Department of Informatics

Prof. Dr. Michael Böhlen
Binzmühlestrasse 14
8050 Zurich
Phone: +41 44 635 4333
Email: boehlen@ifi.uzh.ch

University of Zurich[UZH]

Informatics II — Final Exam
Spring 2019 — 27.05.2019

Name: _____ Matriculation number: _____

Advice

You have 90 minutes to complete the exam of Informatics II. The following rules apply:

- Answer the questions in the space provided.
- Additional sheets are provided upon request. If you use additional sheets, put your name and matriculation number on each of them.
- Check the completeness of your exam (17 numbered pages).
- Use a pen in blue or black colour for your solutions. Pencils and pens in other colours are not allowed. Solutions written in pencil will not be corrected.
- Stick to the terminology and notations used in the lectures.
- Only the following auxiliary material is allowed for the exam:
  - One A4 sheet (2-sided) with your personal notes (handwritten/ printed/ photocopied).
  - A foreign language dictionary is allowed. The dictionary will be checked by a supervisor.
  - A pocket calculator without text storage (memory) like TI-30 XII B/S.

  Computers, pdas, smart-phones, audio-devices or similar devices may not be used. Any cheating attempt will result in a failed test (meaning 0 points).
- Put your student legitimation card ("Legi") on the desk.

Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	5	Total
Points Achieved						
Maximum Points	14	9	9	8	10	50

Exercise 1 — 8 + 1 + 3 + 2 Points

1.1 Tick the correct box to indicate whether the following statements are True of False. No explanations are needed.

i. Consider the recurrence $T(n) = 3T(n/3)+\log n$. Its asymptotic complexity is $T(n) = \Theta(n)$.

True	False
X	

ii. Let T be a complete binary tree with n nodes. Assume we use breadth first search to find a path from the root to a given vertex $v \in T$. The asymptotic runtime complexity of this search is $O(\log n)$.

True	False
	X

iii. Consider an unsorted array $A[1...n]$ of n integers. Building a max-heap out of the elements of A can be done asymptotically faster than building a red-black tree with the elements of A.

True	False
X	

iv. The array $A = [20, 15, 18, 7, 9, 5, 12, 3, 6, 2]$ represents a max-heap.

True	False
X	

v. Every binary search tree with n nodes has height $O(\log n)$.

True	False
	X

vi. Let $G = (V, E)$ be a weighted graph and let M be a minimum spanning tree of G. The path in M between a pair of vertices v_1 and v_2 does not have to be a shortest path in G.

True	False
X	

vii. Assume an array contains n numbers that are either -1, 0, or 1. Such an array can be sorted in $O(n)$ time in the worst case.

True	False
X	

viii. In a doubly linked list with 10 nodes, we have to change 4 pointers of the list if we want to delete a node other than the head node and tail node.

True	False
	X

1.2 Assume hash table `H1` uses `linear probing` with step size 1. The hash function is `h(k) = k mod 9`.

0	1	2	3	4	5	6	7	8
9	18		12	3	14	4	21	

In which order could the elements have been added to the hash table? Identify all correct answers. Assume that the hash table has never been resized, and no elements have been deleted.

a) $9, 14, 4, 18, 12, 3, 21$

b) $12, 3, 14, 18, 4, 9, 21$

c) $12, 14, 3, 9, 4, 18, 21$

d) $9, 12, 14, 3, 4, 21, 18$

e) $12, 9, 18, 3, 14, 21, 4$

c, d

1.3 Consider a hash table $H2$ that uses chaining for collision resolution. $H2$ has x slots. k items are stored in the hash table. Answer the following questions.

i. What is the load factor of $H2$?

$\frac{k}{x}$

ii. Is it necessary that the value of k is smaller than the value of x?

No

iii. What is the average number of items per slot?

$\frac{k}{x}$

iv. In the worst case how many items could be in a single slot?

k

v. Is it possible that $k > x$ and some slots are empty?

Yes

vi. In worst case, what would be asymptotic complexity of successfully deleting `N` items from `H2`?

$O(kN)$

vii. In worst case, what would be the asymptotic complexity of unsuccessfully deleting `N` items from `H2`?

$O(kN)$

1.4 Consider a hash table $H3$ that has N slots and uses linear probing for collision resolution. Answer the following questions.

(a) In the best case what is the asymptotic complexity of adding N elements to $H3$?

