteter) Graph $G = (V, E)$ besteht aus einer Menge V von Knoten und einer Menge E von Kanten mit $E \subseteq V \times V$
 - Gilt für alle Kanten (u, v) eines Graphen $G = (V, E)$: $(u, v) \in E \Rightarrow (v, u) \in E$ so heißt G ungerichtet.
 - Ein Baum ist ein zusammenhängender, zyklensfreier Graph
- (c) Gegeben ist die Variablendeklaration `char c`. Welche der folgenden Ausdrücke in C-Syntax prüfen, ob `c` ein Großbuchstabe ist?
- `'A' <= c <= 'Z'`
 - `(c >= 'A') | (c <= 'Z')`
 - `(c >= 'A') || (c <= 'Z')`
 - `(c >= 'A') && (c <= 'Z')`
- (d) Gegeben sind die drei Funktionen $f(n) = 100n^3 + n^2 + 1000$, $g(n) = 25n^3 + 5000n^2$, $h(n) = n^2 + 5000n \log_2 n$. Welche der folgenden Aussagen sind falsch?
- $f \in O(g)$
 - $g \in O(f)$
 - $h \in O(n^2)$
 - $h \in O(\log_2 n)$

(e) In welcher Reihenfolge wird ein binärer Suchbaum beim Preorder-Durchlauf durchlaufen?

- Wurzel, linker Teilbaum, rechter Teilbaum
- Linker Teilbaum, Wurzel, rechter Teilbaum
- Linker Teilbaum, rechter Teilbaum, Wurzel

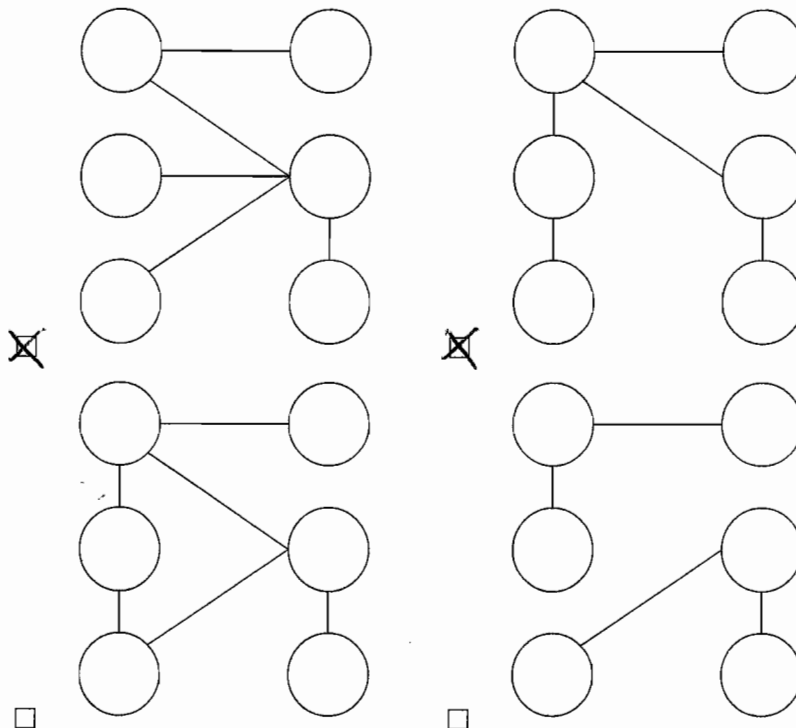
(f) Gegeben ist der folgende rekursive Algorithmus zur Berechnung der Fakultät $n!$

```
int fact(int n) { // n is greater than zero
    if (n<=0) return 1;
    else return n*fact(n-1);
}
```

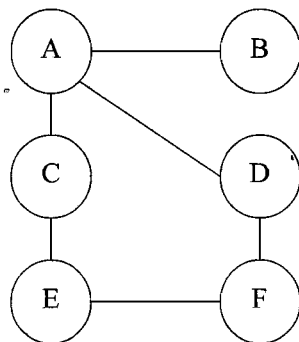
Die Anzahl der Aufrufe von `fact` bei der Berechnung von `fact(n)` ($n \geq 0$) ist:

- n
- $n + 1$
- $n + 2$
- $n - 1$

(g) Welche der folgenden Strukturen ist ein Baum?



