Faculty of Business, Economics and Informatics (abbreviated as WWF) Examination and Assessment Honor Code

The WWF community (faculty, students, and alumni) share a commitment to honesty and integrity and in particular follows the standards of scholarship and professionalism in all examinations and assessments.

Students agree to comply by the WWF Examination and Assessment Honor Code.

Students who violate the WWF Examination and Assessment Honor Code are in breach of this agreement and must accept the sanctions imposed by the WWF, which include official WWF or UZH disciplinary actions.

1. Each member of the WWF community has a personal obligation to report known violations to the Dean's Office.
2. No student shall represent another's work as his or her own.
3. No person shall receive inadmissible assistance of any sort, or provide inadmissible assistance to another student, at any time before, during, or after an examination or in relation to other graded course work.
4. Each student has to confirm the following commitment before starting an exam: "I confirm hereby that I do not violate the WWF Examination and Assessment Honor Code during this examination."
5. The principles embodied in this WWF Examination and Assessment Honor Code apply to every one of the WWF community.
6. Violations of the WWF Examination and Assessment Honor Code that relate to academic issues such as, for example, academic misconduct, will be handled according to the WWF and UZH disciplinary procedures.
7. Purposefully misleading the WWF Examination and Assessment Honor Code judicial process is a violation of the WWF Examination and Assessment Honor Code.

I, ..., confirm hereby that I have read and do not violate the WWF Examination and Assessment Honor Code during this examination:

....

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 — Midterm 2
Spring 2020 — 27.04.2020

Name: ______________________ Matriculation number: ____________

Advice

You have 70 minutes to complete and submit the midterm 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. 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.
- Sign and submit the "Honor Code" as well. Without "Honor Code" the exam cannot be accepted.
- Multiple submissions are allowed. Only your last submission is considered for the correction.
- Only submissions through OLAT will be accepted. Submissions through email will only be considered if OLAT is not working.

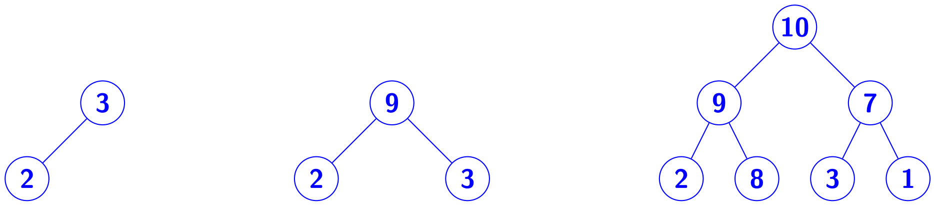
Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	Total
Points Achieved					
Maximum Points	18	15	26	11	70

Exercise 1 15 Points

1.1 [3 points] The values **3 2 ; 9 ; 10 8 7 1 ;** are inserted in the given order into the same `max-heap`. The `max-heap` is initially empty. Draw the `max-heap` at the positions marked by a semicolon.



1.2 [3 points] Consider an algorithm that determines the smallest element in a `max-heap` without accessing all elements of the heap. Assume the `max-heap` contains n elements and is represented with an array. How many elements must the algorithm consider to determine the smallest element in the heap?

The algorithm must consider all leaf nodes of the heap. There are $\lceil n/2 \rceil$ leaves in a max-heap with n elements.

1.3 Consider algorithm `Algo1` below. Input array `A` represents a `min-heap` that contains n distinct integers. $k \leq n$ is a positive integer.

```
Algorithm: Algo1(A,n,k)
num = -1
s = n
for $(i = n; i > n - k; i - -)$ do
    num = A[1]
    exchange A[i] and A[1]
    s = s - 1;
    Heapify(A, 1, s)
return num;
```

(a) [3 points] Apply algorithm `Algo1` to input array `A = [1, 3, 6, 4, 5, 8, 9]`. Complete the table for $k = 3$. The first line of the table shows the initial state of array `A`. The other lines shall show num and array `A` at the end of the for loop.

num	**A[1]**	**A[2]**	**A[3]**	**A[4]**	**A[5]**	**A[6]**	**A[7]**
-	1	3	6	4	5	8	9
1	3	4	6	9	5	8	1
3	4	5	6	9	8	3	1
4	5	8	6	9	4	3	1

(b) [2 points] What does algorithm `Algo1` return?

