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}

AlgoDat — Repetition Exam

Spring 2018 — 18.05.2018

Name: ______________________ Matriculation number: ____________

Advice

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

- Answer the questions on the exam sheets or the backside. Mark clearly which answer belongs to which question. 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 (16 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 materials are allowed for the exam:
  - One A4 sheet (2-sided) with your personal handwritten notes, written by yourself. Sheets that do not conform to this specification will be collected.
  - A foreign language dictionary is allowed. The dictionary will be checked by a supervisor.
  - A pocket calculator.
  - No additional items are allowed. 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	Total
Points Achieved					
Maximum Points	15	20	10	10	55

Sorting

1.1 [3 points] Consider the array given in the following table. In each of the lines, show the form of this array after an iteration of the outer loop of an insertion sort (sorting in ascending order). Perform this task for the first three iterations.

A[1]	**A**[2]	**A**[3]	**A**[4]	**A**[5]	**A**[6]	**A**[7]	**A**[8]	**A**[9]	**A**[10]
5	3	4	9	1	7	0	2	6	8
3	5	4	9	1	7	0	2	6	8
3	4	5	9	1	7	0	2	6	8
3	4	5	9	1	7	0	2	6	8

1.2 [4 points] Given an array `A` of `n` integers, use C or pseudocode to write a recursive version of insertion sort to sort `A` in descending order.

```
Algorithm: recursiveInsertionSort(A, n)
if $n > 1$ then
    /* Sort all but the last element */
    recursiveInsertionSort(A, n-1);
    /* Insert A[n] to the sorted A[1..n-1] */
    j = n - 1;
    t = A[n];
    while $j > 0$ and $A[j] < t$ do
        A[j+1] = A[j];
        j = j - 1;
    A[j+1] = t;
```

1.3 [3 points] Calculate the asymptotic tight bound of the following recurrence. If the Master Theorem can be used, write down a, b, $f(n)$ and the case (1-3).

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

$a = 3,\ b = 2,\ f(n) = n^2,\ n^{log_b a} = n^{log_3 2} = n^{1.58}$
$f(n) = \Omega(n^{log_b a + \epsilon}), \epsilon > 0$

Case 3: $\Theta(n^2)$
Regularity condition: if $c = \frac{3}{4}, a \cdot f(n/b) = 3 \cdot (\frac{n}{2})^2 \leq \frac{3}{4} \cdot n^2$

$T(n) = \Theta(n^2)$

1.4 [5 points] Use repeated (backward) substitution to calculate the asymptotic upper bound of the following recurrence.

$$T(n) = \begin{cases} 1, & n = 1 \\ 8T(\frac{n}{2}) + n^2 & n > 1 \end{cases}$$

$$\begin{aligned}
T(n) &= 8T(n/2) + n^2 \\
&= 8(8T(n/4) + (n/2)^2) + n^2 \\
&= 8^2T(n/2^2) + 2n^2 + n^2 \\
&= 8^2(8T(n/2^3) + (n/2^2)^2) + 2n^2 + n^2 \\
&= 8^3T(n/2^3) + 4n^2 + 2n^2 + n^2 \\
&= \ldots \\
&= 8^iT(n/2^i) + n^2(1 + 2 + 4... + 2^{i-1}) \\
&= (2^i)^3T(n/2^i) + n^2\sum_{k=0}^{i-1} 2^k \\
&= (2^i)^3T(n/2^i) + n^2(2^i - 1) \\
For\ i = \log n \Rightarrow T(n) &= (n)^3T(1) + n^2(n - 1) \\
&= n^3 + n^3 - n^2 \\
&= 2n^3 - n^2 \\
&= \Theta(n^3)
\end{aligned}$$

Exercise 2 20 Points

ADTs

2.1 [4 points] Consider a **circular FIFO queue** of integers with size 5. Illustrate the contents of this queue and mark its head and tail, after applying the following operations:

– draw the empty queue

(h) (t)				

– enqueue(10), enqueue(20), enqueue(30) [0.2 each element - 0.2 tail - 0.2 head]

10 (h)	20	30 (t)		

– dequeue(), enqueue(40), enqueue(50) [0.2 each element - 0.2 tail - 0.2 head]

	20 (h)	30	40	50 (t)

– enqueue(60), dequeue() [0.2 each element - 0.2 tail - 0.2 head]

60 (t)		30 (h)	40	50

The goal of this exercise is to implement a FIFO queue using two LIFO stacks.

2.2 [10 points] Given two stacks (A and B), illustrate their states and the sequence of operations (push, pop) applied on each in order to mimic the FIFO behaviour of the queue in task 2.1.

- enqueue(10), enqueue(20), enqueue(30) [3 points]

A	B
	30
	20
	10

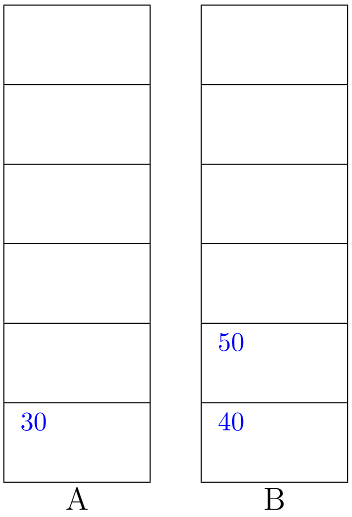
stack	operation	element
B	push	10
B	push	20
B	push	30

- dequeue(), enqueue(40)[5 points]

A	B
20	
30	40

stack	operation	element
B	pop	-
A	push	30
B	pop	-
A	push	20
B	pop	-
A	push	10
A	pop	-
B	push	40

- enqueue(50), dequeue() [2 points]



stack	operation	element
B	push	50
A	pop	-

2.3 [6 points] Consider a `LIFO stack` defined as follows:

```
    struct Node {
        int key;
        struct Node* next;
    };
    struct Node* myStack;
```

Assume the following operations are available:

- *int isEmpty(struct Node* S)* which checks if stack S is empty or not.
- *void push(struct Node** S, int key)* which inserts element `key` into stack S.
- *int pop(struct Node** S)* which removes the last inserted element from stack S and returns its value, in case of an empty stack -1 is returned.
- *int top(struct Node** S)* which return the element on the `top` of the stack S, in case of an empty stack -1 is returned.

Write a function `sortStack(struct Node** S)` that sorts the stack S in **descending** order (largest element is on the top), using **only** one additional `temporary` stack. The original stack S can be modified and either stack S or the *temporary stack* can be returned as the sorted stack. Use either C or pseudocode for your solution.

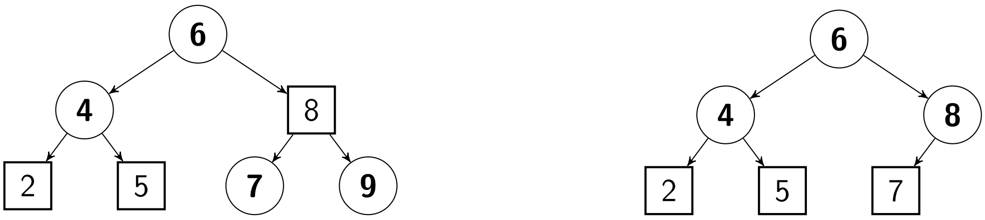
```
struct Node* sortStack(struct Node** S) {
    struct Node* tmpStack = NULL;

    while (!isEmpty(*S)) {
        int tmp = pop(S);
        while (!isEmpty(tmpStack) && top(&tmpStack) > tmp){
            push(S, pop(&tmpStack));
        }
        push(&tmpStack, tmp);
    }
    return tmpStack;
}
```

Name: Matriculation number:

Exercise 3	10 Points

Trees

3.1 [6 points] Consider the red-black tree in Figure 1 where black nodes are denoted with a circle and red nodes are denoted with a square. Table 1 illustrates the cases and actions if value 9 is deleted from red-black tree of Figure 1a. Figure 1b shows the resulting tree.



(a) Before deletion of 9

(b) After deletion of **9**

Figure 1: Deleting 9 from a red-black tree

Case	Action	Arguments
Case 2m	assign color	7, red
	set x equal to	8
Case 0	assign color	8, black

Table 1: Delete cases and operations

1. Delete the node with key **16** from the red-black tree in Figure 2. Draw the resulting tree and fill in Table 2 with the cases and actions applied.

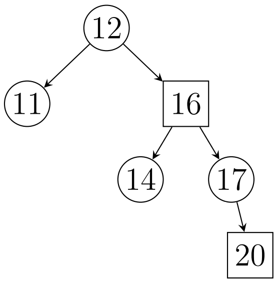
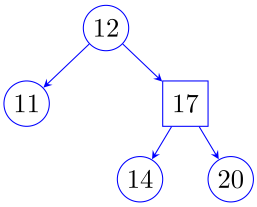


Figure 2



Case	Action	Arguments
Case 4	assign color	17, red
	assign color	14, black
	assign color	20, black
	left rotate	14

Table 2: Delete cases and operations

2. Delete the node with key **13** from the red-black tree in Figure 3. Draw the resulting tree and fill in Table 3 with the cases and actions applied.