$O(N)$

(b) In the worst case what is the asymptotic complexity of adding N elements to $H3$?

$\mathrm{O}(\frac{N(N-1)}{2})$

Name: Matriculation number:

Exercise 2 — 2 + 1 + 6 Points

Run depth first search (DFS) on the graph in Figure 1. Assume the search starts at node C. If in a step multiple nodes can be chosen, select the node that comes first in alphabetical order according to its key.

2.1 Write down the start and end timestamps that each node gets assigned during the DFS.

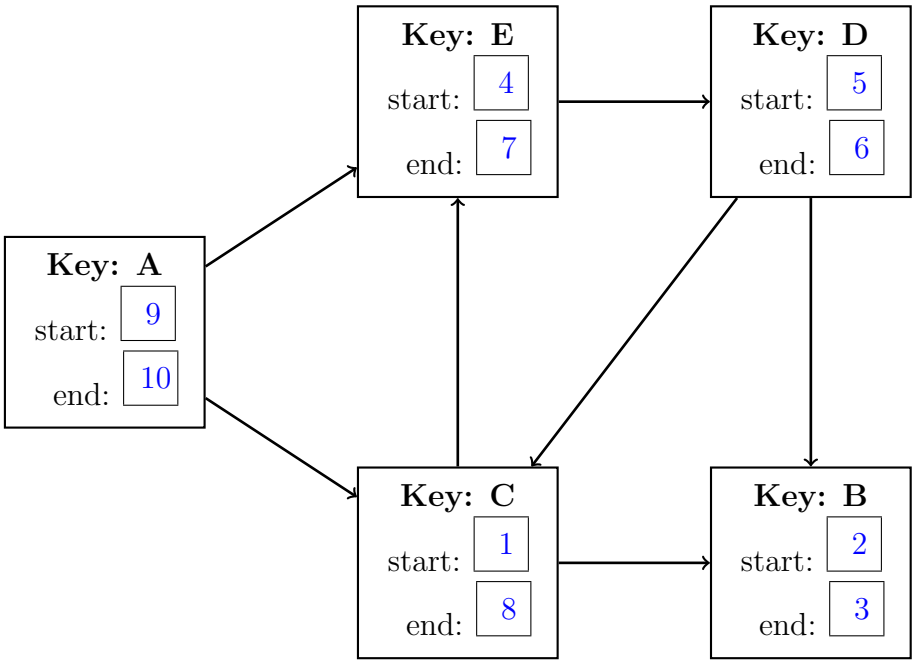


Figure 1: Directed Graph

2.2 Write down the resulting parenthesis structure defined by the depth first search.

(C (B B) (E (D D) E) C) (A A)

2.3 A graph is said to have an `Euler Path` if

- it is a connected graph, and
- it contains no more than two vertices of odd degree.

Given an undirected graph $G = (V, E)$, write the pseudo code of an algorithm that determines if graph G has an Euler Path.

```
1  Algorithm: BFS(G)
2  s.col = G;
3  InitQueue(Q);
4  Enqueue(Q,s);
5  while Q ≠ φ do
6      v = Dequeue(Q);
7      foreach u ∈ v.adj do
8          u.degree ++;
9          if u.col == W then
10             u.col = G; Enqueue(Q,u);
11     v.col = B;
```

```
1  Algorithm: hasEulerPath(G)
2  foreach v ∈ G.V do
3      v.col = W; v.degree = 0;
4  s = RandomVertex(G);
5  G = BFS(G,s);
6  oddCount = 0;
7  foreach v ∈ G.V do
8      if v.degree mod 2 ==1 then
9          oddCount++;
10     if oddCount > 2 or v.degree == 0 then
11         return False;
12 return True;
```

Name: Matriculation number:

Exercise 3 — 3 + 6 Points

Given are two disjoint sorted linked lists `A` and `B` as illustrated in Figure 2.

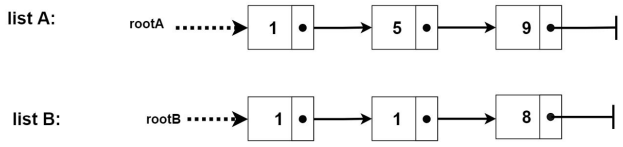


Figure 2: Input linked-lists list A and list B

Implement a function `merge(struct node** rootA, struct node** rootB)` with the following properties:

a. The merged list must be sorted as well.

b. You are not allowed to use an auxiliary data structure whose size depends on the number of elements in the original list. The merging shall be implemented by modifying pointers.