Algo1 returns the kth-smallest element of `A`.

(c) [3 points] What is the asymptotic complexity of algorithm `Algo1`? Explain.

- The for loop is executed k times
- Only the heapify statement in the for loops depends on n. All other statements have a constant time complexity. The runtime of heapify is $\Theta(log(n))$
- The total asymptotic runtime is therefore $\Theta(k * log(n))$.

(d) [4 points] In algorithm `Algo1`, assume input array `A` contains duplicates. State **precisely** what the algorithm returns in this case.

It returns an element x of array A such that there are

- at most $k - 1$ elements in A that are larger than x and
- at least $k - 1$ elements in A larger or equal than x.

Exercise 2

2.1 [6 points] The key element of the quicksort algorithm is its partitioning procedure. Complete the boxes below to complete algorithm Partition(A, l, r) that partitions array $A[l..r]$.

```
Algorithm: Partition(A,l,r)
$x$ = A[l];
i = l;
for $j$ = $\boxed{l + 1}$ to $\boxed{r}$ do
    if $A[j]$ $\boxed{\leq}$ $x$ then
        i = $\boxed{i + 1}$ ;
        exchange A[i] and A[j];
exchange $\boxed{A[i]}$ and $\boxed{A[l]}$ ;
return $\boxed{i}$ ;
```

2.2 Consider a binary tree defined as follows:

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

struct node* root;
```

(a) [7 points] Implement a C function `int smallest(...)` that returns the smallest value in a non-empty binary tree. Show an example of how the function is called. Pseudocode is not accepted.

finding and returning the smallest element in a binary tree:

```
int smallest(struct node* p, int v) {
    if (p == NULL) { return v; }
    return min(smallest(p->left,v),
               min(smallest(p->right,v), p->value));
}
```

calling the function:

```
printf("Smallest elem = %d\n", smallest(root,root->value));
```

(b) [2 points] Assume a binary tree with n nodes. What is the asymptotic complexity of your algorithm? Explain.

To find the smallest value every node must be visited once. Hence the asymptotic complexity is $O(N)$.

Exercise 3

3.1 Consider the following table that shows the memory location and content of four variables. a and b are integers; x and y are pointers.

Variable	**Address**	**Content**
a	62fe10	5
b	62fe20	10
x	62fe30	62fe40
y	62fe40	62fe10

Table 1: Memory Layout

(a) [2 points] Declare variables **x** and **y** in C.

```
int** x;



int* y;
```

(b) [1 point]

Assume $a = 5$ and $b = 10$ and the variable locations shown in Table 1. Give the statements that were used to assign to x and y the values shown in Table 1.

```
x = &y;
y = &a;
```

(c) [1 point]

```
printf("%d", *y+10);
```

Output: 15

(d) [2 points]

```
**x = 30;
printf("%d",*y-5+b);
```

Output: 35

3.2 [2 points] Consider a queue that is implemented with a circular array $Q[0..n-1]$ of length n. Variables h and t point to, respectively, the head and tail of the queue. Which of the statements below correctly check for an overflow of the queue in the first line of the `enqueue` algorithm illustrated below.

