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 1
Spring 2020 — 23.03.2020

Name: ______________________ Matriculation number: ____________

Advice

You have 70 minutes to complete and submit the midterm exam of Informatik II. The following rules apply: Submit it 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 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. Use a tablet and pen (iPad, Surface, etc) to fill in your solution directly into the pdf file, upload the completed pdf file to OLAT.
4. You can use text editor to answer the questions and then 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" exam will not be accepted.
- Multiple submissions are allowed. Only your last submission is considered for 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	5	Total
Points Achieved						
Maximum Points	23	13	16	9	14	75

Name: Matriculation number:

Exercise 1

1.1 [10 points] Assume a non-negative decimal number is represented as an array of single decimal digits with the least significant digit at the end. For example, 123 is represented with array `A` where `A[0] = 1`, `A[1] = 2`, and `A[2] = 3`. Consider two non-negative integers that are represented with arrays `A` and `B`. The arrays have the same length n. Implement a C function `int sum(int A[], int B[], int n)` that computes and returns the sum of `A` and `B`. Give your solution in C; pseudocode is not accepted.

1st solution (without auxiliary array):

```c
int sum(int a[], int b[], int n) {
  int s = 0;
  int o = 0;
  for (int i = n - 1; i >= 0; i--) {
    s = s + ((a[i] + b[i] + o) % 10) * (int)pow(10,n-1-i);
    o = (a[i] + b[i] + o) / 10 ;
  }
  return s + o * (int)pow(10,n);
}
```

code/task1b.c

2nd solution (with auxiliary array):

```c
int sum(int a[], int b[], int n) {
  int c[n + 1];
  for (int i = n - 1; i >= 0; i--) {
    c[i] = (c[i + 1] + a[i] + b[i]) / 10;
    c[i+1] = (c[i+1] + a[i] + b[i]) % 10;
  }
  int sum = 0;
  for (int i = 0 ; i <= n; i++) {
    sum = 10 * sum + c[i];
  }
  return sum;
}
```

code/task1.c

1.2 [13 points] Consider algorithm `Algo1` shown below. Input array `A` contains n **distinct** integers in the range from 0 to $n - 1$.

```
Algorithm: Algo1(A,n)
index = 0;
while $index \leqslant n - 1$ do
    while $index \neq A[index]$ do
        temp = A[index];
        A[index] = A[temp];
        A[temp] = temp;
    index = index + 1;
```

(a) [5 points] Apply the algorithm on array `A = [4, 6, 5, 1, 3, 2, 0]`. Complete the table below to show step-by-step how `A` is modified. The first line of the table shows the initial state of array `A`.

index	**A[0]**	**A[1]**	**A[2]**	**A[3]**	**A[4]**	**A[5]**	**A[6]**
-	4	6	5	1	3	2	0
0	3	6	5	1	4	2	0
0	1	6	5	3	4	2	0
0	6	1	5	3	4	2	0
0	0	1	5	3	4	2	6
2	0	1	2	3	4	5	6

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

Moves each value to the position (index) that is equal to the value.
Result is a sorted array.

(c) [2 points] What is the asymptotic complexity of algorithm `Algo1` in the worst case? Explain.

$\Theta(n)$

Each element is considered once and directly moved to its final position.

(d) [2 points] What is the asymptotic complexity of algorithm `Algo1` in the best case? Explain

$\Theta(n)$

In the best case lines 4-6 are not executed; in the worst case lines 4-6 are executed.

(e) [2 points] Precisely quantify the costs of the computations that are done in the worst case but not in the best case.

$3 * N$ movements of integers
$4 * N$ accesses of array elements
N access to an integer variable

Exercise 2

2.1 [7 points] Consider a non-empty array `A[0..n-1]` with n integer elements that are all different from each other. The array consists of an ascending part followed by a descending part. Thus, there exist an i, $0 \leq i < n-1$ such that `A[0]` $<$ `A[1]` $<$ ... $<$ `A[i]` $>$ `A[i + 1]` $>$ `A[i + 2]` $>$ ... $>$ `A[n - 1]`. Given an array `A` and its length n, implement a C function `search(int A[], int n)` that finds the largest number in array `A`. The time complexity of your solution must be $O(log(n))$. Give your solution in C; pseudocode is not accepted.

