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 — Midterm1
Spring 2014 — 28.03.2014

Name: _____ Matriculation number: _____

Advice

Important: Only the German version is legally binding.

If you attend the module *Informatik IIb* only, you have 90 minutes to complete the exam. If you attend both modules, Informatik IIa and IIb, you have 120 minutes overall to complete both parts of the exam.

The following rules apply for the written part of the exam:

- 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 (8 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.
- For the exam Informatik IIb, only the following items are allowed:
  - One A4 sheet (2-sided) with your personal handwritten notes, written by yourself. Sheets that do not conform to this specification will be collected.
  - For students with a different first language than German: A foreign dictionary is allowed. The dictionary will be checked by a supervisor.
  - No additional items are allowed, notably 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	18	20	20	22	80

Exercise 1 — 18 Points

Arrays

1.1 [8 points] Consider an array `A[0...n_A-1]` with n_A integers. Array A is sorted in increasing order. Give a function `printCounts(A, n_A)` that prints all pairs of the form `(v,t)`, where `v` corresponds to a value in the array and `t` to the number of times it appears. Use either C or pseudocode for your solution.

For example, if `A = [1,2,2,2,2,3,3,4,4,4,5,6]`, your solution should print `(1,1) (2,4) (3,2) (4,3) (5,1) (6,1)`

Algorithm: printCounts(A, n_A)

```
1 count = 1; i = 0;
2 for i=0 to size-1 do
3     if i<n_A-1 ∧ A[i]=A[i+1] then
4         count++;
5     else
6         printf("(%d,%d) ", A[i], count);
7         count = 1;
```

1.2 [10 points] Consider an array `A[0...n_A-1]` with n_A integers, and an array `B[0...n_B-1]` with n_B integers. Arrays A and B are both sorted in increasing order. Create a function `EqualAsSets(A, n_A, B, n_B)` that returns `TRUE` if array A and array B are equal when compared as sets. Use either C or pseudocode for your solution.

For example, if `A = [1,2,2,2,2,3,3,4,4,4,5,6]` and `B = [2,2,3,3,3,4]`, your solution should return `FALSE` since {`1,2,3,4,5,6`} and {`2,3,4`} are not identical.

Algorithm: EqualAsSets(A, sizeA, B, sizeB)

```
1 i = 0; j = 0;
2 while TRUE do
3     if i≥sizeA ∨ j≥sizeB ∨ A[i]≠B[j] then return i≥sizeA ∧ j≥sizeB;
4     repeat i++ until i<sizeA ∧ A[i]==A[i-1];
5     repeat j++ until j<sizeB ∧ B[i]==B[i-1];
```

Name: Matriculation number:

Exercise 2 20 Points

Asymptotic Complexity and Recurrences

2.1 [8 points] Calculate the asymptotic tight bound for the following functions and rank them by their order of growth (lowest first). Clearly work out the calculation steps in your solution: $f_1(n) = (12n)^2$, $f_2(n) = 2^{\log^4(n)}$, $f_3(n) = 10^{732}$, $f_4(n) = log(\pi n) + \log(42^{\log n})$, $f_5(n) = 5^{\lg 7}n^3 + 10^{732}n^2 + \log(3)n^4 + 1321$, $f_6(n) = n \cdot max(\log n, \sqrt{n})$

- $f_1(n) = (12n)^2 \in \Theta(n^2)$
- $f_2(n) = 2^{\log^4 n} = 2^{\log n \log^3 n} = (2^{\log n})^{\log^3 n} = n^{\log^3 n} \in \Theta(n^{\log^3 n})$
- $f_3(n) = 10^{732} \in \Theta(1)$
- $f_4(n) = \log(\pi n) + \log(42^{\log n}) = \log \pi + \log n + \log 42 \log n \in \Theta(log(n))$
- $f_5(n) = 5^{\lg 7}n^3 + 10^{732}n^2 + n\log(n) + 1321 \in \Theta(n\log(n))$
- $f_6(n) = n \cdot max(\log n, \sqrt{n}) = max(n \cdot \log n, n \cdot \sqrt{n}) \in O(n\sqrt{n})$

Solution: $f_3, f_4, f_5, f_6, f_1, f_2$

2.2 [6 points] Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and which case (1-3) applies.

a) $T(n) = 3T(\frac{n}{2}) + n^2$
$a = 3$, $b = 2$, $f(n) = n^2$, Case 3: $\Theta(n^2)$

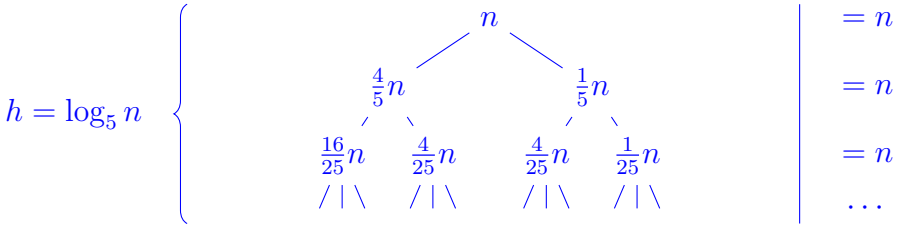
b) $T(n) = \log n + T(\sqrt{n})$
$T(n) = \log n + T(\sqrt{n}) = \log n + \log \sqrt{n} + T(\sqrt{\sqrt{n}})$
$= \log n + \log \sqrt{n} + \log \sqrt{\sqrt{n}} + T(\sqrt{\sqrt{\sqrt{n}}}) + ... =$
$= \log n + \frac{1}{2}\log n + \frac{1}{4}\log n + \frac{1}{8}\log n + ... =$
$= (1 + \frac{1}{2} + \frac{1}{4} +) \cdot \log n =$
$= \sum_{i=0}^{+\infty}(\frac{1}{2})^i \cdot \log n =$
$= 2 \cdot \log n = \Theta(\log n)$

c) $T(n) = 4T(\frac{n}{9}) + 7\sqrt{n}$
$a = 4$, $b = 9$, $f(n) = 7\sqrt{n}$, Case 1: $\Theta(n^{\log_9 4})$

2.3 [6 points] Consider the following recurrence:

$$T(n) = \begin{cases} 1 & \text{, if } n = 1 \\ T(n) = T(4n/5) + T(n/5) + n & \text{, if } n > 1 \end{cases}$$

a) Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Include the tree-based calculations that led to your estimate.



Longest branch on the right. Tree grows until $\left(\frac{1}{5}\right)^h n = 1 \implies h = \log_5 n$

Guess: $O(n \lg n)$

b) Prove the correctness of your estimate using induction.

Proof by induction:

$$\begin{aligned} T(n) &= T(4n/5) + T(n/5) + n && \text{(recurrence)} \\ &\leq c\tfrac{4n}{5} \lg \tfrac{4n}{5} + c\tfrac{n}{5} \lg \tfrac{n}{5} + n && \text{(inductive hyp.)} \\ &= cn \lg n + n + \tfrac{4cn}{5} \lg \tfrac{4}{5} + \tfrac{cn}{5} lg \tfrac{1}{5} && \text{(reformating)} \\ &= cn \lg n + n \left(1 + \tfrac{c}{5}(4 \lg(4) - 5 \lg(5)\right) && \text{(reformating)} \\ &\leq cn \lg n && \text{(for } c \geq \tfrac{5}{5 \lg 5 - 4 \lg 4}, n > 1) \end{aligned}$$

Name: Matriculation number:

Exercise 3 20 Points

Runtime and Recursion

```
Algorithm: whatDoesItDo(A,n)
1 for i = n to 2 do
2     k = i;
3     for j = 1 to i-1 do
4         if A[j]>A[k] then
5             k = j;
6     exchange A[i] and A[k];
```

3.1 [4 points] The algorithm `whatDoesItDo(A,n)` gets as inputs an array `A[1...n]` and the number `n` of integers it contains. Assume A=`[7, 2, 9, 1, 4]`. Complete the following matrix where you need to show the value of i and the content of A after each execution of the outer for loop. The first line of the matrix shows the initial array A before the execution of the loop.

i	**A[1]**	**A[2]**	**A[3]**	**A[4]**	**A[5]**
-	7	2	9	1	4
5	7	2	9	1	4
4	7	2	4	1	9
3	1	2	4	7	9
2	1	2	4	7	9

3.2 [2 points] Describe the best and worst case for this algorithm.

Best case: α always 0, array already sorted in ascending order.
Worst case: α always 1, array sorted in descending order.

3.3 [8 points] Analyze the steps of the algorithm `whatDoesItDo(A,n)` and calculate its exact runtime.

line 1	c_1	n
line 2	c_2	$n-1$
line 3	c_3	$\frac{n(n-1)}{2}^{*} + 2(n-1)^{**}$
line 4	c_4	$\frac{n(n-1)}{2}$
line 5	c_5	$\alpha^{***}\frac{n(n-1)}{2}$
line 6	c_6	$n-1$

$* \sum_{j=1}^{n-1} j = \frac{(n-1)(n-1+1)}{2} = \frac{n(n-1)}{2}$

** $n-1$ times for $i-1$ and $n-1$ times for termination condition

*** $0 \leq \alpha \leq 1$

Running time:

$c_1 n + c_2(n-1) + c_3\left(\frac{n(n-1)}{2} + 2(n-1)\right) + c_4\left(\frac{n(n-1)}{2}\right) + c_5\left(\alpha\frac{n(n-1)}{2}\right) + c_6(n-1)$

3.4 [6 points] Describe in words (maximum three lines) what the algorithm `whatDoesItDo(A,n)` does. Give a recursive version of the algorithm `whatDoesItDo(A,n)`. Use either C or pseudocode for your solution.

The algorithm sorts the array. It's a variant of selection sort that in each iteration places the largest element at the end of the remaining array (in each iteration one less element of the array is considered).

```
Algorithm: selectionSortRec(A,n)
1 if n==1 then return;
2 k = n-1;
3 for j = 0 to n-2 do
4     if A[j]>A[k] then k = j;
5 exchange A[k] and A[n-1];
6 selectionSortRec(A,n-1);
```

Name: Matriculation number:

Exercise 4 22 Points

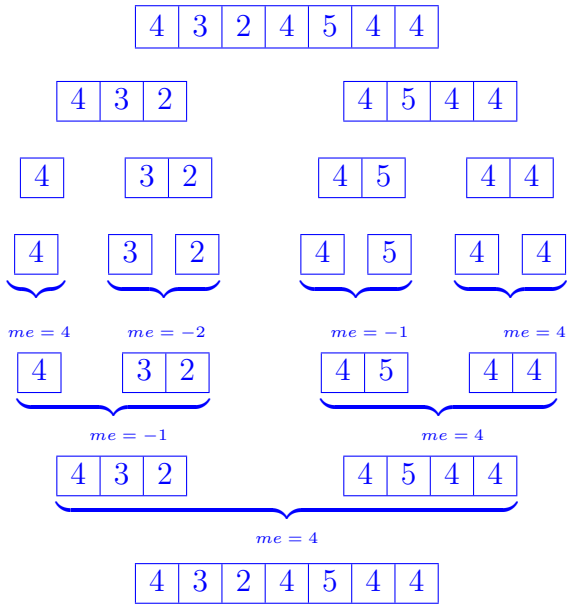
Divide and Conquer

A majority element of an array `A[l..r]` with size `n = r-l+1` is the value that occurs in more than $\frac{n}{2}$ positions. A property of the majority element of A is that it is also the majority element of the left or the right half of A or both.

For example, the majority element in `A = [2,4,2,3,2]` is 2 since it occurs in more than $\frac{5}{2} = 2.5$ positions and, indeed, 2 is also the majority element in the left half A_L`= [2,4,2]`. On the other hand, array `B =[2,2,4,1,4,2,3,5,3,2,4]` does not have a majority element since there is no value occurring in more than $\frac{11}{2} = 5.5$ positions.

Important: If an element x is the majority element of one of the halves of array A, it is not necessarily the majority element in A. In order to confirm it is, we still need to count the number of times it occurs in the whole array.

4.1 [5 points] Based on the above, draw a tree to illustrate the process of determining the majority element in array `A = [4,3,2,4,5,4,4]`. Annotate each node of the tree with its majority element.



4.2 [14 points] Design a **divide and conquer** algorithm which finds and returns the majority element in an array of positive integers. If there is no majority element, your algorithm should return -1. Use either C or pseudocode for your solution.

```
1 Algorithm: FindMajorityElement(A,l,r)
2 if l==r then return A[l];
3 m = ⌊(l+r)/2⌋;
4 meL = FindMajorityElement(A,l,m);
5 meR = FindMajorityElement(A,m+1,r);
6 Combine(A,l,r,meL,meR);
```

```
1 Algorithm: Combine(A,l,r,meL,meR)
2 cntL = 0; cntR = 0;
3 for i = l to r do
4     if A[i]==meL then cntL++;
5     if A[i]==meR then cntR++;
6 if cntL> ⌊(r − l + 1)/2⌋ then return meL;
7 if cntR> ⌊(r − l + 1)/2⌋ then return meR;
8 return-1;
```

4.3 [3 points] Write the recurrence for the complexity of your algorithm in 4.2. Solve the recurrence using the repeated substitution method.

$$T(n) = \begin{cases} 0 & \text{, if } n = 1 \\ 2T(\frac{n}{2}) + 2n & \text{, if } n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + 2n \\ &= 2(2T(n/4) + 2n/2) + 2n \\ &= 4T(n/4) + 4n \\ &= 4(2T(n/8) + 2n/4) + 4n \\ &= 8T(n/8) + 6n \\ T(n) &= 2^i T(n/2^i) + 2 * i * n \\ &= 2^{\lg n} T(n/2^{\lg n}) + 2 * \lg n \cdot n \\ &= nT(1) + 2 * \lg n \cdot n \\ &= \Theta(n \lg n) \end{aligned}$$

4.4 Solve the recurrence and give an asymptotic tight bound.

Master method case 2: $T(n) = \Theta(n \log n) = \Theta(n)$

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 — Midterm 2
Spring 2014 — 09.04.2014

Name: _____ Matriculation number: _____

Advice

Important: Only the German version is legally binding.

If you attend the module *Informatik IIb* only, you have 90 minutes to complete the exam. If you attend both modules, Informatik IIa and IIb, you have 120 minutes overall to complete both parts of the exam.