- (h) In welcher Reihenfolge werden die Knoten des folgenden Graphen bei einer Tiefensuche vom Knoten A aus besucht?



- A, B, C, D, E, F
- A, B, C, E, D, F
- A, B, C, E, F, D
- A, B, D, F, C, E

Aufgabe 2 – Thema Fehlersuche

- a) Das folgende C-Programm soll die Werte der Variablen `a` und `b` vertauschen. Dazu benutzt es die Funktion `swap`. Finden und korrigieren Sie alle Fehler in dem folgenden Quellcode, ohne Teile des Programms neuzuschreiben.

```
1      #include <stdio.h>
2
3      void swap(int *p1, int *p2)
4      {
5          int t;
6          t = *p1;
7          *p1 = *p2;
8          *p2 = t;
9      }
10
11     int main()
12     {
13         int a = 10, b = 20;
14         printf("%d, %d\n", a, b);
15         swap(a, b); swap(&a, &b);
16         printf("%d, %d\n", a, b);
17         return 0;
18     }
```

- b) Das folgende C-Programm soll das Minimum aus a, b, c zurückliefern. Finden und korrigieren Sie alle Fehler in dem folgenden Quellcode, ohne Teile des Programms neuzuschreiben.

```
1  #include <stdio.h>
2  int main()
3  {
4      int a, b, c, min;
5      scanf("%d %d %d", &a, &b, &c);
6
7      min = c;
8      if (a < b) {
9          if (a < c)
10             min = a; }
11     else
12         if (b < c)
13             min = b;
14
15     printf("Minimum: %d\n",min);
16     return 0;
17 }
```

Aufgabe 3 – Thema Zeiger

Betrachten Sie das folgende C-Programm:

```

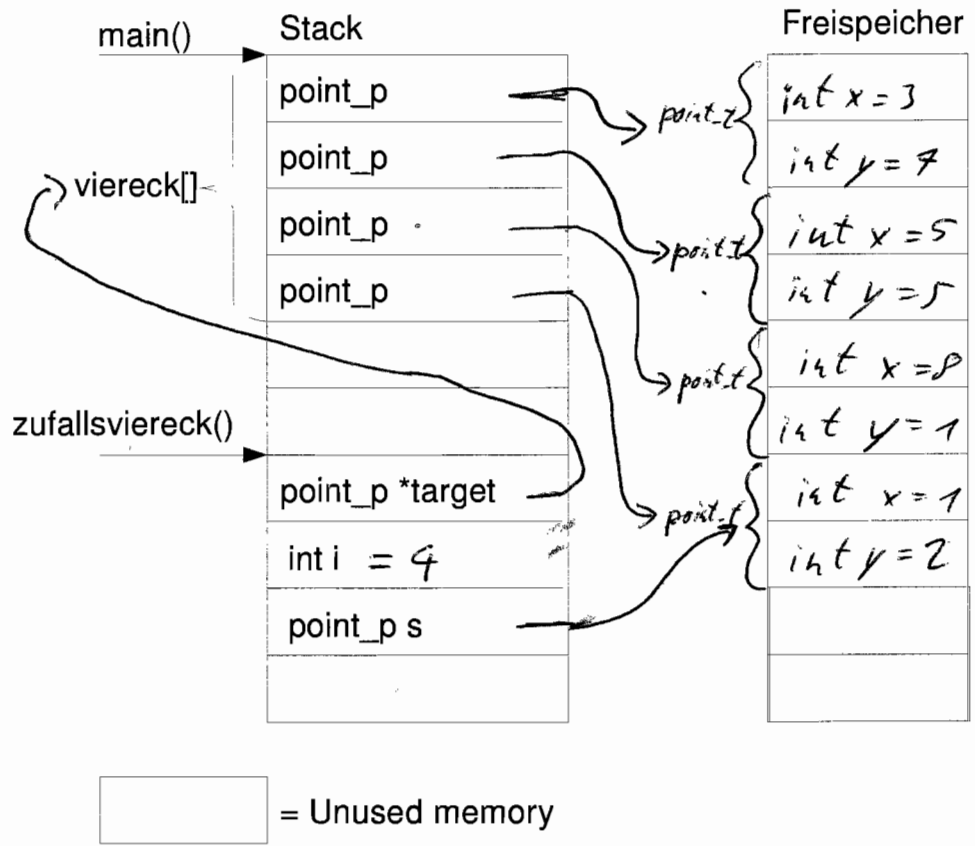
1  #include <stdlib.h> /* for malloc() */
2
3  typedef struct s {
4      int x;
5      int y;
6  } point_t, *point_p;
7
8  point_p* zufallsviereck(point_t **target){
9      int i=0;
10     point_p s;
11     for(i=0;i<4;i++){
12         s = (point_p) malloc(sizeof(point_t));
13         (*s).x = rand();
14         (*s).y = rand();
15         target[i] = s;
16     }
17     return target;
18 }
19
20 int main(int argc, char **argv){
21     point_p viereck[4];
22     zufallsviereck(viereck);
23     return 0;
24 }
```