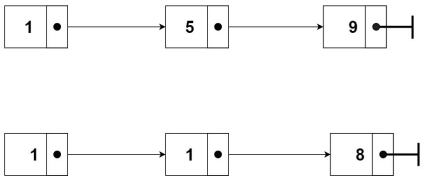
c. After the merging `rootA` and `rootB` both must point to the head of sorted merge linked list.

Use the following data structure and variables for your solution.

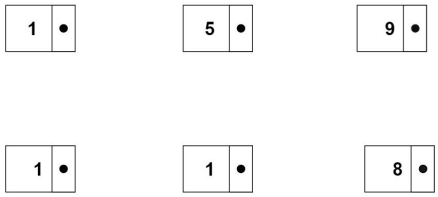
```
1  struct node {
2    int val;
3    struct node* next;
4  };
5
6  struct node* rootA;
7  struct node* rootB;
```

3.1 Illustrate step by step how will you merge the linked lists. Draw all pointers that are used and modified at each step.

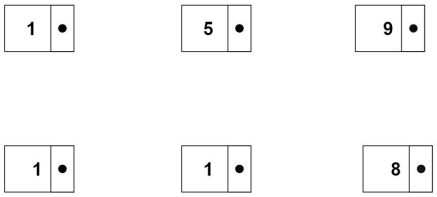
i.



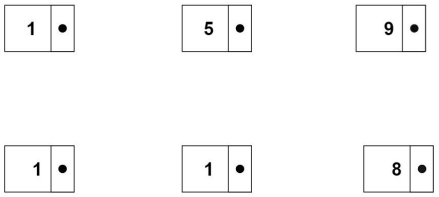
ii.



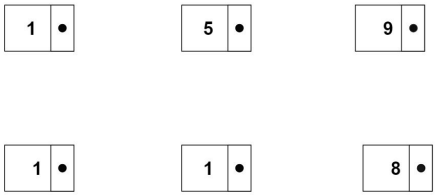
iii.



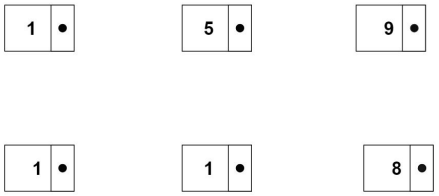
iv.



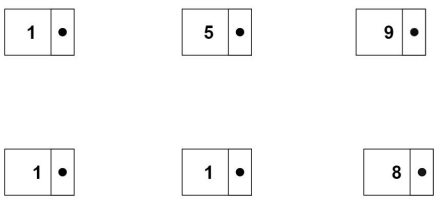
v.



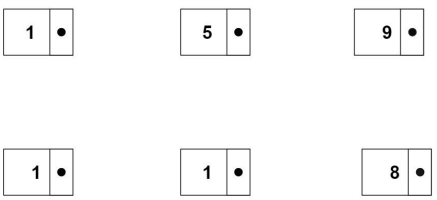
vi.



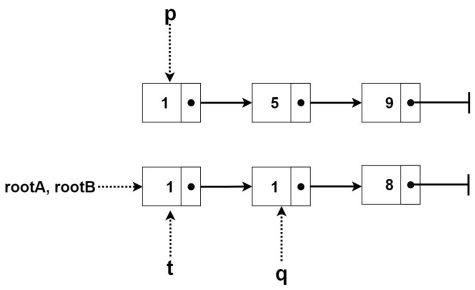
vii.



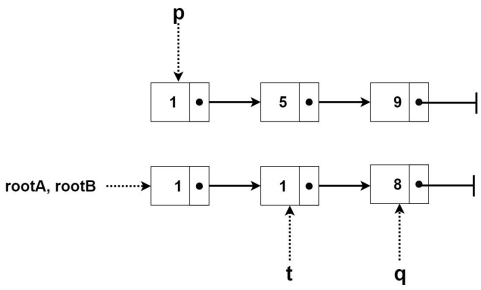
viii.