The following rules apply for the written part of the exam:

- 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 (10 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.
- For the exam Informatik IIb, only the following items are allowed:
  - One A4 sheet (2-sided) with your personal handwritten notes, written by yourself. Sheets that do not conform to this specification will be collected.
  - For students with a different first language than German: A foreign dictionary is allowed. The dictionary will be checked by a supervisor.
  - No additional items are allowed, notably 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	Total
Points Achieved				
Maximum Points	30	30	30	90

Exercise 1 30 Points

Linked Lists

Consider a linked list defined as follows:

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

struct node *head;
```

1.1 [15 points] Create a function `deleteAllButOne` that deletes all occurrences of a value `v` in a list apart from the first occurrence. Precisely work out the arguments of the function and give an example of how to call it.

Describe your solution using C or pseudocode.

Example: In Figure 1, (A) illustrates the initial list while (B) illustrates the list after calling `deleteAllButOne` with `v = 3`.

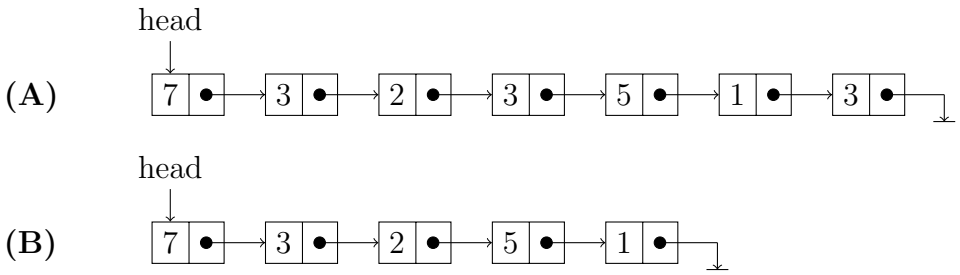


Figure 1: Deleting all occurrences but the first one of `v = 3`

Name: Matriculation number:

Algorithm: void deleteAllButOne(node *head, int v)

```
1  haveFound = 0;
2  *p = head;
3  if head == NULL then
4      return;
5  if p!=NULL && p->key ==v then
6      haveFound = 1;
7  while p->next != NULL do
8      if p->next->key == v then
9          if haveFound == 1 then
10             node *tmp = p->next;
11             p->next = tmp->next;
12             free(tmp);
13         else if haveFound == 0 then
14             haveFound = 1;
15             p = p -> next;
16         else
17             p = p -> next;
18 return;
```

1.2 [10 points] The median is the middle value that separates the higher half of a set of observations from the lower half. The median can be found by sorting all observations from the lowest value to highest value and picking the middle element (e.g., the median of 3, 3, 5, 9, 11 is 5). If there is an even number of observations the median is the mean of the two middle values, rounded down.

Create a function `int ListMedian(node *head)` for an **ordered linked list of positive integers**. Your function shall return the median of the list. In case the median cannot be computed, your function should return -1.

Describe your solution using C or pseudocode.

Algorithm: int ListMedian(node *head)

```
1  if head == null then return -1;
   /* Count elements                                          */
2  int count = 0;
3  float median;
4  node *q = head;
5  while q != null do
6  |   count = count + 1;
7  |   q = q->next;
   /* Select median                                           */
8  q = head;
9  if count % 2 == 1 then
10 |   for i=0 to ⌈(count + 1)/2⌉ do q = q->next;
11 |   median = q->key;
12 else
13 |   for i=0 to ⌊count/2⌋ do q = q->next;
14 |   median = (q->key + q->next->key)/2;
15 return median;
```

1.3 [5 points] Consider the `ListMedian` function from task 1.2. Modify it in such a way that it operates on **an ordered linked list of integers (positive or negative ones)** instead of an ordered linked list of only positive integers. What are the implications? How do you have to modify your solution?

SOLUTION 1

- The function in task 1.2 returns -1 in case the median cannot be computed, that is, when the list is empty.
- In case of a list of negative integers, the function might also return -1 if it is indeed the median.
- To solve this issue, we need to make sure the cases are distinct. So, we can define a struct message int type; int value;. The type could be 0 for error or 1 if the int assigned to value is valid.

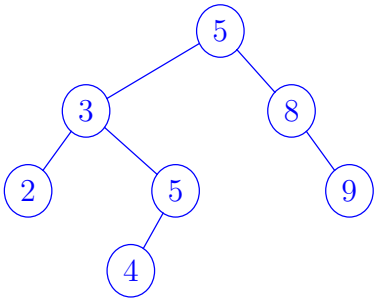
SOLUTION 2

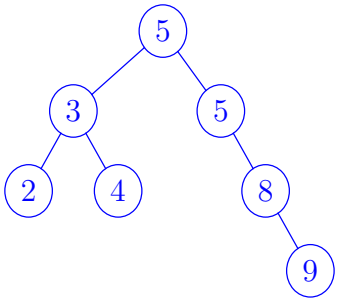
- int ListMedian(int *med,node *head)
- returns 1 if median found and 0 if not computed
- if it returns 1, then the median is stored in the med variable

Exercise 2 20 Points

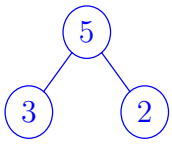
Heaps and Trees

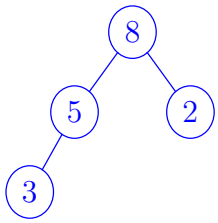
2.1 [5 points] Precisely define a binary search tree. Show the `binary search tree` that is the result of inserting the values **5 3 2 5 8 4 9** in the given order into an empty binary search tree.

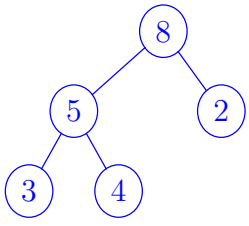


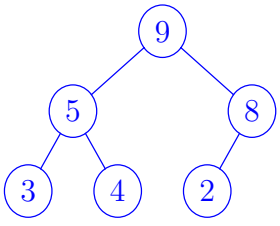


2.2 [10 points] Precisely define a binary heap. The values **5 3 2 ; 8 ; 4 9;** are inserted in in the given order into an empty `binary heap`. Draw the binary heap at the positions marked by a semicolon.



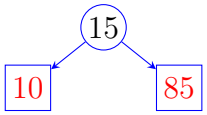


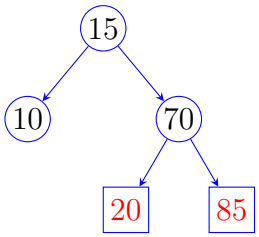


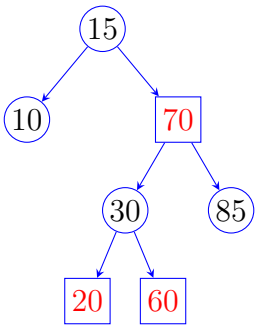


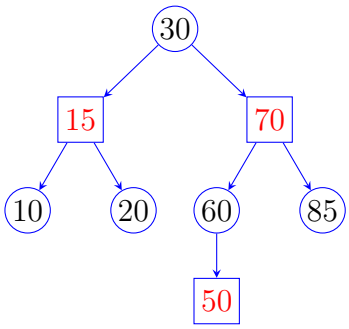
2.3 [17 points] Precisely define a red-black tree. The values **10 85 15 ; 70 20 ; 60 30 ; 50 ;** are inserted in in the given order into an empty `red-black tree`. Draw the red-black tree at the positions marked by a semicolon.

Name: Matriculation number:









Exercise 3 20 Points

Binary Search Trees

Assume a binary search tree defined as follows:

```
struct TreeNode {
  int key;
  struct TreeNode *lft;
  struct TreeNode *rgt;
};

struct TreeNode* root;
```

3.1 [10 points] Create a recursive function that inserts the node `x` into the binary search tree pointed to by `p`. Assume a binary search tree that is not allowed to contain duplicates.

```
void insertRec (struct TreeNode **p, struct TreeNode *x)
```

Describe your solution using C or pseudocode.

Algorithm: void insertRecursive(struct TreeNode **p, struct TreeNode *x)

```
1 if *root == NULL then
2 |   *root = item;
3 |   return;
4 if item->key < (*root)->key then
5 |   insertRecursive(&(*root)->lChild, item);
6 else if item->key > (*root)->key then
7 |   insertRecursive(&(*root)->rChild, item);
8 return;
```

3.2 [5 points] Consider a binary search tree that is not allowed to contain duplicates. The key of a given node shall be replaced by a new key without modifying the structure of the tree.

1. Consider the binary search tree in Figure 2. Determine all possible values with which we can replace the key values of the following nodes:

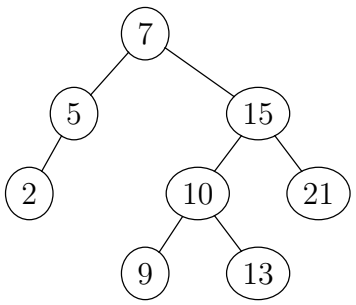


Figure 2: Example of a Binary Search Tree

Key Value	Possible Values
7	
9	

Key Value	Possible Values
7	6, (7), 8
9	8, (9)

2. Precisely define the values with which you can replace a given key value in a general binary search tree that is not allowed to contain duplicates.
   Given `node i` and trying to replace its key with `value x`, then for x the following should hold: $x > pred(i) \ \wedge \ x < succ(i) \ \wedge \ x \neq i$

3.3 [15 points] Create a function that determines the minimum possible value `vMin` with which the key of `x` can be replaced without violating the definition of a binary search tree.

```
int minKeyPossible(struct TreeNode *x)
```

Describe your solution using C or pseudocode.

Algorithm: int minLimit(tree *item)

```
1  int initKey = item->key;
2  tree *minElement;
3  if item->lChild ≠ NULL then
4  |   minElement = TreeMaximum(item->lChild);
5  |   return minElement->key
6  minElement = Parent(item);
7  while minElement ≠ NULL and item == minElement->lChild do
8  |   item = minElement;
9  |   minElement = Parent(item);
10 if minElement ≠ NULL then return maxElement->key;
11 ;
12 else return initKey;
13 ;
```

Algorithm: int maxLimit(tree *item)

```
1  int initKey = item->key;
2  tree *maxElement;
3  if item->rChild ≠ NULL then
4  |   maxElement = TreeMinimum(item->rChild);
5  |   return maxElement->key
6  maxElement = Parent(item);
7  while maxElement ≠ NULL and item == maxElement->rChild do
8  |   item = maxElement;
9  |   maxElement = Parent(item);
10 if maxElement ≠ NULL then return maxElement->key;
11 ;
12 else return initKey;
13 ;
```

Algorithm: int minKeyPossible(tree *item)

```
1  minLimit = minLimit(item);
2  maxLimit = maxLimit(item);
3  if minLimit == maxLimit then
4  |   return-1;
5  if minLimit+1 != item->key then
6  |   return minLimit+1 ;
7  else if item->key != maxLimit+1 then
8  |   return maxLimit+1 ;
```

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 — Midterm1
Spring 2015 — 27.03.2015

Name: _____ Matriculation number: _____

Advice

You have 90 minutes to complete the exam of InformatikIIb. The following rules should 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 (14 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.
- For the exam Informatik IIb, only the following items are allowed:
  - 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 dictionary is allowed. The dictionary will be checked by a supervisor.
  - No additional items are allowed, notably 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	15	15	15	15	60

Exercise 1 15 Points

Arrays

1.1 [5 points] Consider an array `A[0...n`$_A$`-1]` with `n`$_A$ integers, and an array `B[0...n`$_B$`-1]` with `n`$_B$ integers. Arrays A and B are both sorted in increasing order. Create a function `printDifference(A, n`$_A$`, B, n`$_B$`)` that prints the difference between arrays A and B as sets, i.e. prints each value of A that is not in B. Use either C or pseudocode for your solution.

Example: If `A = [1,2,2,3,6,10]` and `B = [2,2,3,3,3,4]`, your solution should print: **1 6 10**

brute force solution with $O(n^2)$ complexity:

Algo: printDiff(p)

for *i = 0* **to** *nA-1* **do**
  if *isFirst(i,A,nA)* $\wedge$ *notOccurs(A[i],B,nB)* **then** print A[i];

Algo: isFirst(i,A,nA)

return i==0 $\vee$ A[i] $\neq$ A[i-1]

Algo: notOccurs(x,A,nA)

notOcc = TRUE;
for *i = 0* **to** *nA-1* **do** notOcc = notOcc $\wedge$ A[i] $\neq$ x;
return notOcc;

synchronous traversal with $O(n)$ complexity:

```
void printDifference(int a[], int na, int b[], int nb) {
  int i, j = 0;
  for (i = 0; i < na; i++) {
    while (j < nb && a[i] > b[j]) j++;
    if (j == nb && (i == 0 || a[i] != a[i-1])
        || i == 0 || a[i] < b[j] && a[i] != a[i-1])
      printf("%d␣", a[i]);
  }
}
```

Name: Matriculation number:

1.2 [10 points] Consider a matrix of integers `A[n][m]`. Create a function `printInSpiral(int A[n][m], int n, int m)` that prints the values of A in an spiral form. Use either C or pseudocode for your solution.

Example: If $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$, with $n = 3$ and $m = 4$, your solution should print: **1 2 3 4 8 12 11 10 9 5 6 7**

```
void printInSpiral(int n, int m) {
  int top = 0, bot = n-1, lft = 0, rgt = m-1, i, j;
  while (1) {
    for (j = lft; j <= rgt; j++) printf("%d␣", a[top][j]);
    if (++top > bot) return;

    for (i = top; i <= bot; i++) printf("%d␣", a[i][rgt]);
    if (lft > --rgt) return;

    for (j = rgt; j >= lft; j--) printf("%d␣", a[bot][j]);
    if (top > --bot) return;