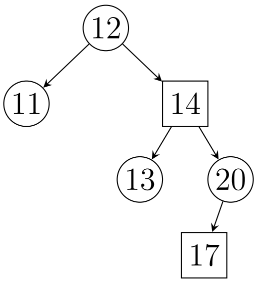
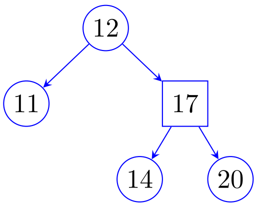


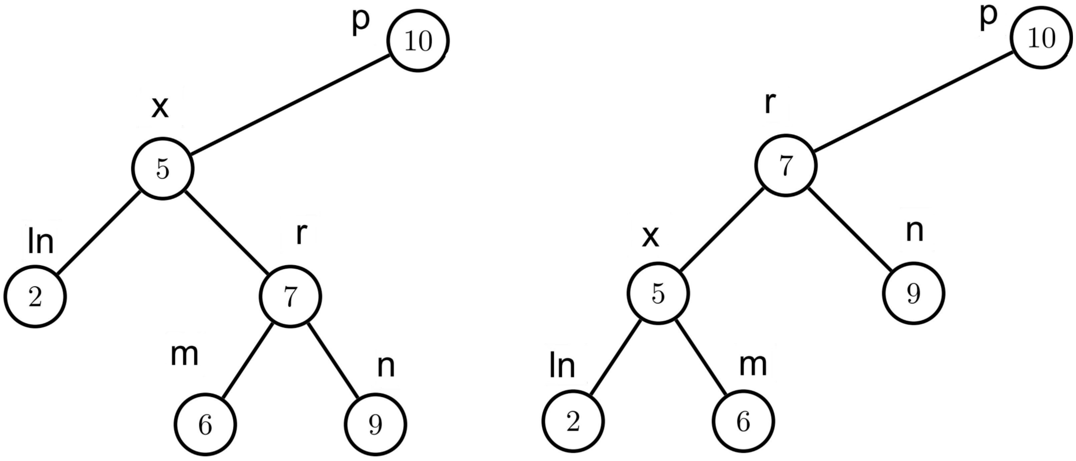
Figure 3



Case	Action	Arguments
Case 3	assign color	17, black
	assign color	20, red
	right rotate	20
Case 4	assign color	17, red
	assign color	14, black
	assign color	20, black
	left rotate	14

Table 3: Delete cases and operations

3.2 [4 points] Figure 4 illustrates **left rotation** of node x in a binary search tree.



(a) Before left rotation of **x**

(b) After left rotation of **x**

Figure 4: Left Rotation of node **x**

Write a function `leftrotate(struct BSTNode* x)` that does a **left rotation** of a node x in a binary search tree. Assume the binary search tree has the structure given below. You can use C or pseudocode.

```
struct BSTNode {
    int key;
    BSTNode * left;
    BSTNode * right;
    BSTNode * parent;
};
struct BSTNode* root;
```

```
void leftRotate(struct BSTNode* x) {
    struct BSTNode* r;

    r = x->right;
    x->right = r->left;
    r->parent = x->parent;

    if (r->left != NULL) {
        (r->left)->parent = x;
    }

    if (x->parent == NULL){
        root = r;
    } else {
        if (x == (x->parent)->left) {
            (x->parent)->left = r;
        } else {
            (x->parent)->right = r;
        }
    }

    r->left = x;
    x->parent = r;
}
```

Exercise 4 10 Points

Dynamic Programming

4. [10 points] The `Maximum Sum Increasing Subsequence` algorithm finds the sum of a maximum increasing subsequence of a given **unsorted** array `A[0..n-1]` of integers, such that the integers in the subsequence are sorted in **increasing** order.
   A **subsequence** is a sequence that can be derived from another sequence by deleting zero or more elements without changing the order of the remaining elements.

   For array `A = [1, 101, 2, 3, 100, 4, 5]` examples of **increasing** subsequences are {`1, 101`}, {`2, 3, 100`} and {`1, 2, 3, 4, 5`}.

   Executing the `Maximum Sum Increasing Subsequence` algorithm for array `A`, the result is `106` which is the sum of the `max increasing subsequence S =` {`1, 2, 3, 100`}.

   a) [3 points] In order to efficiently compute the sum of a maximum increasing subsequence, a dynamic programming solution with array sum[0...n-1] can be used. Element sum[i] in array sum contains the maximum sum of increasing sequence at index i. Given an array `A = [5, 14, 3, 22, 1, 32, 51, 9]`, fill the table below with the maximum sum increasing subsequence at each index.

i	0	1	2	3	4	5	6	7
A[i]	5	14	3	22	1	32	51	9
sum[i]	5	19	3	41	1	73	124	14

b) [7 points] Design a **Dynamic Programming** algorithm that finds and returns the sum of the maximum increasing subsequence of an unsorted array. Use either C or pseudocode for your solution.

```
int maxSumIS (int arr[], int n) {
    int i;
    int j;
    int max = 0;
    int msis[n];

    for (i = 0; i < n; i++) {
        msis[i] = arr[i];
    }

    for (i = 1; i < n; i++) {
        for (j = 0; j < i; j++) {
            if (arr[i] > arr[j] && msis[i] < msis[j] + arr[i])
              {
              msis[i] = msis[j] + arr[i];
            }
        }
    }

    for (i = 0; i < n; i++) {
        if (max < msis[i]) {
            max = msis[i];
        }
    }
    return max;
}
```

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 2018 28.05.2018

Name: ______________________ Matriculation number: ______________

Advice

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

- Answer the questions on the exam sheets or the backside. Mark clearly which answer belongs to which question. 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 (19 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 materials are allowed for the exam:
  - One A4 sheet (2-sided) with your personal handwritten notes, written by yourself. Sheets that do not conform to this specification will be collected.
  - A foreign language dictionary is allowed. The dictionary will be checked by a supervisor.
  - No additional items are allowed except calculators. 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	Total
Points Achieved					
Maximum Points	22	15	18	15	70

Exercise 1 22 Points

1.1 [2 points] Let T be a binary search tree without duplicates. The lowest common ancestor of two nodes $n_1 \in T$ and $n_2 \in T$ is the node in T with the largest depth that has both n_1 and n_2 as descendants. Each node is a descendant of itself.

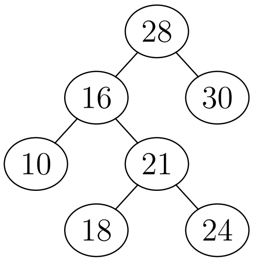


Figure 1: Binary search tree

Given the binary search tree in Figure 1, determine the lowest common ancestor of the following pairs of nodes:

i) 18 and 10: 16

ii) 24 and 16: 16

1.2 [8 points] Assume a binary search tree `T` with integers as keys. Each node has pointers to its children but no pointer to its parent. Use C to define `T`. Use C or pseudocode to describe an algorithm that takes two nodes of the tree as arguments and returns the pointer to the lowest common ancestor.

Node definition:

```
struct node {
  int val;
  struct node* left;
  struct node* right;
};
```

code/bst.c

Recursive solution:

```
struct node* lca_rec(struct node* root, struct node* n1, struct node* n2) {
  if (root==NULL) return NULL;
  if (root->val > n1->val && root->val > n2->val) {
    return lca_rec(root->left, n1, n2);
  }
  if (root->val < n1->val && root->val < n2->val) {
    return lca_rec(root->right, n1, n2);
  }
  return root;
}
```

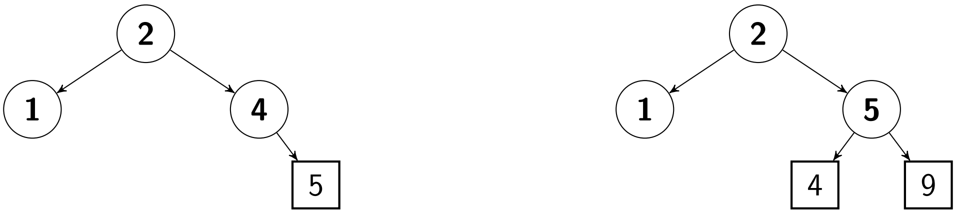
code/bst.c

Iterative solution:

```
struct node* lca(struct node* root, struct node* n1, struct node* n2) {
  while(root!=NULL) {
    if (root->val > n1->val && root->val > n2->val) root=root->left;
    if (root->val < n1->val && root->val < n2->val) root=root->right;
  }
  return root;
}
```

code/bst.c

1.3 [12 points] Consider the red-black tree in Figure 2a where black nodes are denoted with a circle and red nodes are denoted with a square. Table 1 illustrates the cases and actions if value **9** is inserted in the red-black tree of Figure 2a. Figure 2b shows the resulting tree.



(a) Before insertion of 9

(b) After insertion of **9**

Figure 2: Inserting 9 into a red-black tree

Operation	Case	Action	Arguments
Inserting 9	Case 3 mirrored	assign color	4, red
		assign color	5, black
		left rotate	4
Inserting 13	Case 3 mirrored	assign color	10, black
		assign color	9, red
		left rotate	9
Inserting 12	Case 1 mirrored	assign color	10, red
		assign color	9, black
		assign color	13, black
	Case 3 mirrored	assign color	2, red
		assign color	5, black
		left rotate	2

Table 1: Insert cases and actions

Perform the following sequence of insertion operations on the red-black tree in Figure 3: insert 13, insert 12. Each operation is applied to the result of the previous operation. For each operation fill in the missing parts in Table 1 and draw the resulting red-black tree.