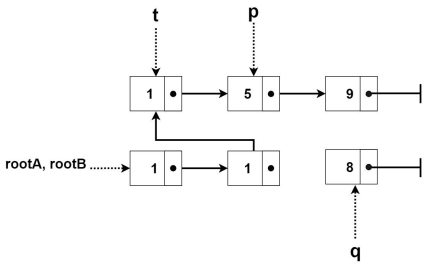
start of loop; 1st iteration



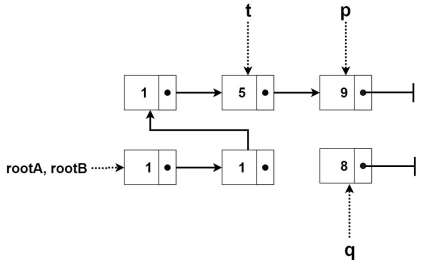
start of loop; 2nd iteration



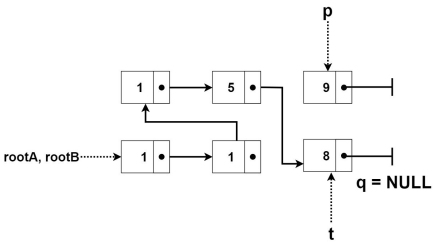
start of loop; 3rd iteration



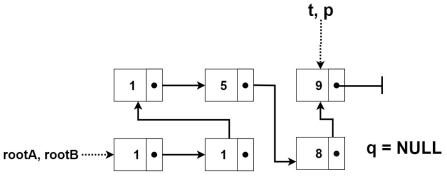
start of loop; 4th iteration



start of loop; 5th iteration



After Loop; If condition



3.2 State the pseudocode for function `merge(struct node** rootA, struct node** rootB)`. State precisely how the function must be called.

```
1  Algorithm: merge(struct node** rootA, struct node** rootB)
2  if *rootA == NULL and *rootB == NULL then
3      return;
4  else if *rootA == NULL or *rootB ≠ NULL then
5      rootA = rootB; return;
6  else if *rootB == NULL and *rootA ≠ NULL then
7      *rootB = *rootA; return;
8  else if (*rootA)→val < (*rootB)→val then
9      t = *rootA; p = (*rootA)→next; q = *rootB; *rootB = *rootA;
10 else
11     t = *rootB; q = *rootB→next; p = *rootA; *rootA = *rootB;
12 while p ≠ NULL and q ≠ NULL do
13     if p→val < q→val then
14         t→next = p;
15         p = p→next;
16     else
17         t→next = q;
18         q = q→next;
19     t = t→next;
20 if p ≠ NULL then
21     t→next = p;
22 if q ≠ NULL then
23     t→next = q;
```

call: merge(&rootA, &rootB)

Name: Matriculation number:

Exercise 4 — 6 + 2 Points

4.1 A `d-ary` heap is defined like a binary heap except that nodes have up to `d` children. Answer the following questions.

a. Consider the node with index i. What is the index of its parent node?

$\lfloor \frac{i-2}{d} + 1 \rfloor$

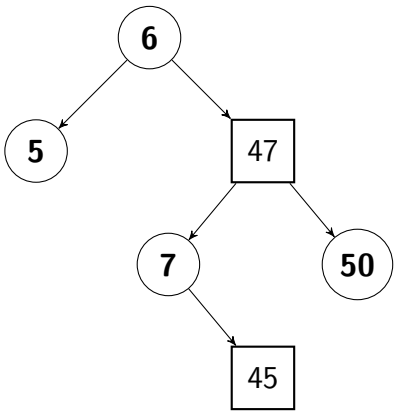
b. Consider the node with index i. Assume $1 \leq j \leq d$. What is the index of the jth child node of node i?

d*(i-1) + j + 1

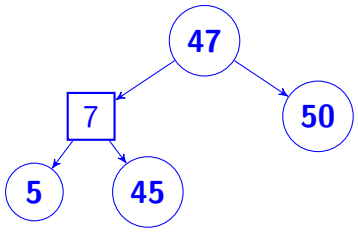
c. What is the height of a `d-ary` heap with n elements?

$\theta(\log_d n)$

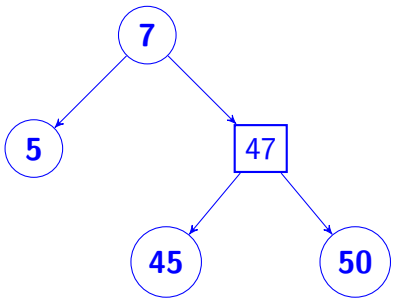
4.2 Consider following red-black tree where black nodes are denoted with a circle and red nodes are denoted with a square. Delete **6** from the tree and draw the resulting red-black tree.