    for (i = bot; i >= top; i--) printf("%d␣", a[i][lft]);
    if (++lft > rgt) return;
  }
}
```

Exercise 2 15 Points

Asymptotic Complexity and Recurrences

2.1 [4 points] Calculate the asymptotic tight bound for the following functions and rank them by their order of growth (lowest first). Clearly work out the calculation steps in your solution: $f_1(n) = 2^n, f_2(n) = 2^{\log^3(n)} + 3, f_3(n) = 2n(1 + 3\log(n)), f_4(n) = 1000^\pi + \log(10), f_5(n) = (250n)^3, f_6(n) = \log^2(n) + 50\sqrt{n} + \log(n), f_7(n) = 4^{log_2 n}$

- $f_1(n) = 2^n \in \Theta(2^n)$
- $f_2(n) = 2^{\log^3(n)} + 3 = 2^{\log n \log^2 n} + 3 = (2^{\log n})^{\log^2 n} + 3$
  $= n^{\log^2 n} + 3 \in \Theta(n^{\log^2 n})$
- $f_3(n) = 2n(1 + 3\log(n)) = 2n + 6n\log(n) \in \Theta(n\log(n))$
- $f_4(n) = 1000^\pi + \log(10) \in \Theta(1)$
- $f_5(n) = (250n)^3 \in \Theta(n^3)$
- $f_6(n) = \log^2(n) + 50\sqrt{n} + \log(n) \in \Theta(\sqrt{n})$
- $f_7(n) = 4^{log_2 n} = (2^2)^{log_2 n} = (2^{log_2 n})^2 = \Theta(n^2)$

Solution: $f_4, f_6, f_3, f_7, f_5, f_2, f_1$

2.2 [5 points] Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and which case (1-3) applies.

Hint: $\sum_{i=1}^{n} \frac{1}{i} = \log(n)$

a) $T(n) = 4T(\frac{n}{2}) + n^2$ $a = 4$, $b = 2$, $f(n) = n^2$, Case 2: $\Theta(n^2 \log(n))$

b) $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log(n)}, T(1) = 0$

$$\begin{aligned} T(n) &= 2T(\tfrac{n}{2}) + \tfrac{n}{\lg(n)} \\ &= 4T(\tfrac{n}{4}) + \tfrac{n}{\lg(\frac{n}{2})} + \tfrac{n}{\lg(n)} \\ &= 8T(\tfrac{n}{8}) + \tfrac{n}{\lg(\frac{n}{4})} + \tfrac{n}{\lg(\frac{n}{2})} + \tfrac{n}{\lg(n)} \\ &= \textstyle\sum_{i=0}^{\lg(n)-1} n/\lg(n/2^i) \\ &= \textstyle\sum_{i=0}^{\lg(n)-1} n/\lg(2^i) \\ &= \textstyle\sum_{i=0}^{\lg(n)-1} n/i \\ &= n \lg\lg(n-1) \\ &= \Theta(n \lg\lg n) \end{aligned}$$

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

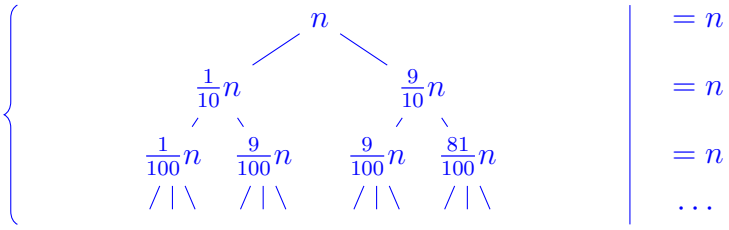
$a = 3$, $b = 2$, $f(n) = n$, Case 1: $\Theta(n^{\log 3})$

2.3 [6 points] Consider the recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n) = T(n/10) + T(9n/10) + n & \text{if } n > 1 \end{cases}$$

a) Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Include the tree-based calculations that led to your estimate.

$h = \log_{\frac{1}{10}} n$



Longest branch on the right. Tree grows until $\left(\frac{1}{10}\right)^h n = 1 \implies h = \log_{\frac{1}{10}} n$

Guess: $O(n \lg n)$

b) Prove the correctness of your estimate using induction.

Proof by induction:

$$\begin{aligned}
T(n) &= T(n/10) + T(9n/10) + n && \text{(recurrence)} \\
&\leq c\tfrac{n}{10} \lg \tfrac{n}{10} + c\tfrac{9n}{10} \lg \tfrac{9n}{10} + n && \text{(inductive hyp.)} \\
&= cn \lg n + n + \tfrac{cn}{10} \lg \tfrac{1}{10} + \tfrac{9cn}{10} lg \tfrac{9}{10} && \text{(reformating)} \\
&= cn \lg n + n \left(1 + \tfrac{c}{10}(9 \lg(9) - 10 \lg(10)\right) && \text{(reformating)} \\
&\leq cn \lg n && \left(\text{for } c \geq \tfrac{10}{10 \lg 10 - 9 \lg 9}, n > 1\right)
\end{aligned}$$

Name: Matriculation number:

Exercise 3 15 Points

Runtime and Recursion

Algo: whatDoesItDo(A, n)

```
tmp = 0;
do
    ready = 0;
    for i = n-1 to 1 do
        if A[i-1]>A[i] then
            tmp = A[i-1];
            A[i-1] = A[i];
            A[i] = tmp;
            ready =1;
    for i = 1 to n-1 do
        if A[i-1]>A[i] then
            tmp = A[i-1];
            A[i-1] = A[i];
            A[i] = tmp;
            ready =1;
while ready=1;
```

3.1 [2 points] The algorithm `whatDoesItDo(A,n)` gets an array `A[0...n-1]` of n integers as an input. Apply the algorithm on the array A=`[ 8, 5, 10, 1, 3]` and complete each line of the following matrix if the content of A is modified. The already-completed line of the matrix shows the initial content of A.

i	**A[0]**	**A[1]**	**A[2]**	**A[3]**	**A[4]**
-	8	5	10	1	3
3	8	5	1	10	3
2	8	1	5	10	3
1	1	8	5	10	3
2	1	5	8	10	3
4	1	5	8	3	10
3	1	5	3	8	10
2	1	3	5	8	10

3.2 [3 points] Describe in words (maximum three lines) what the algorithm `whatDoesItDo(A,n)` does. What is the best and worst case for this algorithm?

The algorithm sorts the array A, it is a variation of bubble sort, this variation sorts A in both directions on each pass through the list.

Best case: O(n), array already sorted in ascending order.
Worst case: O(n^2), array sorted in descending order.

3.3 [4 points] Analyze the steps of the algorithm `whatDoesItDo(A,n)` and calculate its running time (worst case).

line 1	c_1	1
line 2	c_2	1
line 3	c_3	$\frac{n}{2}$*
line 4	c_4	$\frac{n}{2}$*$(n-1)$
line 5-9	c_5	$\frac{n}{2}$*$(n-2)$
line 10	c_6	$\frac{n}{2}$*$(n-1)$
line 11-15	c_7	$\frac{n}{2}$*$(n-1)$
line 16	c_8	$\frac{n}{2}$

* $\frac{n}{2}$ is the worst case, actually if the array is sorted the execution time would be 1 instead of $\frac{n}{2}$

Running time: $c_1 + c_2 + c_3 \cdot \frac{n}{2} + c_4 \cdot \frac{n}{2} \cdot (n-1) + 4 \cdot c_5 \cdot \frac{n}{2} \cdot (n-2) + c_6 \cdot \frac{n}{2} \cdot (n-1) + 4 \cdot c_7 \cdot \frac{n}{2} \cdot (n-1) + c_8 \cdot \frac{n}{2}$

3.4 [6 points] Give a recursive version of the algorithm `whatDoesItDo(A,n)`. Use either C or pseudocode for your solution.

```
void cocktailSort(int a[], int l, int r) {
    int i, tmp, ready = 0;
    if (l >= r) return;
    for (i = r; i >= l+1; i--) {
        if (a[i-1] > a[i]) {
            tmp = a[i-1];
            a[i-1] = a[i];
            a[i] = tmp;
            ready = 1;
        }
    }
    for (i = l+1; i <= r; i++) {
        if (a[i-1] > a[i]) {
            tmp = a[i-1];
            a[i-1] = a[i];
            a[i] = tmp;
            ready = 1;
        }
    }
    if (ready) cocktailSort(a, l+1, r-1);
}
```

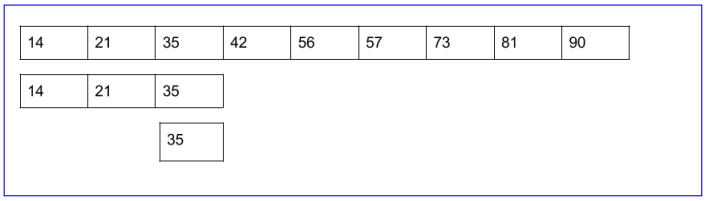
Name: Matriculation number:

Exercise 4 15 Points

Divide and Conquer

The ternary search algorithm determines the index of an element with the desired value k within a sorted array $A[0...n-1]$ or concludes that k is not included in the array. Firstly, it divides the input array into three equal partitions: **I** (A[0] ... A[*mid1*-1]), **II** (A[*mid1*] ... A[*mid2*-1]) and **III** (A[*mid2*] ... A[n-1]). Afterwards, it calls itself recursively on one of the three partitions based on the result of the comparison of the desired value k with the two middle elements, *mid1* and *mid2*. The recursion terminates when the element k is found or until it can be reported unfound.

4.1 [2 points] Based on the ternary search description, draw a tree to illustrate the process of determining the position of the number 35, given `A = [14, 21, 35, 42, 56, 57, 73, 81, 90]`.



4.2 [10 points] Implement the ternary search algorithm which finds and returns a desired value k. If k is not in the array, your algorithm should return -1. Use either C or pseudocode for your solution.

```
int ternarySearch(int a[], int l, int r, int x) {
    int mid1, mid2;
    if (r >= l) {
        if (l == r) {
            if (a[l] == x) return l;
            else return -1;
        }
        mid1 = l + (r - l) * 1 / 3;
        mid2 = l + (r - l) * 2 / 3;
        if (x <= a[mid1]) r = mid1;
        else if (x <= a[mid2]) { l = mid1+1; r = mid2; }
        else l = mid2+1;
        return ternarySearch(a, l, r, x);
    }
    return -1;
}
```

4.3 [3 points] Write the recurrence for the complexity of your algorithm in 4.2. Solve the recurrence using the repeated substitution method.

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(\frac{n}{3}) + 2 & \text{if } n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= T(n/3) + 2 \\ &= T(n/9)) + 2 + 2 \\ &= T(n/27) + 2 + 2 + 2 \\ T(n) &= T(n/3^i) + 2 * i \\ &= \log_3(n) + T(1) \\ &= \Theta(\log_3 n) \end{aligned}$$

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 — Midterm 2
Spring 2015 — 24.04.2015

Name: _____ Matriculation number: _____

Advice

You have 90 minutes to complete the exam of InformatikIIb. 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 (12 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.
- For this exam, only the following items are allowed:
  - One A4 sheet (2-sided) with your personal handwritten notes. Sheets that do not conform to this specification will be collected.
  - A foreign dictionary is allowed. The dictionary will be checked by a supervisor.
  - No additional items are allowed, notably 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	Total
Points Achieved				
Maximum Points	20	20	20	60

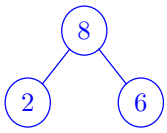
Exercise 1 20 Points

Heaps

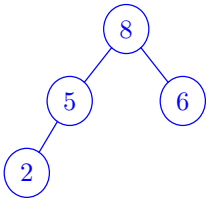
1. 1 [10 points] Define a max-heap. The values **6 2 8 ; 5 ; 9 4 ;** are inserted in the given order into an empty `max-heap`. Draw the max-heap at the positions marked by a semicolon.

   A heap is a binary tree that satisfies the following properties:

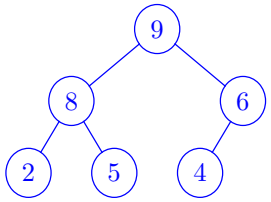
   - all levels of non-maximal depth d are full (have $2d$ nodes) [1.5 points]
   - all leaves with maximal depth are as far left as possible [1.5 points]
   - each node is greater than or equal to all its children [1.5 points]



[2 points]



[1.5 points]



[2 points]

1.2 [2 points] Assume a max-heap $A[1...n]$. Determine the parent and the sibling of element $A[i]$.

- i=0 (root): The root has neither a parent, nor a sibling
- i $\neq$ 0: parent_index = $\lfloor i/2 \rfloor$
- i **mod** 2 = 0 **and** i = n: no sibling
- i **mod** 2 = 0 **and** i $\neq$ n: sibling_index = $i + 1$
- i **mod** 2 = 1: sibling_index = $i - 1$

[0.5 points] $--> $ root/ [0.5 points] $--> $ parent / [1 point] $--> $ siblings

1.3 [8 points] Use induction to prove the following property of heaps:

"In a heap the number of nodes with exactly one child is either zero or one."

Base case - Heap with n nodes:

- BC1: $n = 2$, n is even
  - The root node $n/2 = 1$ is the only node with children.
  - The root node $n/2 = 1$ has one child: a left child and no right child.
- BC2: $n = 3$, n is odd
  - The root node $\lfloor n/2 \rfloor = 1$ is the only node with children.
  - The root node $\lfloor n/2 \rfloor = 1$ has two children: a left child and a right child.

Inductive step from n nodes to $n + 1$ nodes:

- C1: n is even
  - The parent of node $n + 1$ is node $\lfloor (n + 1)/2 \rfloor$.
  - Since n is even we have $\lfloor (n + 1)/2 \rfloor = \lfloor n/2 \rfloor$
  - Node $\lfloor n/2 \rfloor$ has one child (BC1).
  - After adding a child to node $\lfloor n/2 \rfloor$ there is no node with exactly one child and we have a tree with an odd number of nodes.
- C2: n is odd
  - The parent of node $n + 1$ is node $\lfloor (n + 1)/2 \rfloor$.
  - Since n is odd we have $\lfloor (n + 1)/2 \rfloor = (n + 1)/2$
  - Before the addition node $(n + 1)/2$ had no children since $2 * (n + 1)/2 > n$ and $2 * (n + 1)/2 + 1 > n$.
  - After adding a child to node $(n + 1)/2$ there is one more node with exactly one child. Since before there was no node with one child (BC2) we now have one node with one child and we have an even number of nodes.

- Idea [2 points]
- Base case [2 points]
- Inductive Step: Case 1 [2 points], Case 2 [2 points]

Exercise 2 20 Points

Lists

Assume a linked list of integers with the following C definition:

```
struct node { int key; struct node* next; }
struct node* u;
```

The task is to develop a sorting algorithm for unordered linked lists. Specifically, given an unsorted list u, a sorted list s shall be constructed. In each iteration, the smallest element in u shall be appended to s. The sorting must be done by changing pointers and without copying any elements of u.

2.1 [8 points] Illustrate the sorting algorithm on the example list shown in Figure 1. Show the state of the lists u and s after each iteration.

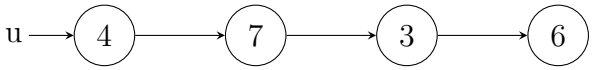
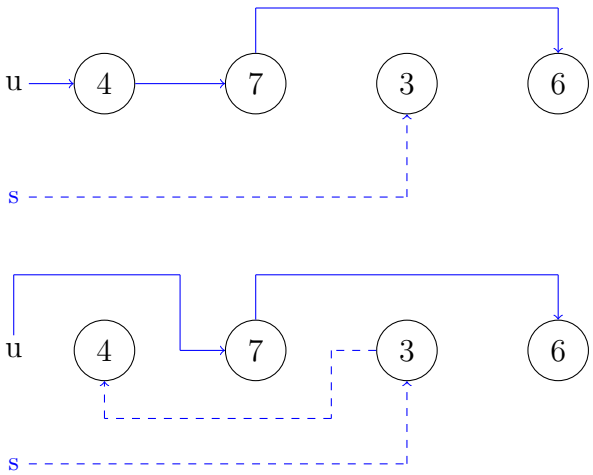
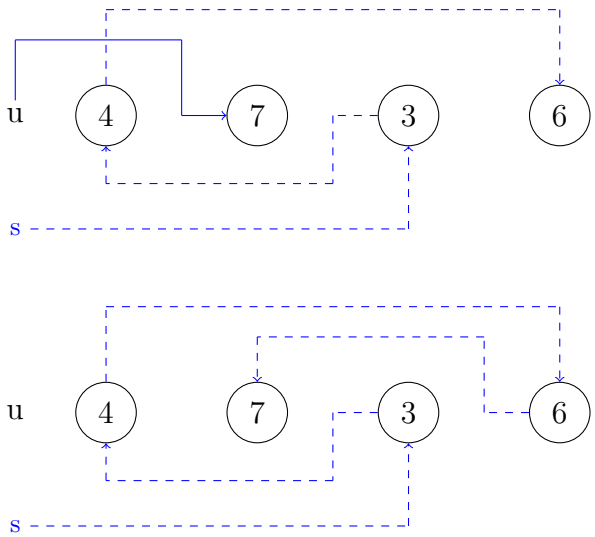


Figure 1: Unordered Linked List





2.2 [10 points] Use C or pseudocode to describe the above sorting algorithm for an unordered linked list.