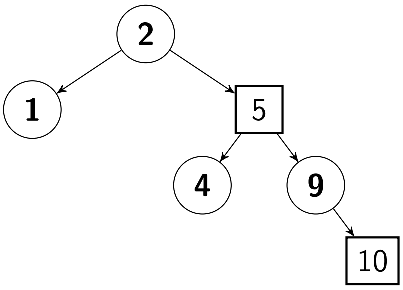
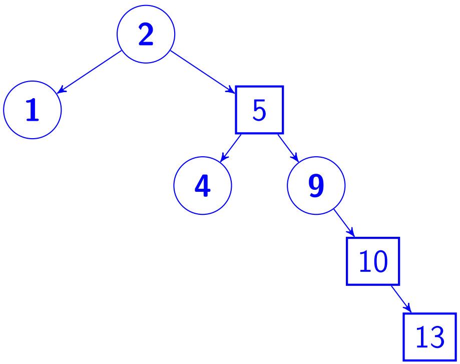
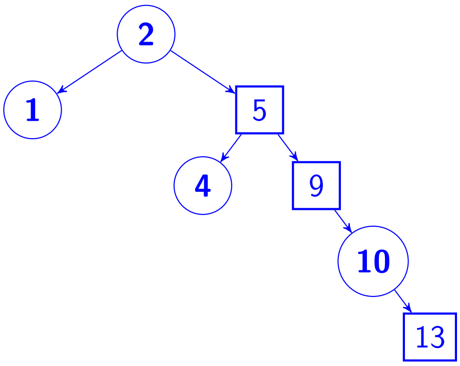
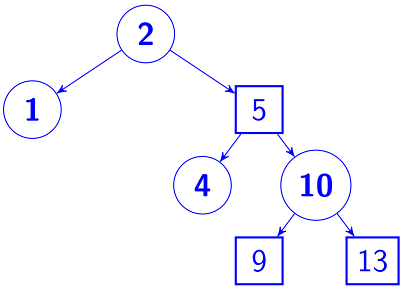


Figure 3: Red-black tree

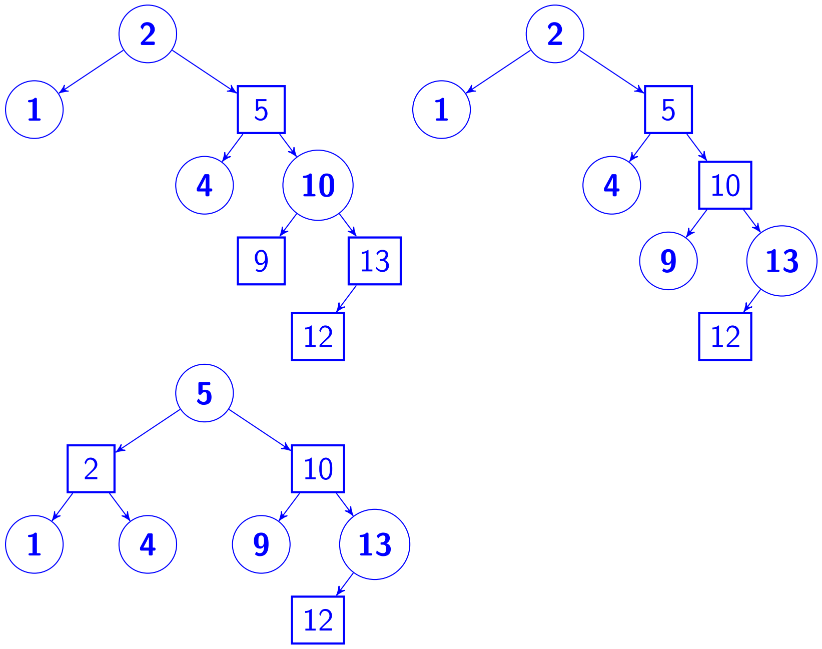
Inserting 13:





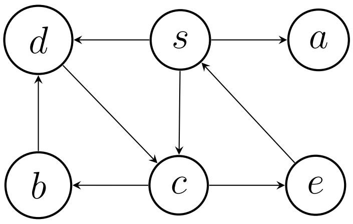


Inserting 12:



Exercise 2 15 Points

2.1 [3 points] Consider the graph in Figure 4 together with its adjacency list representation.



adj(s) = [a,c,d]
adj(a) = [ ]
adj(c) = [e,b]
adj(b) = [d]
adj(d) = [c]
adj(e) = [s]

Figure 4: Graph and its adjacency list

State the order in which the nodes are visited during, respectively, a Breadth First Search (BFS) and Depth First Search (DFS). The search starts at node `s`.

i) **Breadth First Search** (BFS):

Solution: s a c d e b

ii) **Depth First Search** (DFS)

Solution: s a c e b d

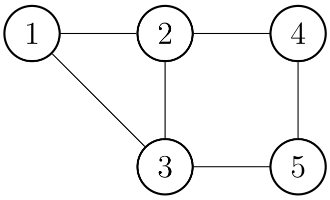
2.2 [3 points] List two data structures that can be used to represent graphs. For each data structure determine the worst time complexity (big O notation) for deciding if node x and node y are connected.

Adjacency List: O(V)
Adjacency Matrix: O(1)

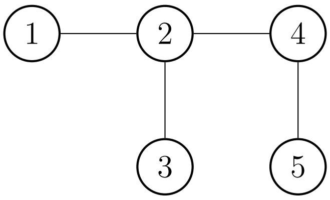
2.3 [9 points] A graph is said to be `Biconnected` if:

(a) It is connected, i.e., it is possible to reach every vertex from every other vertex, by a simple path.

(b) Even after removing any vertex the graph remains connected.



(a) Biconnected graph



(b) Not a biconnected graph

Given an undirected `connected graph G` and a start vertex `s`, write a pseudocode algorithm that determines if the connected graph G is biconnected? In an undirected graph two connected vertices, v_1 and v_2, are represented by two edges as $e_1(v_1, v_2)$ and $e_2(v_2, v_1)$ from v_1 to v_2 and v_2 to v_1, respectively.

```
Algorithm: Init(G)
foreach $v \in G.V$ do
    v.color = W; v.deleted = false;
return isBiConnected(G);
```

```
Algorithm: isBiConnected(G)
foreach $v \in G.V$ do
    v.deleted = true;
    if isConnected(G) == false then
        return false;
return true;
```

```
Algorithm: isConnected(G)
InitQueue(Q);
s = randVertex(G);
Enqueue(Q,s) ;
while $Q \neq \phi$ do
    v = Dequeue(Q) ;
    v.color = G;
    foreach $u \in v.adj$ do
        if $u.color == W$ and $u.deleted == false$ then
            Enqueue(Q,u)
foreach  $v \in G.V$ do
    if $v.color == W$ and $v.deleted == false$ then
        return false;
foreach $v \in G.V$ do
    v.color = W;
    v.deleted =false;
return true;
```

```
Algorithm: randVertex(G)
foreach $v \in G.V$ do
    if $v.deleted == false$ then
        return v;
```

Exercise 3 18 Points

3.1 [6 points] Consider a function $roll(S, n, k)$ that can be applied to a stack S. The roll function rotates the top $n \geq 0$ elements of stack S by $k \geq 0$ positions in a circular fashion. Figure 6 illustrates three examples of the *roll* function.

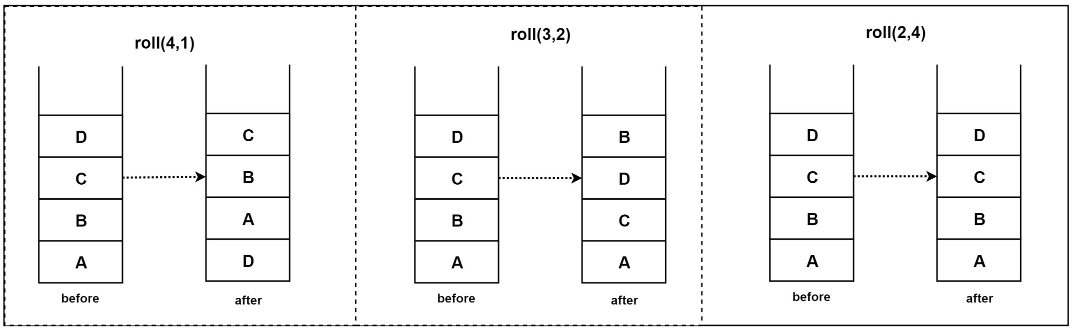


Figure 6: Examples of Roll operator

Design and sketch an algorithm that implements $roll(S, n, k)$. For non-valid inputs the stack must be left unchanged. Note that k can be larger than m, in which case the *roll* operation does more than one complete rotation. The performance of your algorithm must be $O(n)$.

```
Algorithm: roll(Stack s,int n,int k)
if $n < 0$ or $k < 0$ or $n >$ s.size() then return;
Initialize array arr of length $n$;
for i=0 to n-1 do arr[i] = s.pop(); ;
for i=n-1 to 0 do s.push(arr[(i+k) mod n]); ;
```

3.2 [4.5 points] Consider hash table `HT` with 7 slots and hash function $h(k) = k \mod 7$.

Draw the hash table after inserting, in the given order, values $19, 26, 13, 48, 17$.

Assume collisions are handled by

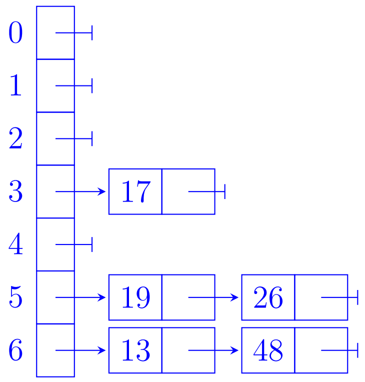
(a) chaining

(b) linear probing

(c) double hashing with secondary hash function $h2(k) = 5 - (k \mod 5)$ for the step size

For each collision resolution scheme show the hash tables after all insertions have been performed.

chaining



linear probing

0	13
1	48
2	
3	17
4	
5	19
6	26

double hashing

0	
1	48
2	26
3	17
4	
5	19
6	13

3.3 [7.5 points] Answer the following:

(a) [1.5 points] Consider an initially empty hash table of size M and hash function `h(x) = x mod M`. In the worst case what is the time-complexity to insert `n` keys into the table if chaining is used to resolve collisions. Assume that overflow chains are implemented as unordered linked lists. Give a brief justification for your answer.

$$\mathbf{O(n)}$$

(b) [1.5 points] What is the answer for question (a) if the overflow lists are ordered? Give a brief justification for your answer.

$$\mathbf{O}(n^2)$$

(c) [1.5 points] Consider the same hash table and function as in task (a), but assume that collisions are resolved using linear probing, and $n \leq \frac{M}{2}$. In the worst case what is the time complexity (in big O notation) to insert n keys into the hash table? Give a brief justification for your answer.

$$\mathbf{O}(n^2)$$

(d) [1.5 points] How big must the hash table be if we have 60000 items in a hash table that uses open addressing (linear probing) and we want a load factor of 0.75?

$$TableSize = \frac{n}{\alpha} = \frac{6000}{0.75} = 80000$$

(e) [1.5 points] What is the expected number of comparisons to search for a key if we must store 60000 items in a hash table that uses open addressing (linear probing) and we have a load factor of 0.75.

$$\frac{(1+\frac{1}{(1-\alpha)})}{2} = \frac{(1+\frac{1}{1-0.75})}{2} = 2.5$$

Exercise 4 15 Points

We are given a rod of metal of length n. We are also given a pricing table of m different cut lengths $l_1, ..., l_m$, and their corresponding prices, $p_1, ..., p_m$. We assume the table is sorted by cut length so that l_1 is the smallest cut length we can sell. We want to cut the rod into different segments to maximize the sum of all segment prices hence maximizing the profit. A piece of length i is worth p_i CHF as listed in Figure 7.

length l_i	3	5	7
price p_i	6	7	10

Figure 7: Prices for rods of different lengths

Thus, if we have a rod of length 10 the most beneficial strategy is to cut the rod into three pieces of length 3, which gives you a benefit of 18 CHF.