Die Funktion `rand()` liefert Pseudo-Zufallszahlen. Die Ergebnisse für die ersten Aufrufe sind in der folgenden Tabelle angegeben.

Aufruf #N	0	1	2	3	4	5	6	7
<code>rand()</code>	3	7	5	5	8	1	1	2

Vervollständigen Sie das folgende Speicherdiagramm so dass es den Zustand *am Ende der Funktion* `zufallsviereck` darstellt. Beachten Sie dabei folgende Punkte:

- Kennzeichnen Sie Objekte, die mehrere Speicherstellen belegen, so wie in der Skizze bereits für das Array `viereck[]` geschehen.
- Geben Sie für jedes Element den Typ und, falls bekannt, den Namen an.
- Sofern Sie den Wert einer Speicherzelle kennen, tragen Sie auch diesen mit ein
- Kennzeichnen Sie für Zeigertypen die Speicherzelle, deren Adresse gespeichert ist, mit Hilfe eines Pfeils.



Aufgabe 4 – Thema Funktionsaufrufe

```
1 #include <stdio.h>
2 typedef int (*t_op)(int a, int b);
3 int add(int a, int b);
4 int sub(int a, int b);
5
6 t_op operators[] = {add, sub, add, sub};
7 int numbers[] = {3, 2, 1, 0};
8
9 int add(int a, int b)
10 {
11     return a + b;
12 }
13 int sub(int a, int b)
14 {
15     return a - b;
16 }
17
18 t_op getop(int index)
19 {
20     return operators[index];
21 }
22
23 int getnum(int index)
24 {
25     return numbers[index];
26 }
27
28 int ex(int a)
29 {
30     int result;
31     int input = getnum(a);
32     printf("%d -> %d\n", a, result = getop(a)(a, input));
33     return result;
34 }
35
36 int main()
37 {
38     printf("Operating...\n");
39     ex(ex(0));
40     return 0;
41 }
```

- a) Im folgenden sind die ersten Einträge des Call Stacks des Programms gegeben. Vervollständigen Sie das Diagramm. Berücksichtigen Sie dabei den Zustand nach jedem Funktionsaufruf sowie nach jedem Rücksprung aus einer Funktion.
- b) Was ist die Ausgabe des Programms?

steps						
0	main					
1	main	printf				
2	main					
3	main	ex(0)				
4	main	ex(0)	getnum(0)	main	ex(0)	add(0,3)
5	main	ex(0)	getnum(0)	main	ex(0)	
6	main	ex(0)	getnum(0)			
7	main	ex(0)				
8	main	ex(0)	printf			
9	main	ex(0)				
10	main					
11	main	ex(3)				
12	main	ex(3)	getnum(3)			
13	main	ex(3)				
14	main	ex(3)	getnum(3)			
15	main	ex(3)		main	ex(3)	sub(3,0)
16	main	ex(3)	printf	main	ex(3)	
17	main	ex(3)				
18	main					
19						
20						
21						
22						
23						
24						
25						

2ab)

Operations ...