```
Algorithm: enqueue(x)
if ... then
    printf("Overflow");
else
    Q[t] = x;
    t = t+1 mod n;
```

(a) `t+1 == n`

(b) `t == n`

(c) `t+1 == h`

(d) `t == h`

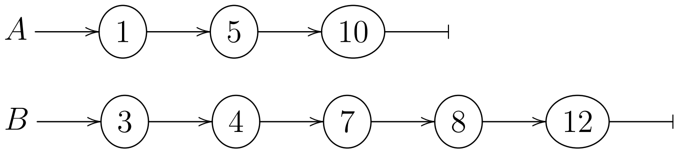
(e) `(h+1) mod n == t`

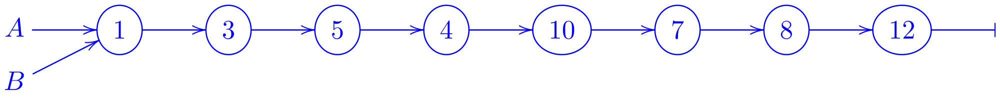
(f) `(t+1) mod n == h`

Answer: (d)

3.3 Consider two linked lists A and B as illustrated below. Function `merge(...)` shall be used to merge the nodes of A and B into a single linked list. It does so by taking the nodes alternatively from the two lists starting with list A. If all nodes of a list have been used, all subsequent nodes are taken from the other list. At the end of the merging A and B shall point point to the same list.

(a) [3 points] Consider the following lists A and B. Show the result of the merging described above.





(b) [3 points] Consider the following data structure and variable definitions for the two linked lists.

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

struct node* A;
struct node* B;
```

Explain why a merge function with signature `void merge(struct node* rA, struct node* rB)` cannot be used to implement the above described merging of two lists.

A functions cannot change the arguments passed to function (arguments are passed by value).

Consider the call merge(A,B). After the call A and B are the same.

(c) [3 points] Give a function signature that allows to implement the described merging of two lists. Show how the function must be called to merge lists A and B as defined above.

void merge(struct node** rA, struct node** rB)

merge(&A, &B);

(d) [9 points] Implement function `merge` in **C**.

Note:

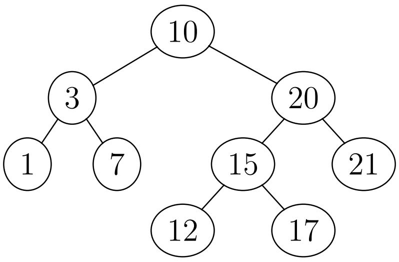
- *You may not use an auxiliary linked list or any other data structures.*
- *The merging should be implemented by modifying pointers.*
- *After merging, A and B must point to the head of the merged linked list.*

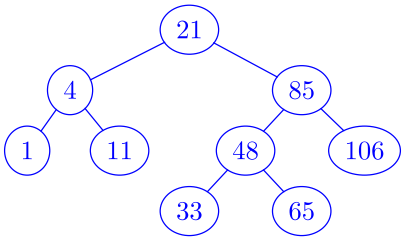
```c
void merge(struct node** A, struct node** B){
    if (*A == NULL && *B == NULL) return;
    else if (*A == NULL) { *A = *B; return; }
    else if (*B == NULL ) { *B = *A; return; }
    struct node* curr = *A;
    struct node* next = *B;
    while (next != NULL) {
        struct node* temp = next;
        next = curr->next;
        curr->next = temp;
        curr = temp;
    }
    *B = *A;
}
```

Exercise 4

Consider a binary search tree. Function `MinAggregate` modifies the tree such that each node contains the aggregated value of all nodes (i.e., summation of values) less than or equal to that node. The modified tree is the *Minimum Aggregate Tree* for the binary search tree.

4.1 [4 points] What is the *Minimum Aggregate Tree* for the following binary search tree?





4.2 [7 points] Implement a C function `MinAggregate` that transforms a binary search tree into its *Minimum Aggregate Tree*.

Note:

- You are only allowed to modify the values in the binary search tree. No additional data structure is allowed.
- The asymptotic complexity of your solution must be $O(n)$ for a tree with n nodes.

```
int MinAggregate(struct TreeNode* node, int sum){
    if (node == NULL) { return sum; }
    sum = node->val + MinAggregate(node->left,sum);
    node->val = sum;
    sum = MinAggregate(node->right,sum);
    return sum;
}
```