4.1 [2 points] Assume $r(n)$ denotes the maximum profit you can get for a rod of length n and price table p. Formulate a recursive definition of $r(n)$.

$$r(n) = \begin{cases} 0 & \text{if } n < l_1 \\ \max_{i=1..m}(p_i + r(n - l_i)) & \text{otherwise} \\ \forall k \text{ such that } l_k \leq n & \end{cases}$$

4.2 [4 points] Assume an array $l[1...m]$ of different cut lengths and an array $p[1...m]$ of corresponding prices. Write a recursive algorithm that computes the maximal profit for cutting a rod of length n into pieces. Formulate a recurrence for the runtime complexity of your algorithm. Determine the asymptotic runtime complexity by solving the recurrence.

```
Algorithm: CutRod(p,n)
if $n == 0$ then
    return 0;
r = - $\infty$ ;
for $i$=1 to $n$ do
    r = max(r,p[i] + CutRod(p,n-i));
return r;
```

4.3 [3 points] In order to efficiently compute the maximal profit by cutting a rod of length n into pieces, a dynamic programming solution with arrays $r[0...n]$ and $c[0...n]$ can be used. Element $r[i]$ in array r contains the maximum profit earned for cutting the rod of length i. Element $c[i]$ in array c contains a cut-length that is part of a solution that yields the maximal profit for a rod of length i. Given the prices for different cut lengths in Table 2, complete Table 3 with the maximal profits and optimal cut lengths.

length l_i	3	5	7
price p_i	6	7	10

Table 2: Prices for rods of different lengths

Fill the table for rod of length 10

length l_i	0	1	2	3	4	5	6	7	8	9	10
r[i]	0	0	0	6	6	7	12	12	13	18	18
c[i]	0	1	1	3	1	5	3	1	3	3	1

Table 3: Maximal profits and optimal cut lengths

4.4 [6 points] Assume an array $l[1...m]$ of different cut lengths and an array $p[1...m]$ of corresponding prices. Describe an algorithm that uses the memoization technique to compute the maximum profit for cutting a rod of length n into different pieces. Also compute an array c[0...n] with the values of the cut lengths that give the optimal value. You can use either C or pseudocode for your solution.

One way we can do this is by writing the recursion as normal, but store the result of the recursive calls, and if we need the result in a future recursive call, we can use the precomputed value. The answer will be stored in r[n].

```
Algorithm: InitArray(p,n)
let r[0..n] be a new array
let c[0..n] be a new array
for $i$=0 to $n$ do
    r[i] = - $\infty$;
for $i$=0 to $n$ do
    c[i] = 0;
return CutRodDP(p,n,r,c);
```

```
Algorithm: CutRodDP(p,n,r,c)
if $r[n] \geq 0$ then
    return r[n];
if $n == 0$ then
    q = 0;
else
    q = - $\infty$;
    for $i$=1 to $n$ do
        val = CutRodDP(p,n-i,r,c));
        if $q < val + p[i]$ then
            q = val + p[i];
            c[n] = i;
    r[n] = q;
return q;
```

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 ☐

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 ☒

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 ☐

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

True ☒ False ☐

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

True ☐ False ☒

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 ☐

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 ☐

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

Grading:

- 1 pt each for correct answer

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

Grading:

- 1 pt for correct answer only. No partial pt.

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?

$\boxed{\frac{k}{x}}$

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

$\boxed{\text{k}}$

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

$\boxed{\text{Yes}}$

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

$\boxed{O(kN)}$

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

$\boxed{O(kN)}$

Grading:

- 0.2 pt each for i - v.
- 1 pt each for vi & vii

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 $H3$?

$\boxed{O(N)}$

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

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

Grading:

- 1 pt each for correct answer

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.

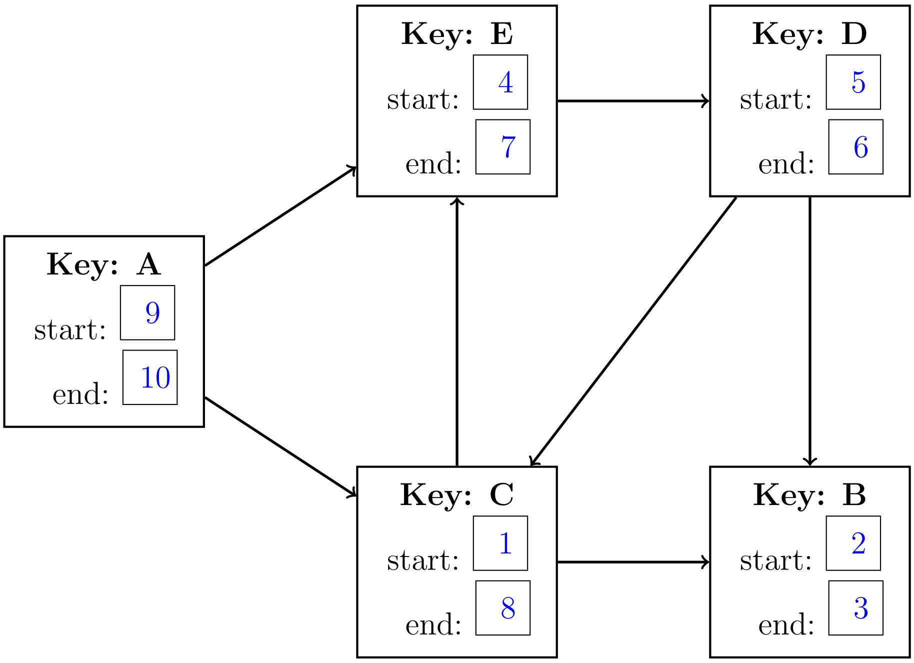


Figure 1: Directed Graph

Grading:

- 0.2 pt each for correct answer

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

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

Grading:

- 1 pt for correct answer. No Partial

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.

```
Algorithm: BFS(G)
s.col = G;
InitQueue(Q);
Enqueue(Q,s);
while $Q \neq \phi$ do
    v = Dequeue(Q);
    foreach $u \in v.adj$ do
        u.degree ++;
        if $u.col == W$ then
            u.col = G; Enqueue(Q,u);
    v.col = B;
```

```
Algorithm: hasEulerPath(G)
foreach $v \in G.V$ do
    v.col = W; v.degree = 0;
s = RandomVertex(G);
G = BFS(G,s);
oddCount = 0;
foreach $v \in G.V$ do
    if $v.degree\ mod\ 2 ==1$ then
        oddCount++;
    if $oddCount > 2\ or\ v.degree == 0$ then
        return False;
return True;
```

Grading:

- 3 pts each for correct and complete code:
  - Connected graph
  - finding graph has no more than two vertices of odd degree.

Exercise 3 3 + 6 Points

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

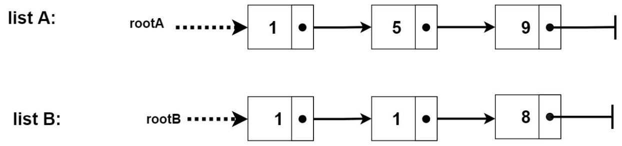


Figure 2: Input linked-lists list A and list B

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

a. The merged list must be sorted as well.

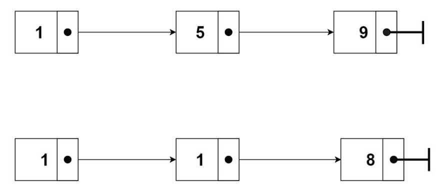
b. You are not allowed to use an auxiliary data structure whose size depends on the number of elements in the original list. The merging shall be implemented by modifying pointers.

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

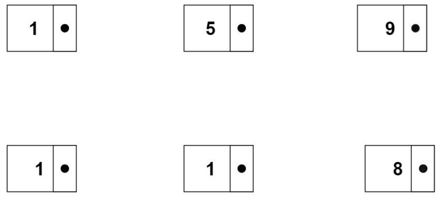
Use the following data structure and variables for your solution.

```
struct node {
  int val;
  struct node* next;
};

struct node* rootA;
struct node* rootB;
```

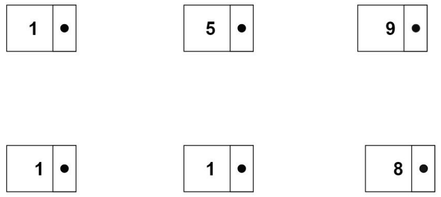
3.1 Illustrate step by step how will you merge the linked lists. Draw all pointers that are used and modified at each step.



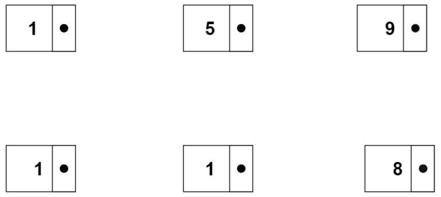
i.