0 → 3

3 → 3

Aufgabe 5 – Thema O-Notation

- a) Zeigen Sie mit Hilfe der Definition der O-Notation die folgende Aussage, indem Sie ein geeignetes $c \in \mathbb{R}$ und ein $n_0 \in \mathbb{N}$ angeben.

$$f(n) = 2n^3 + 5n \in O(n^3)$$

$$c = 3, n_0 = 3$$

$$\Rightarrow \forall n \geq n_0: 2n^3 + 5n \leq 3 \cdot n^3$$

$$\Rightarrow f \in O(n^3)$$

- b) Berechnen Sie mit Hilfe der Rechenregeln der O-Notation das Laufzeitverhalten der folgenden Funktion.

$$g(n) = \frac{(n \log n + n^2)(n^3 + 2)}{n} = (\log n + n)(n^3 + 2)$$

$$= n^4 + 2n + n^3 \log n + 2 \log n \in O(n^4)$$

Aufgabe 6 – Thema Rekursion und Iteration

Gegeben ist ein Array von Datensätzen. Jeder Datensatz hat eine eindeutige ID. Desweiteren ist das Array bereits nach aufsteigenden IDs sortiert. Sie sollen eine C Funktion schreiben, die die Existenz eines Datensatzes in einem Array überprüft. Die Signatur der Funktion ist wie folgt vorgegeben:

```
1 extern int IDs[1000];  
2 int IsThere(int InputID, int Start, int End);
```

Die Funktion `IsThere` durchsucht das Array `IDs[]` im Bereich von `IDs[Start]` bis `IDs[End-1]` nach dem Wert `InputID`. Daraus folgt, dass immer `Start < End` gelten muss. Der Rückgabewert ist 1 falls `InputID` gefunden wurde, 0 andernfalls. Sie verfügen bereits über die im folgenden angegebene rekursive Implementierung der binären Suche. Schreiben Sie eine iterative Variante der Funktion. Achten Sie dabei darauf, dass die Komplexität von $O(\log_2 N)$ erhalten bleibt.

Rekursive Version:

```
1 int IsThere(int InputID, int Start, int End)  
2 {  
3     int Current = (Start + End)/2;  
4     if (IDs[Current] == InputID)  
5     {  
6         return 1;  
7     }  
8     else if (IDs[Current] > InputID)  
9     {  
10        if(Current - 1 >= Start)  
11            return IsThere(InputID, Start, Current);  
12        else  
13            return 0;  
14    }  
15    else  
16    {  
17        if(Current + 1 <= End)  
18            return IsThere(InputID, Current, End);  
19        else  
20            return 0;  
21    }  
22 }
```

Iterative Version:

```
int IsThere(int InputID, int Start, int End)
{
    do{
        if (Start >= End)
            return 0;
        else if (IDs[(Start+End)/2] == InputID)
            return 1;
        else if (IDs[(Start+End)/2] > InputID)
            End = (Start+End)/2;
        else IDs[(Start+End)/2] < InputID
            Start = (Start+End)/2 + 1;

    }while(1);
}
```

Aufgabe 7 – Thema Heap Sort

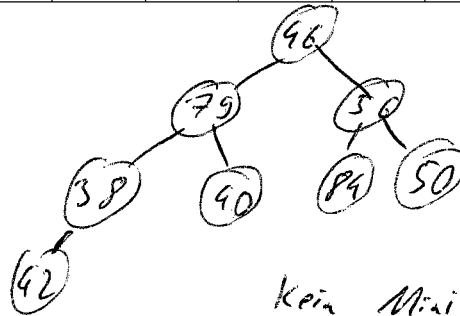
a) Heap Aufbauen

- (a) Gegeben ist der i-te Knoten A[i] (A[0] nicht benutzt). Geben Sie in der folgenden Tabelle den Index des linken und des rechten Nachfolgers an. Gehen Sie davon aus, dass der Knoten tatsächlich sowohl einen linken als auch einen rechten Nachfolger besitzt.

Name	Index
Linker Nachfolger von A[i]	2i 2i
Rechter Nachfolger von A[i]	2i+1

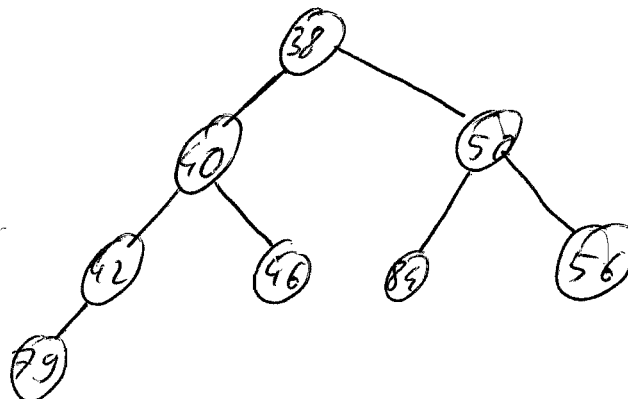
- (b) Nehmen sie an, das folgende Array stelle einen Binärbaum dar (Array-Indizes laufen hier von 1 .. N statt 0 .. N-1). Skizzieren Sie den Baum und überprüfen Sie, ob der Baum einen Minimumsheap darstellt. Begründen Sie ihre Aussage.