```
1  s = NULL; e = NULL;
2  while u ≠ NULL do
3      m = u; p = NULL; q = u;
4      while q→next ≠ NULL do
5          if q→next→val < m→val then p=q; m = q→next;
6          q = q→next;
7      if p == NULL then u = u→next else p→next = m→next;
8      if s == NULL then u = m;
9      if e == NULL then e = m else e→next = m; m→next = NULL; e = m;
```

** p = previous to minimum ** q = iteration variable

[1 point] −− > pointer initialization

[4 points] −− > find minimum

[2 points] −− > removing element from u (line 7)

[3 points] −− > adding element in s (lines 8,9)

2.3 [2 points] Compute the number of pointer modifications that have been performed by the end of the sorting.

Action	Modifications
Remove Node From list u	1
Add Node To list s	2

Given that list u has n elements:

- **n Iterations:** $(1 + 2) \cdot n = 3n$ modifications

[1 point] −− > Explanation of the modifications

[1 point] −− > Adding them

If someone answered in terms of the example:

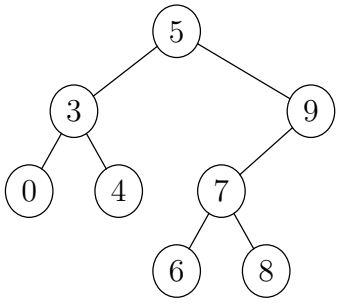
- **4 elements (n=4):** $3 \cdot 4 = 12$ modifications

Name: Matriculation number:

Exercise 3 20 Points

Binary Search Trees

A binary search tree is illustrated in the figure below, along with its definition in C:



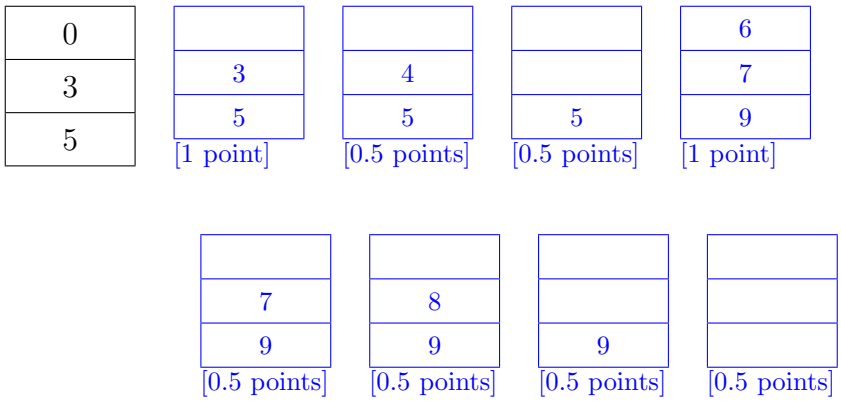

```
struct TreeNode {
    int key;
    struct TreeNode* lChild;
    struct TreeNode* rChild;
};
```

3.1 [5 points] A non-recursive inorder tree traversal of a binary search tree is implemented using a stack according to the procedure described below.

While the stack is not empty, for each node x at the top of the stack:

- If x has no right child, pop x.
- If x has a right child y, pop x and push y on the stack together with the leftmost path of the subtree rooted in y (the leftmost path consists of all nodes traversed by going left only).

Given the initial stack below, illustrate the stack each time one of the above two modifications has been performed.



3.2 [8 points] Based on the description in 3.1, define an iterator for a binary search tree as an ADT. Also, provide the functions *init* and *next*, which initialise the ADT and return the next node to be visited based on the inorder traversal, respectively.

```
struct iterator {
  struct TreeNode *val[MAX_SIZE];
  int size;
};
```

Algorithm: init(struct TreeNode *r)

```
1 struct iterator* it = malloc(sizeof(struct iterator));
2 it→size = 0;
3 it→val[it→size++] = r;
4 struct TreeNode *x = r;
5 while x→left ≠ NULL do
6 |   x = x→left;
7 |   it→val[it→size++] = x;;
8 return it;
```

Algorithm: next(struct iterator *it)

```
1 if it→size = 0 then return NULL;
2 t = it→val[- - (it→size)];
3 x = x→right;
4 while x ≠ NULL do
5 |   it→val[it→size++] = x;
6 |   x = x→left;
7 return t;
```

[2 point] −− > ADT Definition

[3 points] −− > Init (arguments/return [0.5 points], malloc/size [1 point], push root [0.5 points], leftmost path [1 point])

[3 points] −− > Next (arguments/return [0.5 points], empty [0.5 points], pop [0.5 points], push right child [0.5 points], leftmost path [1 point])

3.3 [7 points] Using the ADT from Task 3.2, create a function that returns true if two given binary search trees include the same values.

Algorithm: checkEqual(struct TreeNode *r1, struct TreeNode *r2)

```
1 struct iterator *it1 = init(r1);
2 struct iterator *it2 = init(r2);
3 struct TreeNode *n1, *n2;
4 repeat
5 |   n1 = iterNext(r1);
6 |   n2 = iterNext(r2);
7 until n1 == NULL ∨ n2 == NULL ∨ n1→val ≠ n2→val;
8 return n1 == n2;
```

[2 point] −− > Initialise iterators

[2 point] −− > Get the first element to be visited

[1 points] −− > Properly iterate while there is no more element to be visited [while]

[2 points] −− > key comparison/proper actions [different - 1 point, equal - 1 point]

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 — Midterm1
Spring 2016 — 08.04.2016

Name: _____ Matriculation number: _____

Advice

You have 90 minutes to complete the exam of InformatikIIb. 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 (12 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.
- For the exam Informatik IIb, 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. Notably 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	Total
Points Achieved				
Maximum Points	20	20	20	60

Exercise 1 20 Points

Asymptotic Complexity and Recurrences

1.1 [4 points] Calculate the asymptotic tight bound for the following functions and rank them by their order of growth (lowest first). Clearly work out the calculation steps in your solution: $f_1(n) = \lg 8 \cdot n \cdot (3 + 8 \lg n)$, $f_2(n) = 2^{n+10}$, $f_3(n) = \log \pi^{\log n} \cdot \log (n \cdot 10^{\log 35})$, $f_4(n) = max((\log n)^4, \sqrt{n})$, $f_5(n) = 10n^7 + 5n^3 + 16^{\lg 10n^2}$, $f_6(n) = (n + 10)!$

- $f_1(n) = \lg 8 \cdot n \cdot (3 + 8 \lg n) \in \Theta(n \log n)$
- $f_2(n) = 2^{n+10} \in \Theta(2^n)$
- $f_3(n) = \log \pi^{\log n} \cdot \log (n \cdot 10^{\log 35}) \in \Theta((\log n)^2)$
- $f_4(n) = max((\log n)^4, \sqrt{n}) \in \Theta(\sqrt{n})$
- $f_5(n) = 10n^7 + 5n^3 + 16^{\lg 10n^2} = 10n^7 + 5n^3 + (10n^2)^{\frac{1}{\log_{16} 2}} = 10n^7 + 5n^3 + (10n^2)^4 \in \Theta(n^8)$
- $f_6(n) = (n + 10)! \in \Theta((n + 10)!)$

Ranking: $f_3, f_4, f_1, f_5, f_2, f_6$

1.2 [2 points] Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and which case (1-3) applies.

a) $T(n) = 27T(\frac{n}{3}) + 4n^3 + 7n^2$
$a = 27$, $b = 3$, $f(n) = \Theta(n^3)$, Case 2: T(n) = $\Theta(n^3 \log_3 n)$

b) $T(n) = 9T(\frac{n}{4}) + n^3$
$a = 9$, $b = 4$, $f(n) = n^3$, Case 3: $\Theta(n^3)$
Regularity condition: if $c = \frac{9}{64}, a \cdot f(n/b) = 9 \cdot (\frac{n}{4})^3 \leq \frac{9}{64} \cdot n^3$
$T(n) = \Theta(n^3)$

1.3 [6 points] Consider the following recurrence.

$$T(n) = \begin{cases} 4 & \text{, if } n = 1 \\ 2T(n/2) + 5n + 1 & \text{, if } n > 1 \end{cases}$$

a) Compute the value of $T(n)$ for $n = 8$.

$$\begin{aligned} T(8) &= 2T(4) + 5 \cdot 8 + 1 \\ &= 2(2T(2) + 5 \cdot 4 + 1) + 41 \\ &= 4(2T(1) + 5 \cdot 2 + 1) + 83 \\ &= 159 \end{aligned}$$

b) Solve recurrence exactly by giving a closed form expression (without $T(n)$ on the right side of "=").

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 5n + 1 \\ &= 2\left[2T(\frac{n}{2^2}) + 5\frac{n}{2} + 1\right] + 5n + 1 = 2^2T(\frac{n}{2^2}) + 2 \cdot 5n + (1+2) \\ &= 2^2\left[2T(\frac{n}{2^3}) + 5\frac{n}{2^2} + 1\right] + 2 \cdot 5n + (1+2) = 2^3T(\frac{n}{2^3}) + 3 \cdot 5n + (1+2+2^2) \\ &= ... \\ &= 2^iT(\frac{n}{2^i}) + i \cdot 5n + (1 + 2 + ... + 2^{i-1}) \end{aligned}$$

For $i = lgn$, we have:

$$\begin{aligned} T(n) &= 2^{lgn}T(\frac{n}{2^{lgn}}) + lgn \cdot 5n + (1 + 2 + ... + 2^{lgn-1}) \\ &= nT(1) + 5nlgn + 2^{lgn} - 1 \\ &= 5nlgn + 5n - 1 \end{aligned}$$

c) Find the simplest $f(n)$, such that $T(n) = \Theta(f(n))$.
$f(n) = nlgn$

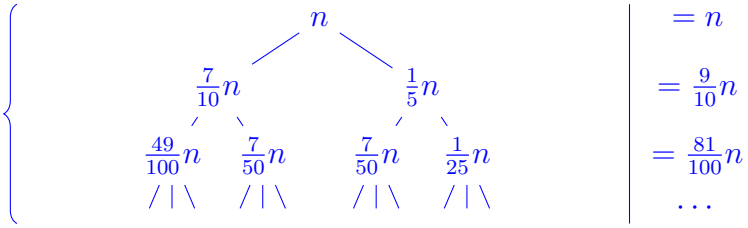
Name: Matriculation number:

1.4 [8 points] Consider the following recurrence:

$$T(n) = \begin{cases} 1 & \text{, if } n = 1 \\ T(n/5) + T(7n/10) + n & \text{, if } n > 1 \end{cases}$$

a) Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Include the tree-based calculations that led to your estimate.

$h = \log_{\frac{10}{7}} n$



$= n$

$= \frac{9}{10}n$

$= \frac{81}{100}n$

$\ldots$

Longest branch on the left. Tree grows until $\left(\frac{7}{10}\right)^h n = 1 \implies h = \log_{\frac{10}{7}} n$

Guess: $O(n)$

b) Prove the correctness of your estimate using induction.

Proof by induction:

$$\begin{aligned} T(n) &= T(n/5) + T(7n/10) + n && \text{(recurrence)} \\ &\leq c\tfrac{n}{5} + c\tfrac{7n}{10} + n && \text{(inductive hyp.)} \\ &= c(\tfrac{9n}{10}) + n && \text{(reformatting)} \\ &= n(\tfrac{9c}{10} + 1) && \text{(reformatting)} \\ &\leq cn && \text{(for } c \geq 10, n > 1\text{)} \end{aligned}$$

Exercise 2 20 Points

Runtime and Recursion

Algorithm: alg(A,n)

```
1  for i = 1 to ⌊n/2⌋ do
2      min = i;
3      max = n − i + 1;
4      if A[min] > A[max] then
5          exchange A[min] and A[max];
6      for j = i + 1 to n − i do
7          if A[j] < A[min] then
8              min = j;
9          if A[j] > A[max] then
10             max = j;
11     exchange A[i] and A[min];
12     exchange A[n − i + 1] and A[max];
```

2.1 [2 points] The algorithm `alg(A,n)` gets as inputs an array `A[1...n]` and the number `n` of integers it contains. Assume A=`[5, 7, 6, 3, 2, 1, 4]`. Complete the following matrix where you need to show the value of i and the content of A after each execution of the outer for loop. The first line of the matrix shows the initial array A before the execution of the loop.

i	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
-	5	7	6	3	2	1	4
1	1	5	6	3	2	4	7
2	1	2	5	3	4	6	7
3	1	2	3	4	5	6	7

2.2 [2 points] Describe in words (maximum three lines) what the algorithm `alg(A,n)` does.

The algorithm sorts the array. It's a variant of selection sort. In every pass it finds both the minimum and maximum values in the list. This reduces the number of scans of the list by a factor of 2, eliminating some loop overhead but not actually decreasing the number of comparisons or swaps.

2.3 [8 points] Analyze the steps of the algorithm `alg(A,n)`, calculate its exact runtime and its asymptotic complexity.

line 1	c_1	$\frac{n}{2}+1$
line 2	c_2	$\frac{n}{2}$
line 3	c_3	$\frac{n}{2}$
line 4	c_4	$\frac{n}{2}$
line 5	c_5	$\alpha_1^* \frac{n}{2}$
line 6	c_6	$^{**}\frac{1}{4}(n^2-2n)^{***}+\frac{n}{2}$
line 7	c_7	$\frac{1}{4}(n^2-2n)$
line 8	c_8	$\alpha_2^{****}\frac{1}{4}(n^2-2n)$
line 9	c_9	$\frac{1}{4}(n^2-2n)$
line 10	c_{10}	$\alpha_3^{****}\frac{1}{4}(n^2-2n)$
line 11	c_{11}	$\frac{n}{2}$
line 12	c_{12}	$\frac{n}{2}$

* $0 \leq \alpha_1 \leq 1$

** $\frac{n}{2}+1$ times for $j = min+1$ to $max-1$ and $\frac{n}{2}+1$ times for termination condition

*** $\sum_{i=1}^{\frac{n}{2}}(n-2i) = \frac{1}{4}(n^2-2n)$

**** $0 \leq \alpha_2 \leq 1$, $0 \leq \alpha_3 \leq 1$, $\alpha_2+\alpha_3 \leq 1$,

Running time:

$T(n) = c_1 + \frac{n}{2}(c_1 + c_2 + c_3 + c_4 + \alpha_1 c_5 + c_6 + c_{11} + c_{12}) + \frac{1}{4}(n^2 - 2n)(c_7 + \alpha_2 c_8 + c_9 + \alpha_3 c_{10})$
$= \frac{n^2}{2}(c_7 + \alpha_2 c_8 + c_9 + \alpha_3 c_{10}) + \frac{n}{2}(c_1 + c_2 + c_3 + c_4 + \alpha_1 c_5 + c_6 + c_{11} + c_{12} - 2c_7 - 2\alpha_2 c_8 - 2c_9 - 2\alpha_3 c_{10}) + c_1$
$\in O(n^2)$

2.4 [8 points] Give a recursive version of the algorithm `alg(A,n)`. Use either C or pseudocode for your solution.

Algorithm: BidirectionSelectionSortRec(A, l, r)