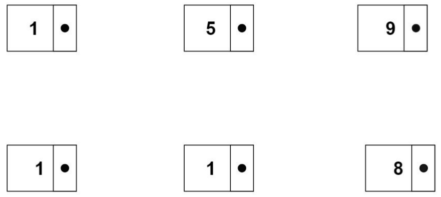
ii.



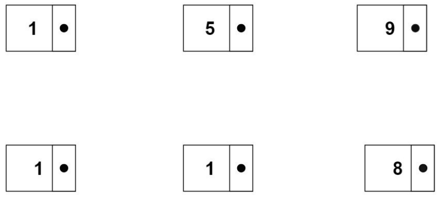
iii.



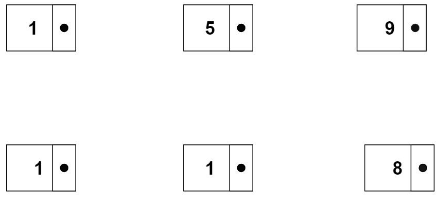
iv.



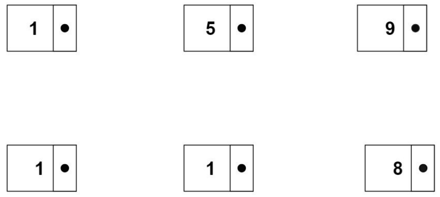
v.



vi.

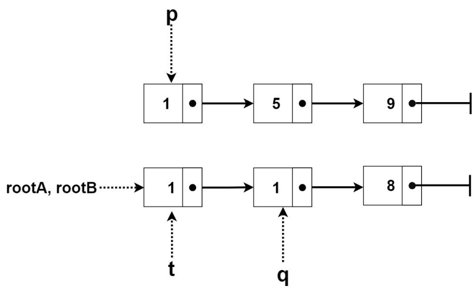


vii.

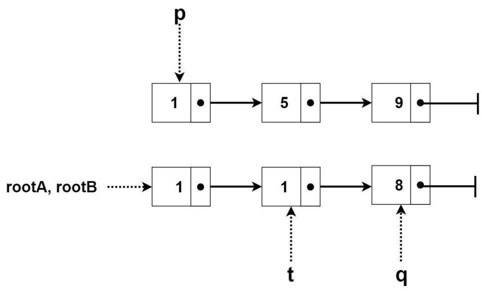


viii.

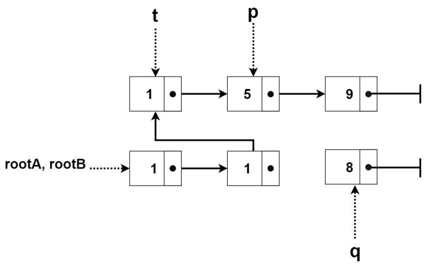
start of loop; 1st iteration



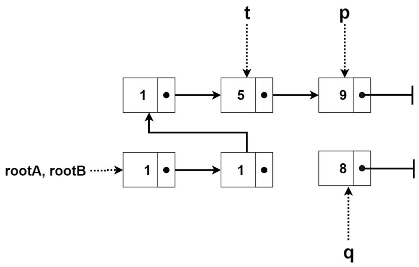
start of loop; 2nd iteration



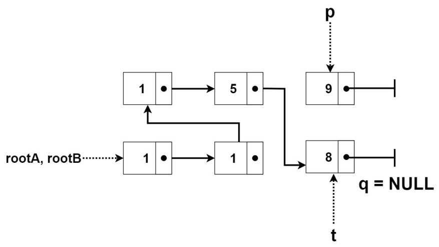
start of loop; 3rd iteration



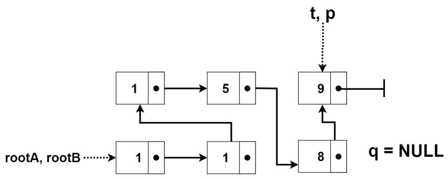
start of loop; 4th iteration



start of loop; 5th iteration



After Loop; If condition



Grading:

- 0.5 pt for labelling rootA,rootB at same node.
- 1 pt for drawing correct arrows with no step missing.
- 1.5 pt for labelling auxilary pointers with no step missing.

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)

Grading:

- 1 pt for correct call to function.
- 0.5 pt for rootA and rootB both NULL condition. [Line 2-3]
- 0.5 pt for rootA NULL but rootB not NULL condition.[Line 4-5]
- 0.5 pt for rootA not NULL but rootB NULL condition. [Line 6-7]
- 0.5 pt for pointing rootA and rootB to same node. [8-11]
- 2 pt for complete and correct merge loop until one list finishes. [Line 12-19]

- 1 pt for cases when one list terminates. [Line 20-23]
- -1 pt for not mentioning * with root and rootB

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

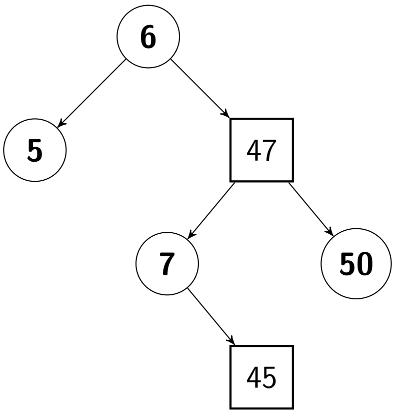
c. What is the height of a `d-ary` heap with n elements?

$\theta(\log_d n)$

Grading:

- 2 pt each for correct answer.

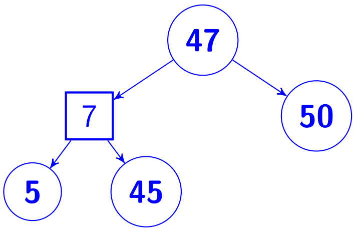
4.2 Consider following red-black tree where black nodes are denoted with a circle and red nodes are denoted with a square. Delete **6** from the tree and draw the resulting red-black tree.



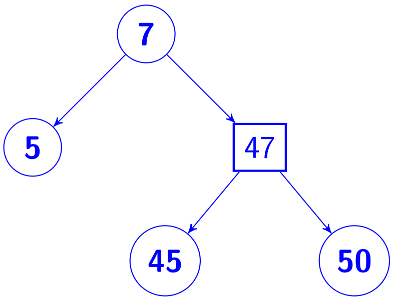
Grading:

- 2 pt each for either Solution A or Solution B.

Solution A:



Solution B:



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.

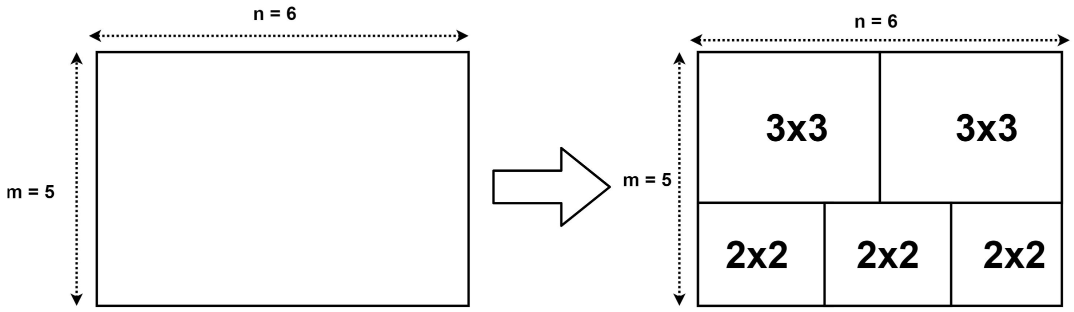


Figure 3: Minimum number of square tiles for a rectangle of size 5x6

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 $MinSquare(n, m)$.

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

Grading:

- 0.5 pt for correct n==m condition
- 0.5 pt for outer min.
- 0.5 pt each for nested mins.

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.

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

Grading:

- 0.25pt each for correct value

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];
```

Grading:

- 1 pt each for base condition n==m.[Line 4-6]
- 1 pt for dp[n][m] check and return condition.[Line 7-8]
- 1 pt for loop of squares in vertical direction.[Line 9-10]
- 1 pt for loop of squares in horizontal direction. [Line 11-12]
- 1pt for finding minimum squares in horizontal in vertical direction. [Line 13]

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 2020 | 27.05.2020

Name: ______________________ Matriculation number: ____________

Address: ______________________ Date of Birth: ____________

Advice

You have 90 minutes to complete and submit the Final exam of Informatics II. This is an open book exam.

Submit your solution in one of the following ways:

1. You can print the pdf file, use the available whitespace to fill in your solution, scan your solution, and upload the pdf file to OLAT.
2. You can use blank white paper for your solutions, scan the sheets, and upload the pdf file to OLAT. Put your name and matriculation number on every sheet. State all task numbers clearly.
3. You can use a tablet and pen (iPad, Surface, etc) to fill in your solution directly into the pdf file and upload the completed pdf file to OLAT.
4. You can use a text editor to answer the questions and submit the document as pdf.

Notes:

- If you do not have scanner it is possible to take pictures of your solution with your phone. We recommend Microsoft Office Lens or Camscanner. Create a pdf file that includes all pictures and submit a single pdf file.
- There is no extra time for scanning and submission. The allotted time already includes the time for scanning and submission.
- Only submissions through OLAT are accepted. Submissions through email are only considered if OLAT is not working.

Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	Total
Points Achieved					
Maximum Points	17	25	21	22	85

Exercise 1 — 17 Points

Complete the boxes by checking the correct checkbox or by filling your answer into the empty box.

1.1 [1 point] Assume chaining is used to resolve collisions for a hash table of size m that stores N elements with unique keys. The worst-case asymptotic time complexity to remove an item from the hash table is $O(N)$.

Answer: ☒ True ☐ False

1.2 [1 point] Assume $f(n) = O(N^2)$ and $g(n) = O(N^3)$. From these complexities it follows that $f(n) + g(n) = O(N^3)$.

Answer: ☒ True ☐ False

1.3 [1 point] The asymptotic time complexity to search an element in a binary search tree with N nodes is $O(\log N)$.

Answer: ☐ True ☒ False

1.4 [1 point] In a max-heap the depth of any two leaves differs by at most 1.

Answer: ☒ True ☐ False

1.5 [1 point] A hash table guarantees a constant lookup time.

Answer: ☐ True ☒ False

1.6 [4 points] Assume a hash table $T[0...9]$ with open addressing is used to store integer keys. Draw the hash table after the keys 18, 34, 88, 66, 70, 80, and 76 have been inserted (in that order). Illustrate all details of the hash table. Assume that conflicts are resolved with double hashing defined as follows:

$$h(k,i) = (h_1(k) + i * h_2(k))\ mod\ 10$$
$$h_1(k) = (k\ mod\ 10)$$
$$h_2(k) = \lfloor k/10 \rfloor$$

Slot	0	1	2	3	4	5	6	7	8	9
Value	70		66	76	34		88		18	
Status	OCC	EMP	OCC	OCC	OCC	EMP	OCC	EMP	OCC	EMP

1.7 Consider the red-black tree in Figure 1a where black nodes are denoted with a circle and red nodes are denoted with a square. We consider four operations:

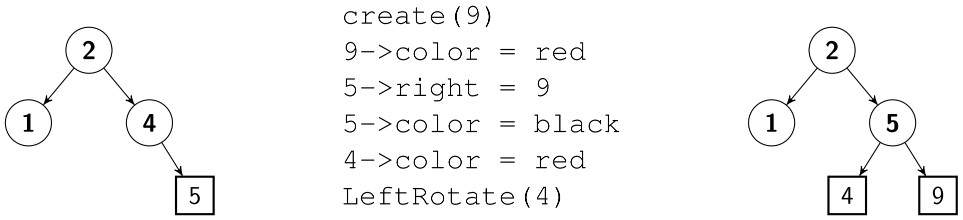
(a) Creating a new node (left and right are set to NULL):
Example: create node 9: `create(9)`

(b) Setting a property (color, key, left, right) of a node:
Example: set the key of node 9 to 5: `9->key = 5`

(c) Rotating a node:
Example: right rotate node 5: `RightRotate(5)`

(d) Deleting a node:
Example: delete node 9: `delete(9)`

The result of applying the operations in Figure 1b to the red-black tree in Figure 1a yields the red-black tree in Figure 1c.



```
create(9)
9->color = red
5->right = 9
5->color = black
4->color = red
LeftRotate(4)
```

(a) Before operations (b) Operations (c) After operations

Figure 1: Tree before and after operations

Use the example notation illustrated in Figure 1b to work out your solutions for the following tasks and the red-black tree in Figure 2.

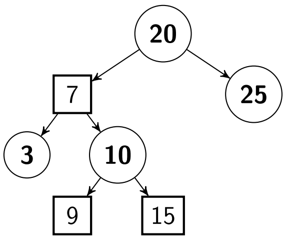
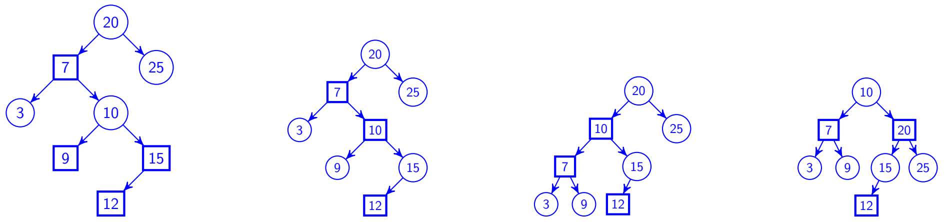


Figure 2: Red-black tree

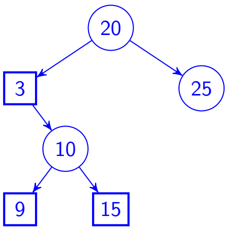
(a) [5 points] State the operations that are required to insert **12** into the red-black tree in Figure 2

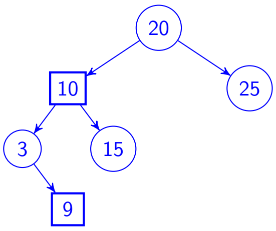
```
create(12)           9->color = black     LeftRotate(7)     10->color = black
15->left = 12        15->color = black                      20->color = red
12->color = red      10->color = red                        RightRotate(20)
```



(b) [3 points] State the operations that are required to delete **7** from the red-black tree in Figure 2.
Note: If you have the option to choose a node from either the left or right subtree choose the node from the left subtree.

```
delete(3)          10->color = red
7->key = 3         3->color = black
                   15->color = black
                   LeftRotate(3)
```





Exercise 2 25 Points

Consider the C code fragment in Figure 3 that defines a structure for a linked list. Each node has a field n that points to the next node in the list, a field p that points to the previous node in the list, and a field k with the key of the node. Variable h points to the head (first node) of the list and variable t points to the tail (last node) of the list. On the right side in Figure 3 an example list is shown.

```
struct node {
    int k;
    struct node* n;
    struct node* p;
};

struct node *h, *t;
struct node *a, *b;
```

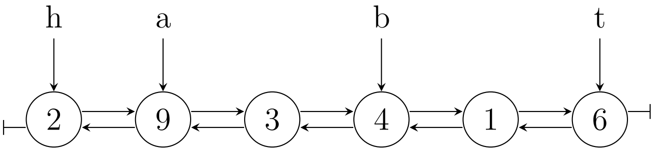


Figure 3: Linked list in C

The task is to develop a C procedure

```
void xchg(struct node* a, struct node* b, ...) {
```

with arguments a and b that point to distinct elements of the list. You may assume that a points to a node that in the list comes before the node b points to. The result of xchg is a list that is equal to the original list except that elements a and b have been exchanged. The solution must be implemented by modifying pointers. From within procedure xchg no global variable may be accessed.

1. [5 points] State the full signature of xchg by completing the header of the above C procedure, i.e., specify all arguments that are required to correctly exchange the elements pointed to by a and b. Show how procedure xchg must be called. Assume h, t, a, and b have been initialized appropriately.

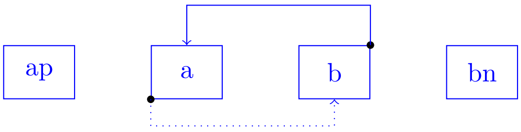
```
void xchg(struct node* a, struct node* b,
          struct node** h, struct node** t) { ... }
```

call: `xchg(a, b, &h, &t);`

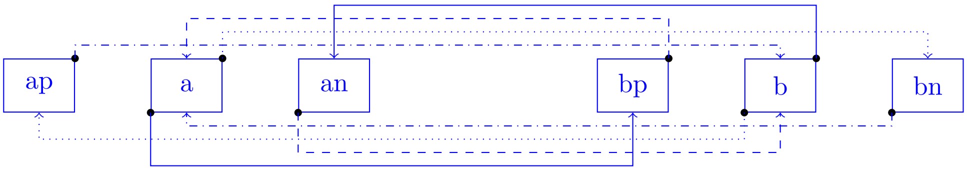
Name: Matriculation number:

2. [4 points] Sketch a picture that illustrates how the pointers must be changed.

a and *b* are adjacent:



a and *b* are not adjacent:



3. [16 points] State the C code for procedure `xchg`.

```c
void xchg(struct node* a, struct node* b,
          struct node** h,struct node** t){

  struct node* ap = a->p;
  struct node* an = a->n;
  struct node* bp = b->p;
  struct node* bn = b->n;

  if (an == b) {
    b->n = a;
    a->p = b;
  } else {
    an->p = b;
    bp->n = a;
    b->n = an;
    a->p = bp;
  }
  if (ap != NULL) {ap->n = b;} else {*h = b;}
  if (bn != NULL) {bn->p = a;} else {*t = a;}
  b->p = ap;
  a->n = bn;
}
```

code/linkedlist.c

Exercise 3 21 Points

Consider a directed graph $G = (V, E)$ with nodes $v \in V$ and $u \in V$. Node u is a **k-hop neighbor** of node v if there is at least one path from v to u and the minimum distance from v to u is k. The distance from v to u is infinite if there is no path from v to u.

Let $|V|$ be the number of nodes in G. Each node is identified by a unique integer between 1 and $|V|$. A directed edge $(v, u) \in E$ is represented as a pair of nodes and denotes an edge from node $v \in V$ to node $u \in V$.

3.1 [2 points] Assume a graph $G = (V, E)$ with vertices $V = \{1, 2, 3, 4, 5, 6\}$ and edges $E = \{(1, 2), (1, 4), (1, 5), (2, 3), (2, 5), (3, 1), (4, 6)\}$. Draw the adjacency matrix of G.

Adjacency Matrix:

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	1	0	1	0
3	1	0	0	0	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	0	0

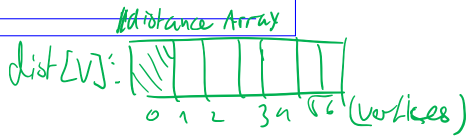
3.2 [4 points] Let $k = 2$. Determine all 2-hop neighbors of node 1? Explain your approach.

- The 2-hop neighbors of node 1 are nodes 3 and 6.
- Run a BFS search until first node in the queue has a distance of k. Dequeue and report nodes as long as they have a distance equal to k.

3.3 [15 points] Assume a directed graph G with n nodes. An adjacency matrix $a[n, n]$ is used to represent the edges of the graph. Give a pseudocode algorithm that prints all k-hop neighbors of node v.

Pseudocode:

```
Algorithm: khop(G, v, k)
for ($i = 1; i <= n; i++$) do dist[i] = -1;
dist[v] = 0;
InitQueue(Q);
while ($v \neq -1$ && $dist[v] < k$) do
    for ($i = 1; i \leq n; i++$) do
        if ($a[v, i] == 1$ && $dist[i] == -1$) then
            dist[i] = dist[v] +1;
            Enqueue(Q,i);
    v = Dequeue(Q);
while $v \neq -1$ && $dist[v] == k$ do
    print(v);
    v = Dequeue(Q);
```



C code:

```
void khop(int n, int v, int k) {
  for (int i = 0; i < n; i++) { dist[i] = -1; }//dist of each vertex set to -1.
  h = 0; t = 0; dist[v] = 0;
  while (v != -1 && dist[v] < k) {
    for (int i = 0; i < n; i++){
      if (a[v][i] == 1 && dist[i] == -1) {//a[v][i]==1 ->there is an adj. vertex.
        dist[i] = dist[v] + 1;     //
        enqueue(i);
      }
    }
    v = dequeue();
  }
  while (v != -1 && dist[v] == k) {
    printf("%d ", v);
    v = dequeue();
  }
  printf("\n");
}
```

code/khop.c

Pseudocode:

```
Algorithm: khop(G, v, k)
for $(i = 1; i <= n; i++)$ do dist[i] = -1;
dist[v] = 0;
InitQueue(Q);
while $(v \neq -1$ && $dist[v] < k)$ do
    for $(i = 1; i \leq n; i++)$ do
        if $(a[v, i] == 1$ && $dist[i] == -1)$ then
            dist[i] = dist[v] +1;
            Enqueue(Q,i);
    v = Dequeue(Q);
while $v \neq -1$ && $dist[v] == k$ do
    print(v);
    v = Dequeue(Q);
```

distance

1	2	3	4	5	6
-1	-1	-1	-1	-1	-1

	1	2	3	4	5	6
1	0	1	0	1	1	0
2	0	0	1	0	1	0
3	1	0	0	0	0	0
4	0	0	0	0	0	1
5	0	0	0	0	0	0
6	0	0	0	0	0	0

queue.

Exercise 4 — 22 Points

Consider a matrix M with integer-valued elements. The task is to compute the length of the side of the largest square that only includes negative numbers. The idea for the solution is to compute an auxiliary matrix S with the same dimensions as matrix M. The value of element $S[i, j]$ is the length of the largest square whose lower right corner is located at position (i, j) in M. Figure 4 shows an example with a 3×3 matrix M_0 and the corresponding solution matrix S_0.

$$\begin{array}{c|c|c|c|}
 & 0 & 1 & 2 \\ \hline
0 & 1 & -1 & 6 \\ \hline
1 & -3 & -2 & -1 \\ \hline
2 & -5 & -1 & 1 \\ \hline
\end{array}
\qquad
\begin{array}{c|c|c|c|}
 & 0 & 1 & 2 \\ \hline
0 & 0 & 1 & 0 \\ \hline
1 & 1 & 1 & 1 \\ \hline
2 & 1 & 2 & 0 \\ \hline
\end{array}$$

Matrix M_0 — Solution S_0 for M_0

Figure 4: Illustration of input matrix M and solution matrix S

Note that in order to compute the largest square with the lower right corner at position (i, j) it is sufficient to consider the solutions for the squares with lower right corners at positions $(i - 1, j)$, $(i, j - 1)$ and $(i - 1, j - 1)$.

4.1 [4 points] Consider Figure 5 with the 4×4 matrix M. Complete the corresponding solution matrix S.

$$\begin{array}{|c|c|c|c|}
\hline
-1 & 0 & -1 & 0 \\ \hline
-1 & 0 & -1 & -1 \\ \hline
-1 & -1 & -1 & 0 \\ \hline
-1 & -1 & -1 & 0 \\ \hline
\end{array}
\qquad
\begin{array}{|c|c|c|c|}
\hline
1 & 0 & 1 & 0 \\ \hline
1 & 0 & 1 & 1 \\ \hline
1 & 1 & 1 & 0 \\ \hline
1 & 2 & 2 & 0 \\ \hline
\end{array}$$

Matrix M — Solution S for M

Figure 5: Input matrix M and solution matrix S

4.2 [6 points] Provide a recursive problem formulation for determining the side length of the largest square in matrix M that includes only negative integers.

$$S[i,j] = \begin{cases} 0 & \text{M[i,j]} \geqslant 0 \\ min(min(S[i-1,j-1], S[i-1,j]), S[i,j-1]) + 1 & M[i,j] < 0 \end{cases}$$

4.3 [12 points] Write a C function `int maxLen(int m)` that computes and returns the length of the largest square in an $m \times m$ matrix M that includes negative integers only.

```c
int maxLen(int m) {
  int maxSize = 0;
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < m; j++) {
      if (i == 0 || j == 0) {
        if (M[i][j] < 0) {
          S[i][j] = 1;
          maxSize = 1;
        }
        else { S[i][j] = 0; }
      } else {
        if (M[i][j] < 0) {
          S[i][j] = 1 + min( min(S[i-1][j-1], S[i][j-1]),
                             S[i-1][j] );
          maxSize = max(maxSize, S[i][j]);
        }
      }
    }
  }
  return maxSize;
}
```

code/dp2.c

DEPARTMENT OF INFORMATICS

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

University of Zurich^UZH

Informatics II — Final Exam

Spring 2021 — 02.06.2021

Name: ______________________ Matriculation number: ____________

Address: ______________________ Date of Birth: ____________

About Exam

You have 90 minutes to answer the questions; you have 20 minutes to download the exam questions and submmit your solutions through EPIS. Only submissions through EPIS are accepted, and only PDF files are accepted.

Submit your solution in one of the following ways:

1. You can print the pdf file, use the available whitespace to fill in your solution, scan your solution, and upload the pdf file to EPIS.
2. You can use blank white paper for your solutions, scan the sheets, and upload the pdf file to EPIS. Put your name and matriculation number on every sheet. State all task numbers clearly.
3. You can use a tablet and pen (iPad, Surface, etc) to fill in your solution directly into the pdf file and upload the completed pdf file to EPIS.
4. You can use a text editor to answer the questions and submit the document as pdf.

Notes:

- If you do not have scanner it is possible to take pictures of your solution with your phone. We recommend Microsoft Office Lens or Camscanner. Create a pdf file that includes all pictures and submit a single pdf file.
- The 20 minutes include downloading exams, preparing your solutions for your submission, and submitting the PDF files through EPIS.
- We suggest that you submit your solutions several minutes earlier than the deadline.
- You bear the risk for your last-minute submission.

Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	Total
Points Achieved					
Maximum Points	11	12	20	22	65

Exercise 1 11 Points

Complete the boxes by checking the correct checkbox or by filling your answer into the empty box.

1.1 [1 point] The worst-case asymptotic time complexity to search an integer in a binary search tree of N integers is O(N).

Answer: ☒ True ☐ False

1.2 [1 point] The worst-case asymptotic time complexity to search an integer in a red-black tree of N integers is O(log N).

Answer: ☒ True ☐ False

1.3 [1 point] The worst-case asymptotic time complexity to search an integer in a max-heap of N integers is O(log N).

Answer: ☐ True ☒ False

1.4 [1 point] Assume $f_1(n) = O(1)$, $f_2(n) = O(N^2)$, and $f_3(n) = O(N \log N)$. From these complexities it follows that $f_1(n) + f_2(n) + f_3(n) = O(N \log N)$.

Answer: ☐ True ☒ False

1.5 [1 point] The master method applies to calculate the asymptotic time complexity of binary search.

Answer: ☒ True ☐ False

1.6 [6 points] Consider the red-black tree in Figure 1a where black nodes are denoted with a circle and red nodes are denoted with a square. We use the key to represent a node. For example, 2 represents the node with key 2. We consider four operations:

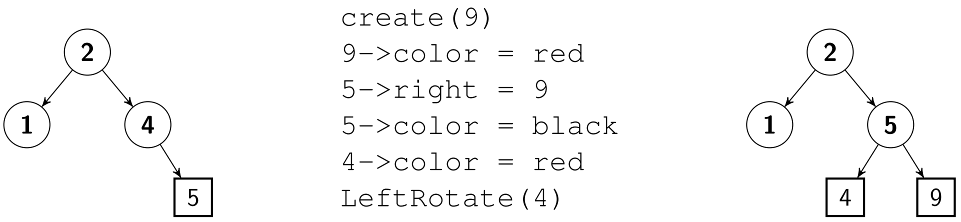
(a) Creating a new node (left and right are set to NULL):
Example: create node 9: `create(9)`

(b) Setting a property (color, key, left, right) of a node:
Example: set the key of node 9 to 5: `9->key = 5`. Here, 9 represents the node with key 9.

(c) Rotating a node:
Example: right rotate node 5: `RightRotate(5)`

(d) Deleting a node:
Example: delete node 9: `delete(9)`

The result of applying the operations in Figure 1b to the red-black tree in Figure 1a yields the red-black tree in Figure 1c.



```
create(9)
9->color = red
5->right = 9
5->color = black
4->color = red
LeftRotate(4)
```

(a) Before operations (b) Operations (c) After operations

Figure 1: Tree before and after operations

Use the example notation illustrated in Figure 1b to work out your solutions for the following tasks and the red-black tree in Figure 2.

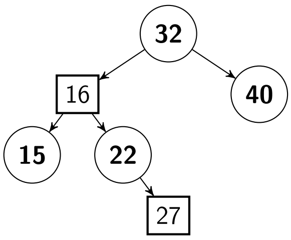
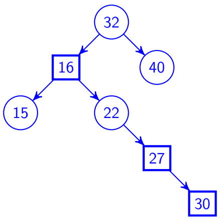
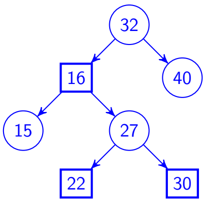


Figure 2: Red-black tree

(a) [3 points] State the operations that are required to insert **30** into the red-black tree in Figure 2.

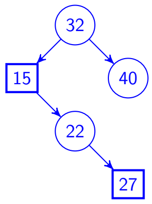
```
create(30)          LeftRotate(22)
27->right = 30      27->color = black
30->color = red     22->color = red
```



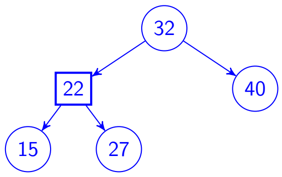


(b) [3 points] State the operations that are required to delete **16** from the red-black tree in Figure 2.
Note: If you have the option to choose a node from either the left or right subtree choose the node from the left subtree.