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
46	79	56	38	40	84	50	42



Kein Minimumsheap, da ³⁸42 < 79, jedoch ist Wert von 79 ist.

- (c) Verwandeln Sie A[] in einen Minimumsheap und skizzieren Sie den resultierenden Binärbaum. Sie brauchen die Zwischenschritte nicht explizit darzustellen.



b) Sortierschritt

- (a) Das folgende Programm implementiert den Heapsort Algorithmus. Gehen sie davon aus, dass die Zeilen 19-21 den Heap aufbauen:

```

1 void sink(int A[], int k, int N) /* A[0] nicht benutzt */
2 { int son; /* speichert kleinsten Sohn falls vorhanden */
3   while (1) /* durchlaufen bis Abbruch */
4     { if ( 2*k > N) break; /* Knoten k hat keinen Sohn */
5       if (2*k+1 <= N) /* Knoten k hat 2 Soehne */
6         { if (A[___1___] < A[___2___]) son = 2*k;
7           else son = 2*k+1;
8         }
9       else son = 2*k; /* 2k <= n, Knoten k hat 1 Sohn */
10      if (A[___3___] > A[___4___]) /* Vertauschen notwendig? */
11        { exchange(A[k],A[son]);
12          k = son; /* Knoten k evtl. weiter sinken lassen */
13        }
14      else break; /* sonst: richtige Pos. fuer k gefunden */
15    }
16  }
17 void heap_sort(int A[], int N) /* A[0] nicht benutzt */
18 { int i;
19   /* Heap
20    aufbauen
21    hier */
22   for (i = N; i > 1; i--)
23     { exchange(A[1],A[i]); /* Min. ans akt. Heap-Ende setzen */
24       sink(A,1,___5___); /* Heap reparieren */
25     }
26 }

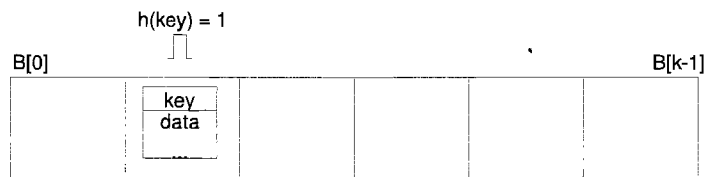
```

Füllen Sie die Lücken im oben angegebenen Programmtext. Nutzen Sie dazu die freien Felder in der unten angegebenen Tabelle.

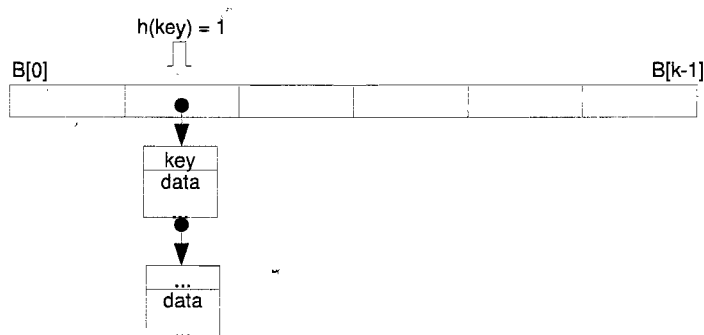
Lücke	Zeile	Antwort
1	6	2k
2	6	2k+1
3	10	k
4	10	son
5	24	i i-1

Aufgabe 8 – Thema Hashtabellen

Im Folgenden werden Sie sich eine weitere Datenstruktur zur Indizierung von Daten erarbeiten, die *Hashtabelle*. Aus dem Schlüssel eines Elements wird hier mittels einer *Hashfunktion* $h : \text{key} \rightarrow i \in [0, k - 1]$ die Position des Elements *key* in einem *Array* *B* der Größe *k* bestimmt:



Die benutzten Hashfunktionen sind in der Regel nicht injektiv. Das bedeutet, es gibt mehrere Schlüssel, die den gleichen Hashwert erzeugen, wodurch *Kollisionen* entstehen. Eine Möglichkeit mit diesen Kollisionen umzugehen ist das *offene Hashing*. Hierbei wird an Stelle des Elements eine Liste aller Elemente mit dem entsprechenden Hashwert gespeichert:



Folgendes Fragment einer Header Datei, welches eine verkettete Liste bereit stellt, ist bereits gegeben. Beachten Sie, dass im folgenden zur Vereinfachung $\text{key} == \text{data}$ gilt.

```

1  /* hashmap.h */
2  typedef struct list {
3      struct list* next;
4      unsigned short data;
5  } list_t, *list_p;
6
7  /* fuegt element hinter cursor ein */
8  void list_insert(list_p cursor, unsigned short int element);
9
10 /* Liefert "key" zurueck, falls key existiert, -1 andernfalls */
11 int list_lookup(list_p head, unsigned short int key);

```

Achten Sie in allen Unterpunkten auf korrekte Deklaration der Typen für Variablen, Parameter und Rückgabewerte. Die folgenden Aufgaben bauen aufeinander auf und sollten in der hier gegebenen Reihenfolge abgearbeitet werden. Zur Vereinfachung sollen folgende Annahmen gemacht werden.

- `g_hashtable` (s.u.) soll bereits initialisiert sein, d.h. alle Zeiger sind gültige Listen, die eventuell leer sind.
- Eine leere Liste hat genau ein Listenelement für das `data == -1` gilt.
- Die Listenfunktionen können alle eventuell auftretenden Fehler, wie z.B. ungültige Zeiger, erkennen und beheben, so dass Sie keine Fehlerbehandlung durchführen müssen.

Aufgaben:

- (a) Ergänzen Sie die Datei `hashmap.h`, welche die Datenstrukturen für ein offenes Hashing enthält. Erstellen Sie hierzu ein globales Array `g_hashtable` der Größe `BUCKETS=512` von Zeigern auf die verketteten Listen. Nutzen Sie zur Festlegung der Größe die `#define` Anweisung des Präprozessors.

```
#define BUCKETS 512
```

```
list_t g_hashtable[BUCKETS];
```

- (b) Implementieren Sie die Funktion `hashtable_lookup`, welche einen Schlüsselwert `key` als Argument erhält und das gesuchte Element zurückliefert oder `-1` falls das Element nicht gefunden wurde. Nehmen Sie die Hashfunktion

```
unsigned int h(unsigned short int key);
```

als gegeben an.

```
unsigned short int hashtable_lookup (unsigned short int key) {
unsigned int h = h(key);
return list_lookup (hashtable[h(key)], key);
}
```

- (c) Implementieren Sie nun die Funktion `hashtable_insert`, welche ein Element `element` in die Hashtabelle einfügt.

```
void hashtable_insert (unsigned short int key) {
list_insert (hashtable[h(key)], key);
list_p (p = hashtable[h(key)];
while (p != NULL)
    p = p->next;
list_insert (p, key);
}
```

- (d) Betrachten sie nun die Komplexität der Funktionen für den schlechtesten Fall, dass h immer den gleichen Index zurückliefert. Nehmen Sie an, dass die Komplexität der Funktionen h , `list_lookup` und `list_insert` jeweils $O(1)$ ist. Geben Sie zuerst für diesen Fall die Länge der entsprechenden Liste in Abhängigkeit von der Anzahl der Datenelemente M an. Geben Sie nun die Komplexität von `hashtable_insert` und `hashtable_lookup` an.

Länge der entsprechenden Liste wäre M .

Komplexität von `-hashtable_insert` = $O(M)$

`-hashtable_lookup` = $O(1)$

- (e) Gehen Sie nun davon aus, dass die Funktion h alle Indizes $k \in [0, \text{BUCKETS} - 1]$ mit der gleichen Häufigkeit erzeugt. D.h., die Länge der einzelnen Listen $L_0 \dots L_{\text{BUCKETS}-1}$ ist $\frac{n}{\text{BUCKETS}}$. Gehen Sie außerdem davon aus, dass `BUCKETS` so gewählt wurde, dass $\frac{n}{\text{BUCKETS}} < c$ gilt, wobei c eine kleine Konstante ist. Geben Sie für diesen Fall die Komplexität der Funktionen `hashtable_insert` und `hashtable_lookup` an.

`hashtable_insert` = $O\left(\frac{n}{\text{buckets}}\right) \ll O(c) \approx O(1)$