Example :
```
Input:  A = [1,2,4,5,7,9,6,3], n = 8
Output:  9
```

```c
int search2(int a[], int n) {
  int l = 0;
  int r = n - 1;
  while (l < r) {
    int m = (l + r) / 2;
    if (l == m && a[m] < a[m+1]) { l = r; }
    else if (l == m) { r = l; }
    else if (a[m-1] > a[m]) { r = m; }
    else { l = m; }
  }
  return a[l];
}
```

code/task2.c

2.2 [6 points] Perform special case analysis for task 2.1.

1. $l < m < r$, up, up: [1,2,3]; (l,r) = (0,2), (1,2), (2,2)
2. $l < m < r$, down, down: [3,2,1]; (l,r) = (0,2), (0,1), (0,0)
3. $l < m < r$, up, down: [3,5,1]; (l,r) = (0,2), (1,2), (1,1)
4. $l = m < r$, up: [3,4]; (l,r) = (0,1), (1,1)
5. $l = m < r$, down: [7,4]; (l,r) = (0,1), (0,0)
6. 1 element: [7]; (l,r) = (0,0)

Exercise 3

3.1 [6 points] Indicate, for each pair of functions ***(A,B)***, in the table below, whether the statement in the top row is correct. Check all cells for which the statement is correct (an unmarked cell gives no points; a correctly marked cell gives positive points; a wrongly marked cell gives negative points). Assume constants $k \geq 1$, $\epsilon > 0$ and $c > 1$.

A	**B**	$\mathbf{B} \in O(\mathbf{A})$	$\mathbf{B} \in \Omega(\mathbf{A})$	$\mathbf{B} \in \Theta(\mathbf{A})$
$(\log n)^k$	n^ϵ		yes	
n^k	c^n		yes	
$\sqrt{n}$	$n^{\sin n}$			
2^n	$2^{n/2}$	yes		
$n^{\log c}$	$c^{\log n}$	yes	yes	yes

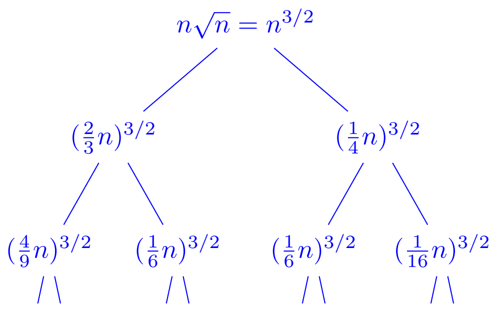
3.2 [10 points] Determine the asymptotic tight bound of the following recurrences using the indicated method.

(a) [2 points] Solve $T(n) = 3T(\frac{n}{2}) + n/2$ with the master theorem method. Specify a, b, $f(n)$ and the correct master theorem case.

* $a = 3,\ b = 2,\ f(n) = n/2$
* Case 1: $\Theta(n^{1.5})$

Name: Matriculation number:

(b) [4 points] Solve $T(n) = T(\frac{2n}{3}) + T(\frac{n}{4}) + n\sqrt{n})$ with the recursion tree method.



$= n^{3/2}$

$= (\frac{11}{12}n)^{3/2}$

$= (\frac{121}{144}n)^{3/2}$

$= ((\frac{11}{12})^i n)^{3/2}$

$= \Theta(n\sqrt{n})$

$$n\sqrt{n}\sum_{i=0}^{h}(\frac{11}{12}^i)^{3/2} \leq \sum_{i=0}^{\infty}(\frac{11}{12}^i)^{3/2} = n\sqrt{n}(\frac{1}{1-11/12})^{3/2} = n\sqrt{n}(12)^{3/2} = 12n\sqrt{12n}$$

(c) [4 points] Prove the correctness of the estimate computed in task (b) with the substitution method.