```
delete(15)
16->key = 15
```



```
LeftRotate(15)
22->color = red
15->color = black
27->color = black
```



Exercise 2 — 12 Points

Consider a hash table `HT` consisting of m slots, using open addressing **with linear probing and a step of size 1**. Write a C code for the function `int HTDelete(int k)`, which deletes key k from the hash table and also rearranges the other elements. The rearrangement leaves the table exactly as it would have been had key k never been inserted i.e., you cannot simply set the status as deleted.

The Hash table `HT` is defined as follows:

```c
struct elem{
  int key;
  int status;   // 0:OCCUPIED, -1: EMPTY
};

 struct elem HT[m];
```

The following example shows the state of a hash table after calling `HTDelete(11)`. The hash table uses linear probing with step of size 1 and a hash function $h(k) = k\%m$, with m=5.

Slot	Status	Key
0	-1	-1
1	0	11
2	0	22
3	0	31
4	0	2

$\xrightarrow{\text{After Deleting 11}}$

Slot	Status	Key
0	-1	-1
1	0	31
2	0	22
3	0	2
4	-1	-1

Table 1: Illustration of function call `HTDelete(11)` on the hash table.

You can use the following helper functions:

1. `int hash(int k,int i)` - return the hash slot for key k and step of size i.
2. `int HTInsert(int k)` - inserts key k in the hash table HT and return the slot number. Returns -1 if hash table is full.

Note: Initially all slots are empty and keys are initialized with -1. The function `HTDelete(int k)` returns the slot number of a deleted key k and -1 if key k does not exist in the hash table.

```c
int HTDelete(int k){
  int i=0;
  int probe = hash(k,i);
  int actualHashIdx= probe;
  while(i<m && HT[probe].status == 0 && HT[probe].key!=k){
    i++;
    probe = hash(k,i);
  }
  if(i>=m || HT[probe].status==-1) return -1;
  HT[probe].status = -1;
  HT[probe].key = -1;
  int j=(probe+1)%m;
  while(j!=actualHashIdx && HT[j].status!=-1){
    if(hash(HT[j].key,0)!=j){
      int tmpKey = HT[j].key;
      HT[j].key = -1;
      HT[j].status = -1;
      HTInsert(tmpKey);
    }
    j = (j+1)%m;
  }
  return probe;
}
```

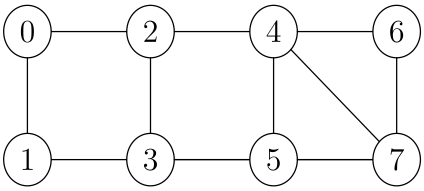
code/Hash.c

Exercise 3 20 Points

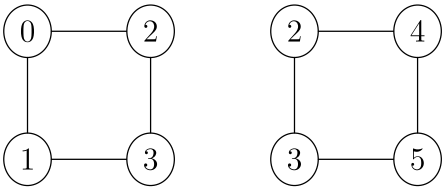
Consider an undirected graph $G = (V, E)$ with vertices V and edges E. A BiQuad subgraph $B = (X, Y)$ is an induced subgraph of a graph G formed from a subset of the vertices of the graph G and all of the edges connecting pairs of vertices in that subset. The vertices X is a subset of vertices V, that is $X \subseteq V$, and edges Y is a subset of edges E, that is $Y \subseteq E$, such that the following conditions hold.

1. A BiQuad subgraph only consists of four vertices i.e. $|X| = 4$, connected to each other.
2. Each vertex is connected to exactly two vertices in the BiQuad graph.

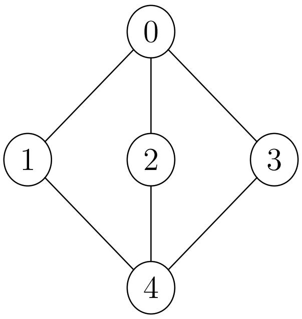
Let $|V|$ be the number of nodes in G. Each node is identified by a unique integer between 0 and $|V| - 1$. Consider the following graph G.

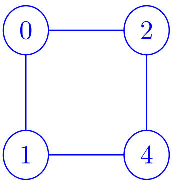


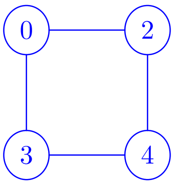
The graph G contains two BiQuad Subgraphs:

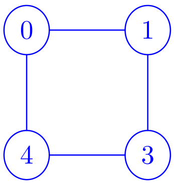


3.1 [3 points] State the BiQuad Subgraphs of the following graph G.









3.2 [17 points] Assume an undirected graph G with n vertices. An adjacency matrix $graph[n,n]$ is used to represent the edges of the graph. Give a C code that determines and prints all BiQuad subgraphs of graph G.

```
void BiQuad(int graph[N][N],int s) {
 int i=0,u=0,k=0,q=0, neighbors[N],tmpN=-1;
 for (i = 0; i < N; i++) { dist[i] =
   -1;color[i]=0;pred[i]=-1;neighbors[i]=-1;}
 dist[s] = 0; color[s] = 1;
 enqueue(s);
 while(!isEmpty()){
  int v = dequeue();
  int count = 0,tmpN=-1;
 for(u=0;u<N;u++){
   if(graph[v][u]==1){
    if(color[u] == 0){
     color[u]=1; dist[u] = dist[v]+1; pred[u]=v;
     enqueue(u);
   }else if(color[u]==2 && dist[u]<=dist[v]){
    neighbors[u]=1;
    count=count + 1;
    tmpN=u;
   }
  }
 }
 if(count>1){
  for(u=0;u<N;u++){
   for(q=u+1;q<N;q++){
    if(neighbors[u]==1 && neighbors[q]==1){
     int fourthVertex = -1, distDiff = dist[q]-dist[u];
     if(distDiff==0 || distDiff==1 || distDiff==-1){
      if(graph[u][pred[q]]==1){
       fourthVertex = pred[q];
      }else if(graph[q][pred[u]]==1){
       fourthVertex = pred[u];
      }
     }
     if(fourthVertex!=-1){
      printf("BiQuad Subgraph: %d, %d, %d, %d
        \n",fourthVertex,q,v,u);
     }
     if(distDiff == 0){
      for(k=0;k<N;k++){
       if(k!=pred[q] && color[k]==2 && graph[q][k]==1 &&
         graph[u][k]==1){
        printf("BiQuad Subgraph: %d, %d, %d, %d
          \n",k,q,v,u);
       }
      }
     }
    }
   }
   neighbors[u]=-1;
  }
}else if(count == 1){
   neighbors[tmpN] = -1;
}
color[v]=2;
}
```

code/Graph.c

Exercise 4 22 Points

Consider a matrix M with integer-valued elements. The task is to compute the number of zeros in a rectangle of M. The rectangle is defined by the upper left coordinate $(0, 0)$ and lower right coordinate (i, j). Both coordinates are **included** in the rectangle. The idea for the solution is to compute an auxiliary matrix S with the same dimensions as matrix M. The value of element $S[i, j]$ is the number of zeros in the rectangle whose upper left is $(0, 0)$ and lower right corner is (i, j) in M.

	0	1	2
0	0	-2	0
1	0	0	-9
2	25	0	1

Matrix M_0

	0	1	2
0	1	1	2
1	2	3	4
2	2	4	5

Solution S_0 for M_0

Figure 3: Illustration of input matrix M and solution matrix S

4.1 [3 points] Consider Figure 4 with the 3×4 matrix M. Complete the corresponding solution matrix S.

12	0	4	-5
11	9	0	3
0	-4	0	8

Matrix M

0	1	1	1
0	1	2	2
1	2	4	4

Solution S for M

Figure 4: Input matrix M and solution matrix S

4.2 [14 points] Write a C function `void computeS(int m, int n)` that computes the matrix S for an $m \times n$ matrix M. Both M and S have been initialized, and you can directly use them in your solution.

```c
void computeS(int m, int n) {
  if(M[0][0] == 0) {
    S[0][0] = 1;
  }
  for (int i = 1; i < n; i++) {
    if(M[0][i] == 0) {
      S[0][i] = S[0][i - 1] + 1;
    } else{
       S[0][i] = S[0][i - 1];
    }
  }
  for (int i = 1; i < m; i++) {
    if (M[i][0] == 0) {
      S[i][0] = S[i - 1][0] + 1;
    } else {
       S[i][0] = S[i - 1][0];
    }
  }
  for (int i = 1; i < m; i++) {
    for (int j = 1; j < n; j++) {
      if (M[i][j] == 0) {
        S[i][j] = S[i - 1][j] + S[i][j - 1] - S[i - 1][j - 1] +
           1;
      } else {
        S[i][j] = S[i - 1][j] + S[i][j - 1] - S[i - 1][j - 1];
      }
    }
  }
}
```

code/dp.c

4.3 [5 points] Consider that the upper left coordinate is an arbitrary coordinate (a, b). Provide a recursive problem formulation for determining $n(a, b, i, j)$ that is the number of zeros contained in the rectangle whose upper left coordinate is (a, b) and lower right coordinate is (i, j) using S.

$$n(a,b,i,j) = \begin{cases} S[i,j] - S[i][b-1] & \texttt{a = 0 and b > 0} \\ S[i,j] - S[a-1,j] & \texttt{a > 0 and b = 0} \\ S[i,j] - S[a-1,j] - S[i][b-1] + S[a-1][b-1] & \texttt{a > 0 and b > 0} \end{cases}$$