```
1  if l ≤ r then
2      min = l;
3      max = r;
4      if A[min] > A[max] then
5          exchange A[min] and A[max];
6      for j = l + 1 to r − 1 do
7          if A[j] < A[min] then
8              min = j;
9          if A[j] > A[max] then
10             max = j;
11     exchange A[l] and A[min];
12     exchange A[r] and A[max];
13     BidirectionSelectionSortRec(A, l + 1, r − 1);
```

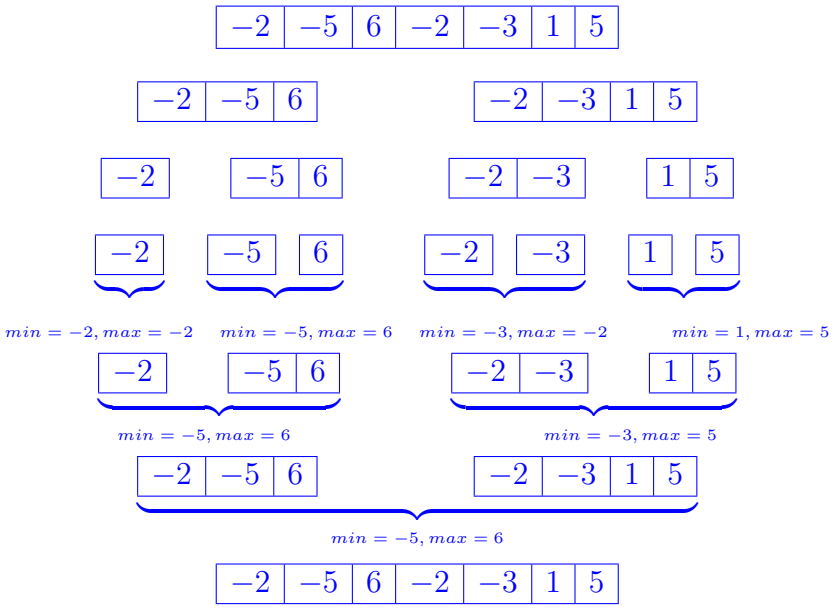
Exercise 3 20 Points

Divide and Conquer

The min-max search algorithm determines the minimum and maximum elements of an unsorted array $A[1...n]$. First, it divides the input array into two equal partitions: **I** ($A[1]$ ... $A[mid]$) and **II** ($A[mid+1]$ ... $A[n]$). Afterwards, it calls itself recursively on both partitions to find the minimum and the maximum of each partition, then combines them to find the global minimum and maximum elements.

For example, if the given array is `[-2, -5, 6, -2, -3, 1, 5]`, then the minimum is -5 and the maximum is 6.

3.1 [4 points] Based on the above, draw a tree to illustrate the process of determining the minimum and the maximum of the array `A = [-2, -5, 6, -2, -3, 1, 5]`. Annotate each node of the tree with its minimum and maximum.



3.2 [10 points] Design a **divide and conquer** algorithm which finds and returns the minimum and maximum elements of an unsorted array. Use either C or pseudocode for your solution.

```
Algorithm: MinMax(A,l,r)
1  if l = r then
2      return (A[l], A[l]);
3  else
4      m = ⌊(l+r)/2⌋;
5      min1, max1 = MinMax(A, l, m);
6      min2, max2 = MinMax(A, m+1, r);
7      if min1 < min2 then
8          min = min1;
9      else
10         min = min2;
11     if max1 > max2 then
12         max = max1;
13     else
14         max = max2;
15 return min, max;
```

3.3 [6 points] Write the recurrence for the complexity of your algorithm in Task 3.2. Solve the recurrence using the repeated substitution method.

$$T(n) = \begin{cases} 0 & \text{, if } n = 1 \\ 2T(\frac{n}{2}) + 1 & \text{, if } n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + 1 \\ &= 2(2T(n/4) + 1) + 1 \\ &= 4T(n/4) + 3 \\ &= 4(2T(n/8) + 1) + 3 \\ &= 8T(n/8) + 7 \\ T(n) &= 2^i T(n/2^i) + 2^i - 1 \\ &= 2^{\lg n} T(n/2^{\lg n}) + 2^{\lg n} - 1 \\ &= nT(1) + n - 1 \\ &= \Theta(n) \end{aligned}$$

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
Spring 2016

Midterm 2
29.04.2016

Name: _____ Matriculation number: _____

Advice

You have 90 minutes to complete the exam of Informatik IIb. 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 (12 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.
- For the exam Informatik IIb, 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. Notably 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	Total
Points Achieved				
Maximum Points	20	20	20	60

Exercise 1 20 Points

Stacks and Queues

The goal of this exercise is to implement a LIFO stack using two FIFO queues.

1.1 [5 points] Consider a **stack** of integers implemented using an array of size 5 and stack size s. Illustrate the contents of this stack and the value of its size s, after applying the following operations:

- push(10), push(20), push(30)

1	2	3	4	5
10	20	30		

s
3

- pop(), push(40)

1	2	3	4	5
10	20	40		

s
3

- pop(), pop(), pop(), push(50)

1	2	3	4	5
50				

s
1

- 0.5 point each number
- (-1 point) if correct but reversed.
- (-1 point) if written_size = correct_size +/-1.

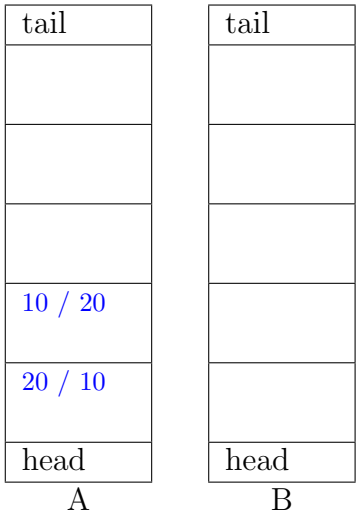
Name: Matriculation number:

1.2 [8 points] In this task you shall use two FIFO queues to implement a stack. Given two queues (A and B), illustrate their states and the sequence of operations applied on each queue in order to mimic the LIFO behaviour of a stack. The operations you can apply on a queue are: (i) `isEmpty(A)` (ii) `enqueue(A, value)`, (iii) `dequeue(A)`, and (iv) `swap(A, B)`. Function `swap(A, B)` swaps the content of two queues. For example, if it is applied on the following queues A=[2, 3, 5] and B=[6, 8], after the execution the content of the queues will be: A=[6, 8] and B=[2, 3, 5].

Hint: At the end of each operation (either push or pop) all elements should be stored in Queue A. Queue B should be used only to temporarily store elements in order to get access to the required element.

There are two different solutions. According to the first solution, `pop(A)` runs in constant time ($O(1)$) and `push(A, value)` runs in linear time ($O(n)$). According to the second solution, `pop(A)` runs in linear time and `push(A, value)` in constant time. In tables hereafter the first solution is written on the left side and the second on the right.

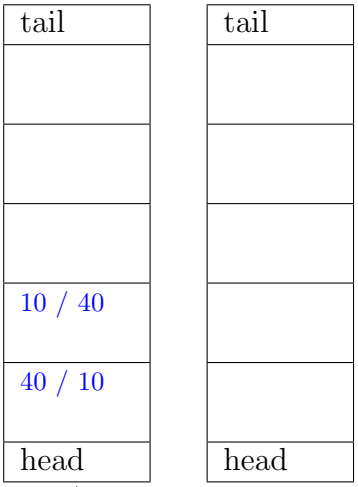
i) push(10), push(20)



queue		operation		element	
B	A	enqueue	enqueue	10	10
A,B	A	swap	enqueue	-	20
B	-	enqueue	-	20	-
A	-	dequeue	-	10	-
B	-	enqueue	-	10	-
A,B	-	swap	-	-	-

- 4 points: see bellow.

ii) pop(), push(40)



queue		operation		element	
A	A	dequeue	dequeue	20	10
B	B	enqueue	enqueue	40	10
A	A	dequeue	dequeue	10	20
B	A,B	enqueue	swap	10	-
A,B	A	swap	enqueue	-	40

- 4 points: see below:
  - 1 points: correct content of queue A (0.5 point/number)
  - 3 points: correct operations (1.5 point/LIFO operation)
    - 2/3 points: if operations lead to currect result but in a "strange" way (more operations than needed).
    - 1/1.5: if all operations instead of swap

Name: Matriculation number:

1.3 [7 points] Use C or pseudo-code to describe the algorithms for operations *push(A, B, value)* and *pop(A, B)* that mimic the LIFO behaviour of a stack using two queues A and B.

Solution 1

```
Algorithm: Pop(A, B)
1 Return Dequeue(A);
```

```
Algorithm: Push(A, B, v)
1 Enqueue(B, v);
2 while not isEmpty(A) do
3 |   t = Dequeue(A);
4 |   Enqueue(B, t);
5 Swap(A, B);
```

Solution 2

```
Algorithm: Pop(A, B)
1 while not isEmpty(A) do
2 |   t = Dequeue(A);
3 |   if not isEmpty(A) then
4 |   |   Enqueue(B, t);
5 Swap(A, B);
6 Return t;
```

```
Algorithm: Push(A, B, v)
1 Enqueue(A, v);
```

- Small mistakes:
  - Both solutions
    - If loop instead of swap (-1)
    - If an algorithm works but does more operations than needed (1.5)
  - Solution 1
    - 2 point: Correct pop operation
    - 1 point: Enqueue in B
    - 2 points: Dequeue everything from A and enqueue in B
    - 2 point: swap queues
  - Solution 2
    - 2 point: Correct push operation
    - 1 point: Dequeue everything from A
    - 2 points: Enqueue everything but the last element in B (1/2 if they enqueue and the last element in B).
    - 2 point: swap queues and return last element
- If one algorithm only works: 3.5
- If both algorithms are correct but O(n): 4/7
- If only the simple algorithm is correct/given but in T1.2 use its complex version 1/3.5 + 0/3.5

Exercise 2 20 Points

Heaps

2.1 [5 points] Consider the arrays given below. Determine which one is min-heap, which one is max-heap, and which one is not a heap.

```
A = [ 90, 15, 10, 7, 12, 2 ]
B = [ 6, 8, 15, 20, 10, 25, 33, 5 ]
C = [ 2, 3, 4, 7, 5, 9, 6, 12, 11, 8 ]
D = [ 23, 8, 96, 6, 15, 33, 10 ]
E = [ 43, 9, 27, 8, 6, 19, 22, 4, 3, 5, 2, 10, 13, 18, 16 ]
```

A = [90, 15, 10, 7, 12, 2] max-heap
B = [6, 8, 15, 20, 10, 25, 33, 5] not a heap
C = [2, 3, 4, 7, 5, 9, 6, 12, 11, 8] min-heap
D = [23, 8, 96, 6, 15, 33, 10] not a heap
E = [43, 9, 27, 8, 6, 19, 22, 4, 3, 5, 2, 10, 13, 18, 16] max-heap

- 1 point each array

2.2 [7 points] Use C or pseudo-code to implement a function `isMaxHeap(int A[], int n)` to check if a given array `A[0..n-1]` of integers is a valid max-heap.

```
int lchild(int i) { return 2*i + 1;}
int rchild(int i) { return 2*i + 2;}

int isMaxHeapRecursive(int A[], int i, int n) {

  if (i>=n) // A NIL node is a max-heap
    return 1;
  else if (lchild(i) >= n && rchild(i) >= n) // A leaf node is a max-heap
    return 1;

  if ( (lchild(i) < n && A[i] < A[lchild(i)])
      || (rchild(i) < n && A[i] < A[rchild(i)]))
    return 0;

  return isMaxHeapRecursive(A, lchild(i), n)
       & isMaxHeapRecursive(A, rchild(i), n);
}

int isMaxHeapIterative(int A[], int n) {
    int i;

    for (i=0; i<n/2; i++)
    {
        if (lchild(i) < n && A[lchild(i)] > A[i])
            return 0;

        if (rchild(i) < n && A[rchild(i)] > A[i])
            return 0;
    }
    return 1;
}
```

Recursive solution:

- 1 point: lchild, rchild
- 2 point: base case for TRUE (leaf & NIL)
- 2 points: base case for FALSE (1 point for checking that the child exists, 1 point for checking values)
- 2 points: correct recursive calls

Iterative solution:

- 1 point: lchild, rchild (0.5 if 2i and 2i+1)
- 1 point: for loop (0.5/1 if $\leq$n/2 or n instead of n/2)
- 2 points: correct conditions FALSE (1 point for checking that the child exists, 1 point for checking values)
- 1.5 point: return FALSE
- 1.5 point: return TRUE

- If use heapify and the compare heap and original array (-1)
- If compare arrays without defining (-1)

2.3 [8 points] Given an array `A[0..n-1]` of integers of size `n`, use C or pseudo-code to write a function `findKthLargest(int A[], int n, int k)` that returns the `k-th` largest element in array `A`. Your function must use a `min-heap H` of size n to find the k-th element. You can use the function `minHeapify(H, i, n)`.

Build a min-heap H of all elements of A then remove $n-k$ times the root of the heap and heapify. At the end the k largest elements will remain in the heap and the k-th largest element will be in the root (A[0]).

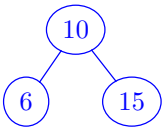
```
int findKthLargest(int A[], int n, int k) {
    int s, i;
    int H[n];

    for (i=0; i<n; i++)
        H[i] = A[i];

    s = n;
    for(i=n/2-1;i>=0;i--)
        minHeapify(H, i, s);