Proof by induction:

$$\begin{aligned}
T(n) &= T(2n/3) + T(n/4) + n\sqrt{n} && \text{(recurrence)}\\
&\leq c(\tfrac{2n}{3})^{3/2} + c(\tfrac{n}{4})^{3/2} + n^{3/2} && \text{(inductive hyp.)}\\
&= c(\tfrac{11n}{12})^{3/2} + n^{3/2} && \text{(rearranging)}\\
&= n^{3/2}(c(\tfrac{11}{12})^{3/2} + 1) && \text{(rearranging)}\\
&\leq cn^{3/2} && \text{(for } c \geq 12\sqrt{12}, n > 1\text{)}
\end{aligned}$$

Exercise 4

Given an array A[1...N] of N elements, the following algorithm fragment is a **bi-directional bubble sort** algorithm that operates in both directions.

```
for $(i = 1; i <= \lceil (N+1)/2 \rceil; i++)$ do
    for $(j = i; j <= N - i; j++)$ do
        if $(A[j] > A[j+1])$ then
            exchange A[j] and A[j+1];
    for $(k = N - i; k > i; k--)$ do
        if $(A[k] < A[k-1])$ then
            exchange A[k] and A[k-1];
```

4.1 [3 points] Formulate the loop invariant for outer loop.

The elements in $A[1...i]$ and $A[N-i+1...N]$ are sorted

$A[1...i]$ contains the i smallest elements of A;

$A[N-i+1...N]$ contains i biggest elements of A.

$\forall j \in [1...i] : A[j] \leq A[j+1]$

$\forall k \in [N-i+1...N] : A[k] \leq A[k+1]$

$\forall j \in [1...i], k \in [N-i+1...N], \forall l \in [i+1...N-i] : A[j] \leq A[l] \leq A[k]$

4.2 [6 points] Formulate the loop invariant for the inner for-loop on line 2.

$A[i...N-i-1], A[N-i], A[N-i+1...N]$

$A[N-i] \geq A[j] \ for \ i \leq j < N-i$

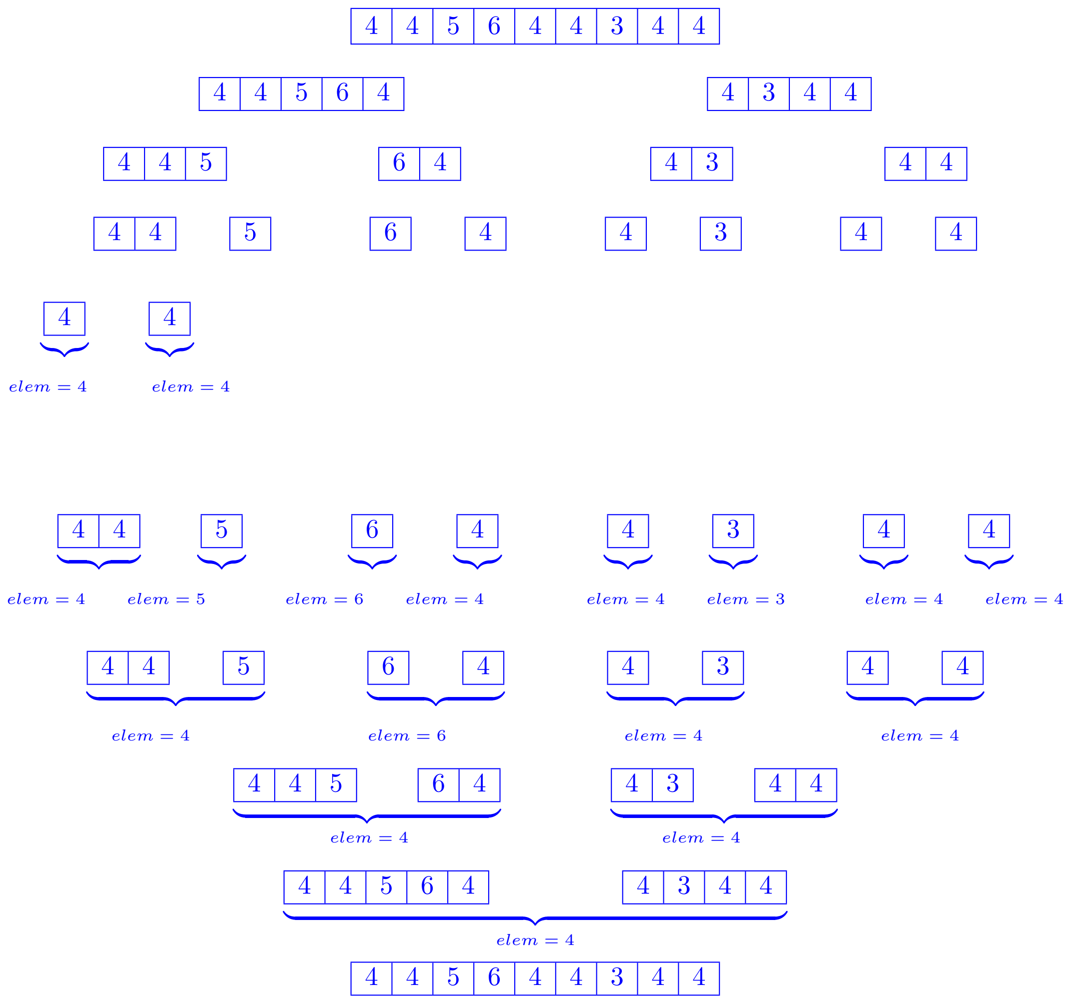
$A[N-i] \circ A[N-i+1...N] \quad sorted$