Solution A:



Solution B:



Name: Matriculation number:

Exercise 5 — 2 + 3 + 5 Points

Assume a rectangle of size $n \times m$. Determine the minimum number of squares that is required to tile the rectangle. The side length of rectangles must be an integer. Figure 3 shows a rectangle of size 6×5 that requires tiles to tile it with the minimum number of tiles.

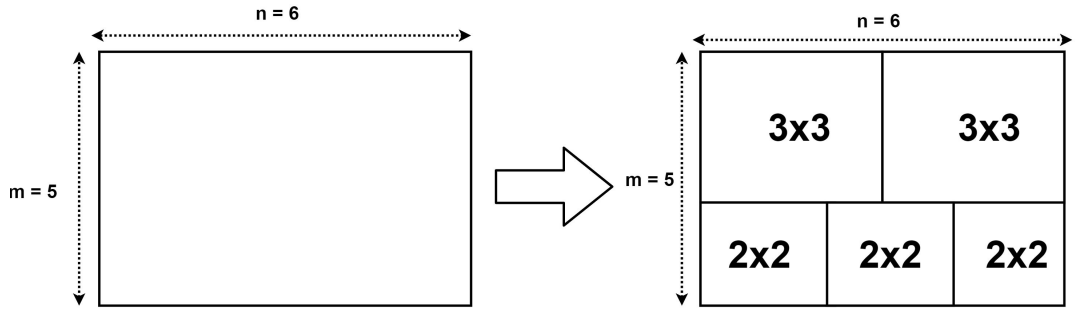


Figure 3: Minimum number of square tiles for a rectangle of size 5x6

The idea for a dynamic programming solution is to recursively split a rectangle of size $(n \times m)$ into either two horizontal rectangles $((n \times h)$ and $(n \times (m - h))$ or two vertical rectangles $((v \times m)$ and $((n - v) \times m))$ where $h = 1, 2, ..., m - 1$ and $v = 1, 2, ..., n - 1$. Then finding the minimum squares that fill these splits.

5.1 Assume $MinSquare(n, m)$ denotes the miminum squares that is required to tile a rectangle of size $n \times m$. State a recursive definition of $MinSquare(n, m)$.

$$MinSquare(n,m) = \begin{cases} 1 & \text{if } n = m \\ min \begin{cases} min_{h=1..m-1}(MinSquare(n,h) + MinSquare(n,m-h)) \\ min_{v=1..n-1}(MinSquare(v,m) + MinSquare(n-v,m)) \end{cases} & \text{otherwise} \end{cases}$$

5.2 To efficiently compute the minimum squares that is required to tile a rectangle of size $n \times m$, a dynamic programming solution with 2D array $dp[0...n][0...m]$ can be used. Element $dp[i][j]$ is the minimum number of squares that is required to tile a rectangle of size $i \times j$. Complete the 2D array below with the minimum squares required to tile rectangles up to size 4×3.

m \ n	0	1	2	3	4	
	0					
	1	1	2	3	4	
	2	2	1	3	2	
	3	3	3	1	4	

5.3 For a rectangle of size $n \times m$, specify an algorithm that uses a dynamic programming solution to compute the minimum number of squares that is required to tile the rectangle. You can use either C or pseudocode for your solution.

```
1  Algorithm: InitArray(n,m)
2  let dp[0..n][0..m] be a new array
3  for i=0 to n do
4  |   for j=0 to m do
5  |   |   dp[i][j] = ∞;
6  return MinSquare(n, m, dp);
```

```
1  Algorithm: MinSquare(n, m, dp)
2  horizontalMin = ∞;
3  verticalMin = ∞;
4  if n == m then
5  |   dp[m][n] = 1;
6  |   return 1;
7  if dp[n][m] then
8  |   return dp[n][m];
9  for i = 1 to n do
10 |   verticalMin = min(MinSquare(i, m, dp) + MinSquare(n-i, m ,dp),
   |     verticalMin)
11 for i = 1 to m do
12 |   horizontalMin = min(MinSquare(n, i, dp) + MinSquare(n, m-i ,dp),
   |     horizontalMin)
13 dp[n][m] = min(verticalMin, horizontalMin);
14 return dp[n][m];
```