    for(i=0;i<n-k;i++) {
        H[0] = H[s-1];
        s--;
        minHeapify(H,0,s);
    }
    return H[0];
}
```

1. Copy values in H is not really necessary so I do not add or remove points for it.
2. 3 points: Build correctly the heap
3. 3 points: Correctly remove n-k elements
4. 2 points: Return H[0] at the end;

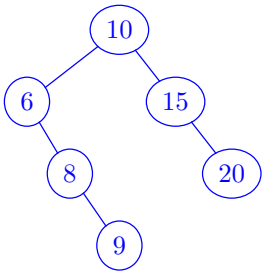
Exercise 3 20 Points

Binary Search Trees

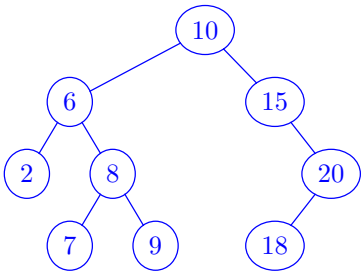
3.1 [5 points] The values **10 6 15 ; 8 9 20 ; 7 18 2 ;** are inserted in the given order into an empty `binary search tree`. Draw the binary search tree at the positions marked by a semicolon.



[1 point]



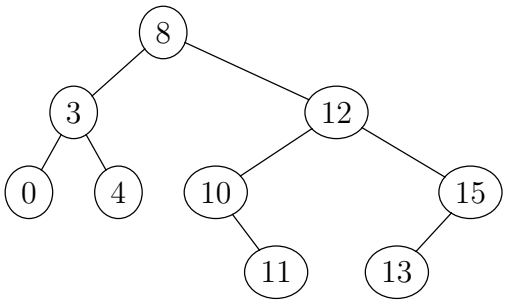
[2 points]

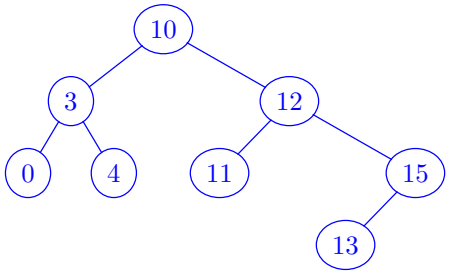


[2 points]

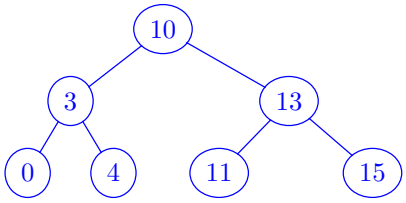
If values are inserted into 3 separated trees, only get maximum half points for that part.

3.2 [6 points] The values **8 ; 12 ;** are deleted in the given order from the below `binary search tree`. Draw the binary search tree at the positions marked by a semicolon.





[3 points]
candidate node is chosen, get 1/3 points.



[3 points]

If correct

3.3 [9 points] Given the BST definition below, write in C or pseudocode the function `printRange(struct TreeNode *p, int rangeMin, int rangeMax)` that prints in ascending order the value of nodes in a tree rooted at `p` within the range `[rangeMin..rangeMax]`. You can assume that `rangeMin` $<$ `rangeMax`.

```
struct TreeNode {
    int key;
    struct TreeNode* lChild;
    struct TreeNode* rChild;
};

struct TreeNode* root;
```

```
void printRange(struct TreeNode *p, int rangeMin, int rangeMax)
{
    /* base case */
    if ( NULL == p )
        return;

    if ( rangeMin <= p->data )
      printRange(p->left, rangeMin, rangeMax);

    if ( rangeMin <= p->data && rangeMax >= p->data )
      printf("%d ", p->data );

    if (p->data <= rangeMax )
      printRange(p->right, rangeMin, rangeMax);
}
```

- 1 point: base case
- 2 points: correct conditions to select left/right tree
- 2 points: correct recursion calls
- 1 points: correct print root→data
- 3 point: print in ascending order

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]

Informatik II
Spring 2017

Midterm1
31.03.2017

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.
  - No additional items are allowed. Notably 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:

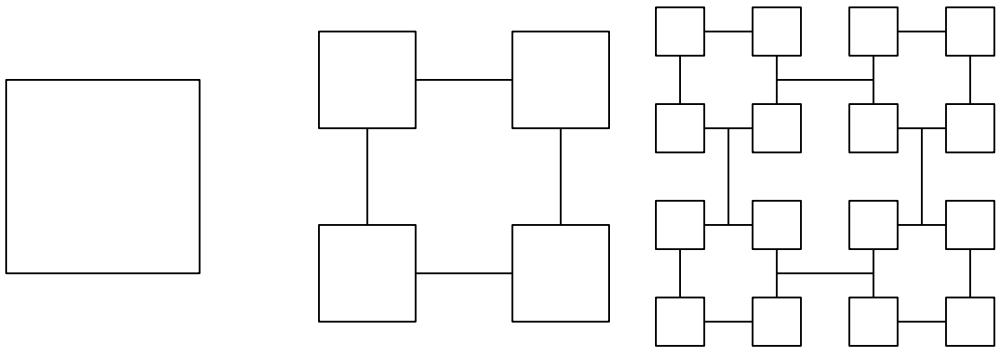
Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	Total
Points Achieved					
Maximum Points	15	17	15	13	60

Exercise 1 15 Points

Arrays and Algorithms

1.1 [5 points] Consider the star fractal depicted in Figure 1. Write a recursive function `drawStarFractal(x,y,l,i)` to draw this fractal. x and y represent the center coordinates of a square and l represents its side length. i is the recursion depth. Assume a function `square(x0, y0, l)` that draws a square with the point $(x0, y0)$ as left-lower corner and side length equal to l. Use either C or Pseudocode for your solution.



(a) i=1 (b) i=2 (c) i=3

Figure 1: Star Fractal

Algorithm: drawStarFractal(x,y,l,i)

```
1 if i > 0 then
2     square(x-l/2, y-l/2, l);
3     drawStarFractal(x-l/2, y-l/2, l/2, i-1);
4     drawStarFractal(x-l/2, y+l/2, l/2, i-1);
5     drawStarFractal(x+l/2, y-l/2, l/2, i-1);
6     drawStarFractal(x+l/2, y+l/2, l/2, i-1);
```

1.2 [10 points] Consider an unsorted array of integers $A[n]$. An element $A[i]$ is called a **leader** if it is greater than all elements $A[j]$ for $i < j \leq n$.

i. [2 points] Specify all the leaders of the following array: $[34, 81, 77, 61, 75, 19, 72]$
Leaders: $81, 77, 75, 72$

ii. [8 points] Implement an algorithm `printLeaders(A, n)`, with asymptotic complexity $O(n)$ that prints all the leaders of a given array A. Use either C or Pseudocode for your solution.
Note: Solutions with greater complexity (e.g., $O(n^2)$) will also be accepted but they will get 4 points at most.

Algorithm: printLeadersSimple(A,n)

```
/* Complexity:  O(n^2)                                    */
1 for i = 1 to n do
2     j := i + 1;
3     while j ≤ n and A[i] ≤ A[j] do
4         j++;
5     if j = n then
6         print(A[i]);
```

Algorithm: printLeaders(A,n)

```
/* Complexity:  O(n)                                      */
1 leftmostLeader := A[n];
2 print(leftmostLeader);
3 for i = n − 1 to 1 do
4     if leftmostLeader < A[i] then
5         leftmostLeader := A[i];
6         print(leftmostLeader);
```

Name: Matriculation number:

Exercise 2 17 Points

Asymptotic Complexity and Special Case Analysis

2.1 [6 points] Calculate the simplest possible asymptotic tight bound for the following functions. Clearly work out the calculation steps in your solution:

$$f_1(n) = \sqrt{343n^3} + n \log_2 n^9$$
$$f_2(n) = 2^{16 \log_2 n^2}$$
$$f_3(n) = 22n^{\log_2 \sqrt{2}} + \sqrt{n \log_2 n}$$
$$f_4(n) = 30n^{\max(\log_2 n^4, \sqrt{n})}$$
$$f_5(n) = \log_2(2 \log_2 4^n)$$
$$f_6(n) = 18n^{24} + 4096^{108} + 3^{2n}$$

- $f_1(n) = \sqrt{343n^3} + n \log_2 n^9 = \sqrt{343} n^{\frac{3}{2}} + 9n \log_2 n \in \Theta(\sqrt{n^3})$
- $f_2(n) = 2^{16 \log_2 n^2} = 2^{32 \log_2 n} = (2^{\log_2 n})^{32} = n^{32} \in \Theta(n^{32})$
- $f_3(n) = 22n^{\log_2 \sqrt{2}} + \sqrt{n \log_2 n} = 22n^{\frac{1}{2} log_2 2} + \sqrt{n \log_2 n} = 22\sqrt{n} + \sqrt{n \log_2 n} \in \Theta(\sqrt{n \log n})$
- $f_4(n) = 30n^{\max(\log_2 n^4, \sqrt{n})} \in \Theta(n^{\sqrt{n}})$
- $f_5(n) = \log_2(2 \log_2 4^n) = \log_2 2 + \log_2 n + \log_2 \log_2 4 \in \Theta(\log n)$
- $f_6(n) = 18n^{24} + 4096^{108} + 3^{2n} \in \Theta(9^n)$

2.2 [7 points] The daily schedule of bus drivers is defined by a pair of integers (s,e) where s represents the hour of the day that the driver starts working and e represents the hour that the driver's shift ends. Assume you are given the time schedules (s_A, e_A) and (s_B, e_B) of drivers A and B respectively. Write an algorithm that computes the number of hours when both drivers are working. For example, if bus driver A works from 8:00 to 16:00 ($s_A = 8$, $e_A = 16$) and driver B works from 12:00 to 20:00 ($s_B = 12$, $e_B = 20$), their schedules have an overlap of 4 hours.

a) [4 points] Specify all special cases for s_a, e_a, s_b, and e_b that must be considered. For each case, provide an example input as well as the result that the algorithm will return.

#	Case	s_a	e_a	s_b	e_b	result
1	$s_A \geq e_B$	14	16	13	14	0
2	$s_B \geq e_A$	13	14	14	16	0
3	$s_A \leq s_B \wedge e_A > s_B \wedge e_A < e_B$	10	15	12	17	15-12=3
4	$s_B \leq s_A \wedge e_B > s_A \wedge e_B < e_A$	12	17	10	15	15-12=3
5	$s_B > s_A \wedge e_B \leq e_A$	12	20	15	17	17-15=2
6	$s_A > s_B \wedge e_B \geq e_A$	15	17	12	20	17-15=2
7	$s_A = s_B \wedge e_B = e_A$	12	20	12	20	20-12=8
8	$s_A \notin [0,24] \vee s_B \notin [0,24] \vee$ $e_A \notin [0,24] \vee e_B \notin [0,24]$	12 -12	25 20	-1 1	17 17	-1
9	$s_A > e_A \vee s_B > e_B$	20	10	15	17	-1
10	$s_A = \omega \vee s_B = \omega \vee$ $e_A = \omega \vee e_B = \omega$	- -	25 25	- 1	17 -	-1

b) [3 points] Implement the algorithm `overlappingHours(`s_A`, `e_A`, `s_B`, `e_B`)` that receives the starting and ending hours of the working schedules of two drivers and computes the number of overlapping hours between these schedules.

Algorithm: overlappingHours(s_A, e_A, s_B, e_B)

1 **if** $e_A = \omega \vee e_B = \omega \vee e_A = \omega \vee e_B = \omega$ **then return** -1;
2 **if** $s_A < 0 \vee s_B < 0 \vee e_A < 0 \vee e_B < 0$ **then return** -1;
3 **if** $s_A > 24 \vee s_B > 24 \vee e_A > 24 \vee e_B > 24$ **then return** -1;
4 **if** $s_A > e_A \vee s_B > e_B$ **then return** -1;

5 **if** $s_A \geq e_B \vee s_B \geq e_A$ **then return** 0;

6 **if** $s_A \leq s_B \wedge e_A > s_B \wedge e_A < e_B$ **then return** $e_A - s_B$;
7 **if** $s_B \leq s_A \wedge e_B > s_A \wedge e_B < e_A$ **then return** $e_B - s_A$;
8 **if** $s_B > s_A \wedge e_B \leq e_A$ **then return** $e_B - s_B$;
9 **if** $s_A > s_B \wedge e_B \geq e_A$ **then return** $e_A - s_A$;
10 **if** $s_A = s_B \wedge e_B = e_A$ **then return** $e_A - s_A$;
11 **return** *0*

Algorithm: overlappingHours(s_A, e_A, s_B, e_B)

1 **if** $e_A = \omega \vee e_B = \omega \vee e_A = \omega \vee e_B = \omega$ **then return** -1;
2 **if** $s_A < 0 \vee s_B < 0 \vee e_A < 0 \vee e_B < 0$ **then return** -1;
3 **if** $s_A > 24 \vee s_B > 24 \vee e_A > 24 \vee e_B > 24$ **then return** -1;
4 **if** $s_A > e_A \vee s_B > e_B$ **then return** -1;

5 **if** $s_A \geq e_B \vee s_B \geq e_A$ **then return** 0;

6 **if** $s_A > s_B$ **then** $maxS = s_A$;
7 **else** $maxS = s_B$;
8 **if** $e_A > e_B$ **then** $minE = e_B$;
9 **else** $minE = e_A$;
10 **return** $minE - maxS$

2.3 [4 points] You are given the bubble sort algorithm.

Algorithm: bubbleSort(A,n)

```
1 for i = n to 2 do
2     for j = 2 to i do
3         if A[j] < A[j − 1] then
4             t = A[j];
5             A[j] = A[j − 1];
6             A[j − 1] = t;
```

a) Formulate the loop invariant for the outer loop of the bubble sort algorithm, given above.

At the beginning of the outer loop the subarray $A[i+1..n]$ is in sorted order and consists of the $n-i$ biggest elements of array $A[1..n]$

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

$$\forall j \in [i+1, n], \forall k \in [1, i] : A[j] \geq A[k]$$

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

b) What happens if we change the outer loop (line 1) into the following: **for** $i = n$ **to** 1 **do**? Will the algorithm still produce correct results? Justify your answer.

The algorithm still produces correct results. After the change i gets first from n to 2 and the table is sorted. When i gets 1 the inner loop **for** $j = 2$ **to** i **do** is not executed because $2 > 1$.

Name: Matriculation number:

Exercise 3 15 Points

Runtime and Recursion

Algorithm: alg(A,n)