Exercise 5

Assume an array A[1...N] of N elements. A `half element` occurs more than $\lceil \frac{N}{2} \rceil$ times in array `A`. Use a divide and conquer approach to find a half element. The divide and conquer approach, firstly divides the input array into two partitions: $(A[1] \ldots A[mid])$ and $(A[mid + 1] \ldots A[N])$. Afterwards, it recursively determines the half elements of both partitions. The left and right partitions are combined as follows.

- If both partitions have the same `half element`, this means they agree on the `half element`, and hence the `half element` of the combined partition should also be this half element.
- If the partitions do not have the same `half element`, then we count the occurrences of the `half elements` in the combined partition to determine the `half element`.

5.1 [5 points] Draw a tree to illustrate the process of finding `half element` of array `A = [4, 4, 5, 6, 4, 4, 3, 4, 4]` according to your divide and conquer algorithm.


4 4 5 6 4 4 3 4 4
4 4 5 6 4
4 3 4 4
4 4 5
6 4
4 3
4 4
4 4
5
6
4
4
3
4
4
4
4
elem = 4
elem = 4
4 4
5
6
4
4
3
4
4
elem = 4
elem = 5
elem = 6
elem = 4
elem = 4
elem = 3
elem = 4
elem = 4
4 4
5
6
4
4
3
4
4
elem = 4
elem = 6
elem = 4
elem = 4
4 4 5
6 4
4 3
4 4
elem = 4
elem = 4
4 4 5 6 4
4 3 4 4
elem = 4
4 4 5 6 4 4 3 4 4

5.2 [7 points] Implement a **divide and conquer** algorithm that takes an array A and returns its `half element`. Use C code for your solution.

```c
int countElem(int A[], int num, int l, int r) {
  int count = 0;
  int i = 0;
  for (i=l; i<=r; i++) {
    if (A[i] == num) { count++; }
  }
  return count;
}

int halfElement(int A[], int l, int r) {
  if (l == r) { return A[l]; }
  int mid = (l+r)/2;
  int left = halfElement(A, l, mid);
  int right = halfElement(A, mid+1, r);
  if (left == right) { return left; }
  int leftCount = countElem(A, left, l, r);
  int rightCount = countElem(A, right, l, r);
  if (leftCount>=rightCount) { return left; }
  return right;
}
```

code/task5.c

5.3 [2 points] State the recurrence for the complexity of your algorithm in Task 5.2.

2T(n/2) + 2n