```
1  for i = 1 to n do
2      if i is odd then
3          j := 3;
4          while j ≤ n do
5              if A[j] < A[j − 1] then
6                  swap(A[j], A[j − 1]);
7              j := j+2;
8      else
9          j := 2;
10         while j ≤ n do
11             if A[j] < A[j − 1] then
12                 swap(A[j], A[j − 1]);
13             j := j+2;
```

3.1 [2 points] The algorithm `alg(A,n)` gets as input an array `A[1...n]` and the number `n` of integers it contains. Assume A=`[9, 1, 3, 6, 2, 4, 1, 8]`. Complete the following matrix where you need to show the value of i and the content of A after each execution of the outer for loop. The first line of the matrix shows the initial array A before the execution of the loop.

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

3.2 [7 points] Analyze the steps of the algorithm `alg(A,n)`, calculate its exact runtime and its asymptotic complexity. Assume that the array $A[n]$ is of even length.

line 1	c_1	$n+1$
line 2	c_2	n
line 3	c_3	$\frac{n}{2}$
line 4	c_4	$\frac{n}{2}\left(\frac{n}{2}\right)$
line 5	c_5	$\frac{n}{2}\left(\frac{n}{2}-1\right)^{*}$
line 6	c_6	$\alpha^{***}\frac{n}{2}\left(\frac{n}{2}-1\right)$
line 7	c_7	$\frac{n}{2}\left(\frac{n}{2}-1\right)$
line 8	c_8	n
line 9	c_9	$\frac{n}{2}$
line 10	c_{10}	$\frac{n}{2}\left(\frac{n}{2}+1\right)$
line 11	c_{11}	$^{**}(\frac{n}{2})^2$
line 12	c_{12}	$\beta^{***}(\frac{n}{2})^2$
line 13	c_{13}	$(\frac{n}{2})^2$

* $\frac{n}{2}-1$ times for j from 3 to n, and $\frac{n}{2}$ times that i is odd.
** $\frac{n}{2}$ times for j from 2 to n, and $\frac{n}{2}$ times that i is even.
*** $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$

Running time:

$T(n) = c_1 + (c_1 + c_2 + c_8)n + \frac{n}{2}(c_3 - c_5 - \alpha c_6 - c_7 + c_9 + c_{10}) + (\frac{n}{2})^2(c_4 + c_5 +$

$\alpha c_6 + c_7 + c_{10} + +c_{11} + \beta c_{12} + c_{13}$
$\in O(n^2)$

3.3 [6 points] Give a recursive version of the bubble sort algorithm `bubbleSortRec(A,n)` that sorts an array A of n integers in ascending order. Use either C or pseudocode for your solution.

Algorithm: bubbleSortRec(A,n)

```
1 for j = 2 to n do
2     if A[j] < A[j − 1] then
3         t = A[j];
4         A[j] = A[j − 1];
5         A[j − 1] = t;
6 if n > 1 then bubbleSortRec(A, n − 1) ;
```

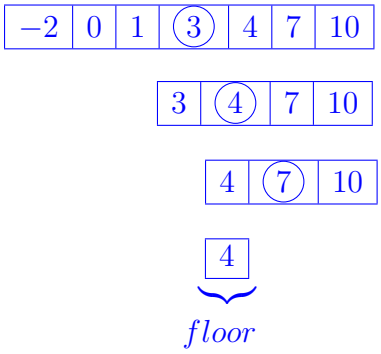
Exercise 4 13 Points

Divide and Conquer

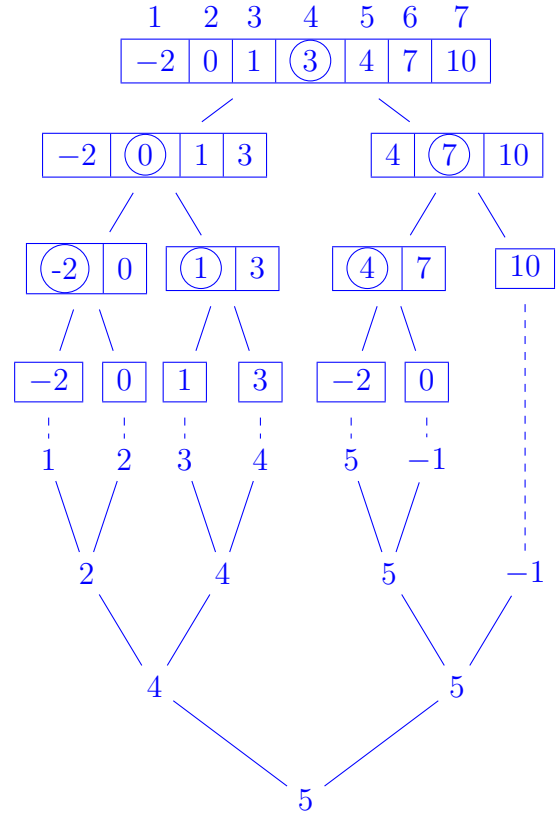
The floor of an integer x in an array A is the largest element $a \in A$ such that $a \leq x$. Implement a divide and conquer algorithm that, given a value x and an array $A[1 \ldots n]$ of integers sorted in **ascending order**, finds the floor of x and returns its position in A.

4.1 [3 points] Draw a tree to illustrate the process of finding the floor of 5 in array `A = [-2, 0, 1, 3, 4, 7, 10]`, according to your divide and conquer algorithm.

Solution 1:



Solution 2:


1 2 3 4 5 6 7
−2 0 1 3 4 7 10
−2 0 1 3
4 7 10
-2 0
1 3
4 7
10
−2 0 1 3 −2 0
1 2 3 4 5 −1
2 4 5 −1
4 5
5

4.2 [10 points] Implement a **divide and conquer** algorithm that finds the floor of a given number x in a given sorted array A and returns the position of the floor in A. Use either C or pseudocode for your solution.

Solution 1:

```
int floorSearch(int arr[], int l, int r, int x)
{
  int m = (l+r)/2;

  if (r−l≤1) {
    if (arr[r] ≤x) return r;
    else if (arr[l] ≤x) return l;
    else return −1;
  }

  if (arr[m] > x) r = m−1;
  else l=m;
  return floorSearch(arr, l, r, x);
}
```

Solution 2:

```
int floorSearch2(int arr[], int l, int r, int x)
{
  int m, m1, m2;

  if (l==r) return l;
  else {
    m = (l+r)/2;
    m1 = floorSearch2(arr, l, m, x);
    m2 = floorSearch2(arr, m+1, r, x);

    if (m2 ≥0 && arr[m2] ≤x) return m2;
    else if (m1 ≥0 && arr[m1] ≤x) return m1;
    else return −1;
  }
}
```

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]

Informatik II
Spring 2017

Midterm2
28.04.2017

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 (15 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. Notably 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	15	15	15	15	60

Exercise 1 — 15 Points

1.1 [8 points] Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and justify which case (1-3) applies.

(a) $T(n) = 81T(\frac{n}{3}) + 2n^3 + 7n$, $T(1) = 0$

- $a = 81$, $b = 3$, $f(n) = 2n^3 + 7n$
- $n^{\log_3 81} = n^4$
- $f(n) = 2n^3 + 7n = O(n^{\log_3 81 - 0.5})$
- Case 1: $T(n) = \Theta(n^{\log_3 81})$

(b) $T(n) = 3T(\frac{n}{9}) + n^2$, $T(1) = 0$

- $a = 3$, $b = 9$, $f(n) = n^2$
- $n^{\log_9 3} = n^{0.5}$
- $f(n) = n^2 = \Omega(n^{\log_9 3 + 1})$
- Case 3: $T(n) = \Theta(n^2)$
- Regularity condition: $3\left(\frac{n}{9}\right)^2 \leq cn^2$, for $\frac{3}{81} \leq c < 1$

(c) $T(n) = 2T(n-3) + 5$, $T(0) = 5$ Repeated substitution:

$$\begin{aligned} T(n) &= 2T(n-3) + 5 \\ &= 2(2T(n-6) + 5) + 5 = 2^2T(n-6) + (1+2)5 \\ &= 2^2(T(n-9) + 5) + (1+2)5 = 2^3T(n-9) + (1+2+2^2)5 \\ &= \dots \end{aligned}$$

$$\Rightarrow T(n) = 2^kT(n-3k) + 5\sum_{i=0}^{k-1} 2^i$$

k grows until it reaches $n - 3k = 0 \Rightarrow k = \frac{n}{3}$, thus we have:

$T(n) = 2^{\frac{n}{3}}5 + 5\sum_{i=0}^{\frac{n}{3}-1} 2^i = 5\sum_{i=0}^{\frac{n}{3}} 2^i = 5(2^{\frac{n}{3}+1} - 1) = \Theta(2^{\frac{n}{3}})$

(d) $T(n) = 64T(\frac{n}{4}) + 32n^3 + 7$, $T(1) = 0$

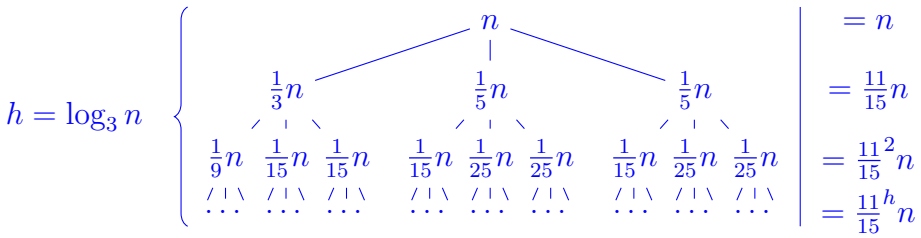
- $a = 64$, $b = 4$, $f(n) = 32n^3 + 7$
- $n^{\log_4 64} = n^3$
- $f(n) = 32n^3 + 7 = \Theta(n^{\log_4 64})$
- Case 2: $T(n) = \Theta(n^3 \log_4 n)$

Name: Matriculation number:

1.2 [7 points] Consider the following recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/3) + 2T(n/5) + n & \text{if } n > 1 \end{cases}$$

(a) Draw the corresponding recursion tree and use it to calculate an estimate for the asymptotic upper bound of $T(n)$. Include the tree-based calculations that left to your estimate.



Longest branch on the left. Tree grows until $\left(\frac{1}{3}\right)^h n = 1 \implies h = \log_3 n$

Guess: $T(n) = n + \frac{11}{15}n + \frac{11}{15}^2 n + \cdots + \frac{11}{15}^{\log_3 n} n = n\Sigma_{i=0}^{\log_3 n}(\frac{11}{15})^i < n\Sigma_{i=0}^{\infty}\frac{11}{15} = cn \in O(n)$

$T(n) < n\Sigma_{i=0}^{\infty}(\frac{11}{15})^i = n\frac{1}{1-\frac{11}{15}} = O(n)$

(b) Prove the correctness of your estimate using induction.

Proof by induction:

$$\begin{aligned} T(n) &= T(\frac{n}{3}) + 2T(\frac{n}{5}) + n && \text{(recurrence)} \\ &\leq c\frac{n}{3} + 2c\frac{n}{5} + n && \text{(inductive hyp.)} \\ &= cn\left(\frac{1}{3} + \frac{2}{5}\right) + n && \text{(rewriting)} \\ &= cn - \left(\frac{4}{15}c - 1\right)n && \text{(rewriting)} \\ &\leq cn && \text{for } n \geq 1, c > \frac{15}{4} \end{aligned}$$

Exercise 2 — 15 Points

2.1 [2 points] Assume a string composed of capital letters and the asterisk character (*). An algorithm scans the string and, when it finds a capital letter, it pushes it on stack `S` (initially empty). When it finds an asterisk, it pops a letter from `S` and prints the popped letter. Write the sequence of letters printed by this algorithm if the following string is given as input.
B K A * G * * E D * F * * * N M *
A G K D F E B M

2.2 [5 points] Consider a linked list defined as follows:

```
1   struct node {
2     int key;
3     struct node *next;
4   };
5
6   struct list {struct node *head};
```

Use C or pseudocode to describe an algorithm `deleleteLast(listA)` that deletes the last node of list `listA`.

```
1  void deleteLast(struct list* listA) {
2    struct node* toDelete;
3    struct node* prev;
4    int i=0;
5
6    if (listA->head != NULL) {
7      toDelete = listA->head;
8      while(toDelete->next!=NULL) {
9        prev = toDelete;
10       toDelete = toDelete->next;
11     }
12
13     if (toDelete == listA->head) listA->head = NULL;
14     else prev->next = NULL;
15     free(toDelete);
16   }
17 }
```

2.3 [8 points] Consider a queue `Q` defined as follows:

```
1   struct queue {
2     int A[6];
3     int head;
4     int tail;
5   };
6   struct queue *Q;
```

The queue is implemented by the array `A`, traversed in circular fashion. Variables `head` and `tail` function as pointers to the head and the tail of the queue. Elements are dequeued at the head and enqueued at the tail. The algorithm `search(Q, val)` returns the number of dequeue operations needed until we dequeue the element `val` from `Q`. If `val` is not an element of the queue, the algorithm returns -1.

(a) For each of the following `Q` and `val` combinations, give the result returned by `search(Q, val)`.

Q								val	search(Q, val)
head	tail	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
1	4	1	9	0	4	3	6	0	2

Q								val	search(Q, val)
head	tail	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
1	4	3	12	1	9	10	7	3	-1

Q								val	search(Q, val)
head	tail	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
4	1	11	4	8	9	2	15	11	3

Q								val	search(Q, val)
head	tail	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]		
2	0	9	3	15	9	2	14	14	4

(b) Use either C or pseudocode to describe the algorithm `search(Q, val)`.

```
1  int search(struct queue* q, int val) {
2    int curr = q->head;
3    int res = 1;
4
5    while (curr != q->tail) {
6      if (q->arr[curr] == val)
7        return res;
8      res++;
9      curr = (curr+1) % 6;
10   }
11   return -1;
12 }
```

Name: Matriculation number:

Exercise 3 — 15 Points

3.1 [2 points] Assume a heap $A[1...n]$ and a node index `i`. Use either C or pseudocode to describe the functions:

(a) `parent(i)`, that returns the index of `i`'s parent.

(b) `sibling(i)`, that returns the index of `i`'s sibling.

```
1 Algorithm: PARENT(i)
2 if i ≤ 1 ∨ i > n then
3     return −1
4 return ⌊i/2⌋
```

```
1 Algorithm: SIBLING(i)
2 if i ≤ 1 ∨ i > n then
3     return −1
4 if i % 2 = 1 then return
    i − 1;
5 if i = n then return −1;
6 return i + 1
```

3.2 [3 points] Consider the following array of integers `A=[ 4 9 2 13 8 ]`. Apply the algorithm `buildMaxHeap` on `A` and illustrate the status of `A` at the end of each iteration of the algorithm.

i	**A[1]**	**A[2]**	**A[3]**	**A[4]**	**A[5]**
-	4	9	2	13	8
2	4	13	2	9	8
1	13	9	2	4	8

3.3 [4 points] Consider a max heap `h` defined as follows:

```
1  struct maxHeap {
2    int A[10];
3    int s;
4  };
5  struct maxHeap h;
```

The size `s` of the max heap indicates that the max heap includes the first `s` elements of A. The remaining $10 - s$ elements of `A` are ignored. Use C or pseudocode to describe an algorithm `deleteMax(h)` that removes and returns the maximum element of max heap `h`. After removing the maximum element, `h` must be modified so that it still represents a max-heap.

Note: Consider the algorithm `heapifyMax(A, i, n)` as known.

```
1  int deleteMax(struct maxHeap *h) {
2    int max = h->A[0];
3    h->A[0] = h->A[h->s-1];
4    h->s--;
5    heapifyMax(h->A, 0, h->s);
6    return max;
7  }
```

3.4 [6 points] Consider the max heap `h` of task 3.3 and an integer `val`. Use C or pseudocode to describe an algorithm `increaseValue(h, i, val)` that sets `h.A[i]` to `val` if `val` > `h.A[i]`. After the assignment, `h` must be adapted so that it is a max heap again. If `val` $\leq$ `h.A[i]`, an error message must be printed.

Note: You can not use the algorithm `heapifyMax(A, l, r)`.

```
1  void increaseValue(struct maxHeap *h, int i, int val) {
2    if (i≥h->s || i<0 || h->A[i] > val) printf("error\n");
3    else {
4      h->A[i] = val;
5      while (i>0 && h->A[parent(i)] < h->A[i]) {
6        swap(h->A, parent(i), i);
7        i = parent(i);
8      }
9    }
10 }
```

Name: Matriculation number:

Exercise 4 — 15 Points

4.1 [4 points] Assume an array `A = [ 4, 6, 1, 9, 0, 3, 7, 5 ]`. Demonstrate Lomuto partitioning with $l = 1$ and $r = 8$ by completing the following matrix. The first line of the matrix shows the initial array `A` before the execution of the loop.

i	j	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
0	-	4	6	1	9	0	3	7	5
1	1	4	6	1	9	0	3	7	5
1	2	4	6	1	9	0	3	7	5
2	3	4	1	6	9	0	3	7	5
2	4	4	1	6	9	0	3	7	5
3	5	4	1	0	9	6	3	7	5
4	6	4	1	0	3	6	9	7	5
4	7	4	1	0	3	6	9	7	5
4	7	4	1	0	3	5	9	7	6

4.2 [2 points] At the end of Lomuto partitioning in an array `A[0...n-1]`, the initially rightmost element of `A` is moved to a new position `i` in `A`. If we sorted array `A`, in which position would this element be? Justify your answer.

If array `A` is sorted the initially rightmost element would be in position `i`. Lomuto partitioning uses the rightmost element of `A` as pivot. The pivot is the element that is used to partition the array in quicksort and similar algorithms. The array is arranged so that all elements smaller than the pivot are to the left of it and all elements greater than the pivot are to the right of it. This means that the pivot is always in its final sorted position after partitioning. In case the array has duplicates, the pivot element may be swaped with another element if they have the same value. But this case cannot be noticed by the final result of the sorted array.

4.3 [5 points] The kth-smallest element of an array of `n` integers `A[1...n]` is the element that would be at position `k` if the array `A` was sorted. For example, if `A = [ 4, 6, 1, 9, 0, 2, 7, 5 ]`, the 3^{rd} smallest element of `A` is 2.

Use C or pseudocode to describe the algorithm `selectKthSmallest(A, n, k)` that uses `LomutoPartition(A, left, right)` to find the kth-smallest element in an array `A` of `n` integers, without sorting it.

```
1  int selectKthSmallest(int A[], int n, int k) {
2    int pivot = 0, left = 0, right = n−1;
3
4    do {
5      pivot = lomuto(A, left, right);
6      if (k < pivot) right = pivot−1;
7      else left = pivot+1;
8    } while(pivot != k);
9
10   return A[k];
11 }
```

4.4 [4 points] Use C or pseudocode to describe a variation of Hoare partitioning, that uses the leftmost element to partition the array. Formulate the loop invariant for the outer loop of this algorithm.

```
1  int hoare(int A[], int l, int r) {
2    int x = A[l], i = l−1, j = r+1;
3
4    while (1) {
5      do {
6        j−−;
7      } while (A[j]>x);
8      do {
9        i++;
10     } while (A[i]<x);
11     if (i<j) swap(A, i, j);
12     else {
13       return j;
14     }
15   }
16 }
```

Loop invariant:

$$\forall q \in [l, i-1] : A[q] \leq x$$

$$\forall r \in [j+1, r] : A[r] \geq x$$

where $A[l]$ is assigned to x before the outer loops be executed.

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]

Informatik II
Spring 2018

Midterm1
23.03.2018

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 (13 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	15	18	13	14	60

Exercise 1 15 Points

Arrays and Sorting

1.1 [8 points] Consider an array of `n` integers `A[0..n-1]`, which contains all integers between 1 and `n+1` except one.

1. Assume the array is **sorted**. Find and describe an algorithm with sublinear (i.e., less than linear) asymptotic complexity that determines the missing number. Determine the asymptotic worst-case complexity of your algorithm.
2. Assume the array is **not sorted**. Find and describe an algorithm with linear asymptotic complexity that determines the missing number.
   Hint: You can use an auxiliary array.

You do **not** need to implement the algorithm.

1. We use binary search for a sorted array:
   - We look for the smallest index i for which $A[i] = i + 2$; this is the position of our missing number.
   - We find the `mid` element, $mid = \lfloor \frac{n}{2} \rfloor$. If $A[mid] = mid + 1$ we recurse on the second half of the array $A[mid+1..high]$, otherwise we recurse on first half of the array $A[low..mid]$,
   - The algorithm terminates when the search range consists of only one element (i.e. $A[low] == A[high]$) and $A[i] = i + 2$. The missing number is $i + 1$.
   - We get the recurrence $T(n) = T(\frac{n}{2}) + \Theta(1)$, which gives a worst-case running time of $\Theta(\log n)$
2. We use the auxiliary array to keep track of the numbers we have seen: $A[i] = 0$ if we have not seen i; $A[i] = 1$ if we have seen i.
   - Create auxiliary array B with indices 1 to $n + 1$
   - Initialize all locations to 0
   - Go through all elements of A and set $B[A[i]] = 1$
   - Go through all elements of B and return the i for which $B[i] = 0$. This is the missing number.

Name: Matriculation number:

1.2 [7 points] Consider an array of `n` integers `A[0..n-1]`. The array is sorted in **ascending order** and there are no duplicate elements. Assume the array has been **rotated** i.e., the elements have been shifted multiple times to the right with the last element being moved to the first position in each step. For instance, `A = [10, 20, 25, 40, 52]` becomes `A = [25, 40, 52, 10, 20]` after its elements were shifted to the right three times. Implement an algorithm that finds the position of an element in a rotated array. The asymptotic running time complexity of your algorithm must be $O(\log n)$. Use either C or pseudocode for your solution.

```
int search(int arr[], int low, int high, int key) {
    if (low > high) return −1;

    int mid = (low + high) / 2;
    if (arr[mid] == key) return mid;

    if (arr[low] ≤arr[mid]) {
        if (key ≥arr[low] && key ≤arr[mid]) {
            return search(arr, low, mid−1, key);
        }
        return search(arr, mid+1, high, key);
    }

    if (key ≥arr[mid] && key ≤arr[high]) {
        return search(arr, mid+1, high, key);
    }
    return search(arr, low, mid−1, key);
```

Name: Matriculation number:

Exercise 2 18 Points

Asymptotic Complexity

2.1 [6 points] Calculate the simplest possible asymptotic tight bound for the following functions. Include any calculation steps in your solution:

a) $f_1(n) = \log(n * \log(n))$
$f_1(n) = \log(n * \log(n)) = \log(n) + \log\log(n) \in \Theta(\log(n))$

b) $f_2(n) = \log(\sum_{i=1}^{n} i)$
$f_2(n) = \log(\sum_{i=1}^{n} i) = \log(\frac{n*(n+1)}{2}) = \log(n * (n+1)) - \log(2) = \log(n) + \log(n+1) - \log(2) \in \Theta(\log n)$

c) $f_3(n) = 2^{16 \log_2 n^2}$
$f_3(n) = 2^{16 \log_2 n^2} = 2^{32 \log_2 n} = (2^{\log_2 n})^{32} = n^{32} \in \Theta(n^{32})$

d) $f_4(n) = 22n^{\log_2 \sqrt{2}} + \sqrt{n \log_2 n}$
$f_4(n) = 22n^{\log_2 \sqrt{2}} + \sqrt{n \log_2 n} = 22n^{\frac{1}{2} log_2 2} + \sqrt{n \log_2 n} = 22\sqrt{n} + \sqrt{n \log_2 n} \in \Theta(\sqrt{n \log n})$

e) $f_5(n) = 3\sqrt{n} + 8^{127} + \log 9^n$
$f_5(n) = 3\sqrt{n} + 8^{127} + \log 9^n = 3\sqrt{n} + 8^{127} + n \log 9 \in \Theta(n)$

f) $f_6(n) = 3 \log n^9 + 4^{2n} + n^3$
$f_6(n) = 3 \log n^9 + 4^{2n} + n^3 = 27 \log n + 16^n + n^3 \in \Theta(16^n)$

2.2 [6 points] Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used, write down a, b, $f(n)$ and the case (1-3).

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

$a = 3$, $b = 2$, $f(n) = n^2$, $n^{log_b a} = n^{log_3 2} = n^{1.58}$
$f(n) = \Omega(n^{log_b a + \epsilon}), \epsilon > 0$ Case 3: $\Theta(n^2)$
Regularity condition: if $c = \frac{3}{4}, a \cdot f(n/b) = 3 \cdot (\frac{n}{2})^2 \leq \frac{3}{4} \cdot n^2$
$T(n) = \Theta(n^2)$

b) $T(n) = 4T(\frac{n}{9}) + 7\sqrt{n}$

$a = 4$, $b = 9$, $f(n) = 7\sqrt{n} = 7n^{0.5}$, $n^{log_a b} = n^{log_4 9} = n^{0.63}$
$f(n) = O(n^{log_a b - \epsilon}), \epsilon = 0.13 > 0$ Case 1: $\Theta(n^{\log_9 4})$

c) $T(n) = 9T(\frac{n}{4}) + n^3$

$a = 9$, $b = 4$, $f(n) = n^3$, $n^{log_b a} = n^{log_9 4} = n^{0.63}$
$f(n) = \Omega(n^{log_b a + \epsilon}), \epsilon > 0$ Case 3: $\Theta(n^3)$
Regularity condition: if $c = \frac{9}{64}, a \cdot f(n/b) = 9 \cdot (\frac{n}{4})^3 \leq \frac{9}{64} \cdot n^3$
$T(n) = \Theta(n^3)$

d) $T(n) = \log n + T(\sqrt{n})$

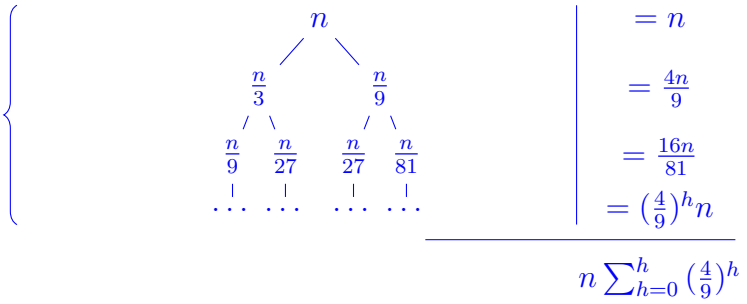
$$\begin{aligned} T(n) &= \log n + T(\sqrt{n}) = \log n + \log \sqrt{n} + T(\sqrt{\sqrt{n}}) \\ &= \log n + \log \sqrt{n} + \log \sqrt{\sqrt{n}} + T(\sqrt{\sqrt{\sqrt{n}}}) + ... \\ &= \log n + \frac{1}{2}\log n + \frac{1}{4}\log n + \frac{1}{8}\log n + ... \\ &= (1 + \frac{1}{2} + \frac{1}{4} +) \cdot \log n \\ &= \sum_{i=0}^{+\infty} (\frac{1}{2})^i \cdot \log n \\ &= 2 \cdot \log n = \Theta(\log n) \end{aligned}$$

2.3 [6 points] Consider the recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/3) + T(n/9) + n & \text{if } n > 1 \end{cases}$$

a) [3 points] Draw a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Include the tree-based calculations that led to your estimate.

$h = \log_3 n$



Longest branch on the left. Tree grows until $\left(\frac{1}{3}\right)^h n = 1 \implies h = \log_3 n$
To get an upper bound, we can use the sum $n \sum_{h=0}^{h} (\frac{4}{9})^h$.
Guess: $O(n)$

b) [3 points] Prove the correctness of your estimate using the substitution method.

Proof by induction:

$$\begin{aligned}
T(n) &= T(n/3) + T(n/9) + n && \text{(recurrence)} \\
&\leq c\frac{n}{3} + c\frac{n}{9} + n && \text{(inductive hyp.)} \\
&= (\frac{4c+9}{9})n && \text{(rewriting)} \\
&= (c - \frac{5c}{9} + \frac{9}{9})n && \text{(rewriting)} \\
&= cn - (\frac{5c-9}{9})n && \text{(rewriting)} \\
&\leq cn && \text{(for } c \geq \frac{9}{5}\text{)}
\end{aligned}$$

Exercise 3 13 Points

Runtime Analysis

Algorithm: whatDoesItDo(A, n)

```
1  tmp = 0;
2  do
3      ready = 0;
4      for i = n-1 to 1 do
5          if A[i-1]>A[i] then
6              tmp = A[i-1];
7              A[i-1] = A[i];
8              A[i] = tmp;
9              ready =1;
10     for i = 1 to n-1 do
11         if A[i-1]>A[i] then
12             tmp = A[i-1];
13             A[i-1] = A[i];
14             A[i] = tmp;
15             ready =1;
16 while ready=1;
```

3.1 [3 points] The algorithm `whatDoesItDo(A,n)` gets an array `A[0...n-1]` of n integers as an input. Apply the algorithm on the array A=`[8, 5, 10, 1, 3]` and complete each line of the following matrix if the content of A is modified. The already-completed line of the matrix shows the initial content of A.

i	**A[0]**	**A[1]**	**A[2]**	**A[3]**	**A[4]**
-	8	5	10	1	3
3	8	5	1	10	3
2	8	1	5	10	3
1	1	8	5	10	3
2	1	5	8	10	3
4	1	5	8	3	10
3	1	5	3	8	10
2	1	3	5	8	10

3.2 [4 points] Describe in words what the algorithm `whatDoesItDo(A,n)` does (maximum three lines). What is the best and worst case for this algorithm, and what are the asymptotic complexities in these cases.

The algorithm sorts the array A, it is a variation of bubble sort, this variation sorts A in both directions on each pass through the list.

Best case: O(n), array already sorted in ascending order.
Worst case: O(n^2), array sorted in descending order.

3.3 [6 points] Analyze the steps of the algorithm `whatDoesItDo(A,n)` and calculate its running time (worst case).

line 1	c_1	1
line 2	c_2	1
line 3	c_3	$\frac{n}{2}$
line 4	c_4	$\frac{n}{2}(n-1)$
line 5-9	c_5	$\frac{n}{2}(n-2)$
line 10	c_6	$\frac{n}{2}(n-1)$
line 11-15	c_7	$\frac{n}{2}(n-1)$
line 16	c_8	$\frac{n}{2}$

Running time: $c_1 + c_2 + c_3 \cdot \frac{n}{2} + c_4 \cdot \frac{n}{2} \cdot (n-1) + 4 \cdot c_5 \cdot \frac{n}{2} \cdot (n-2) + c_6 \cdot \frac{n}{2} \cdot (n-1) + 4 \cdot c_7 \cdot \frac{n}{2} \cdot (n-1) + c_8 \cdot \frac{n}{2}$

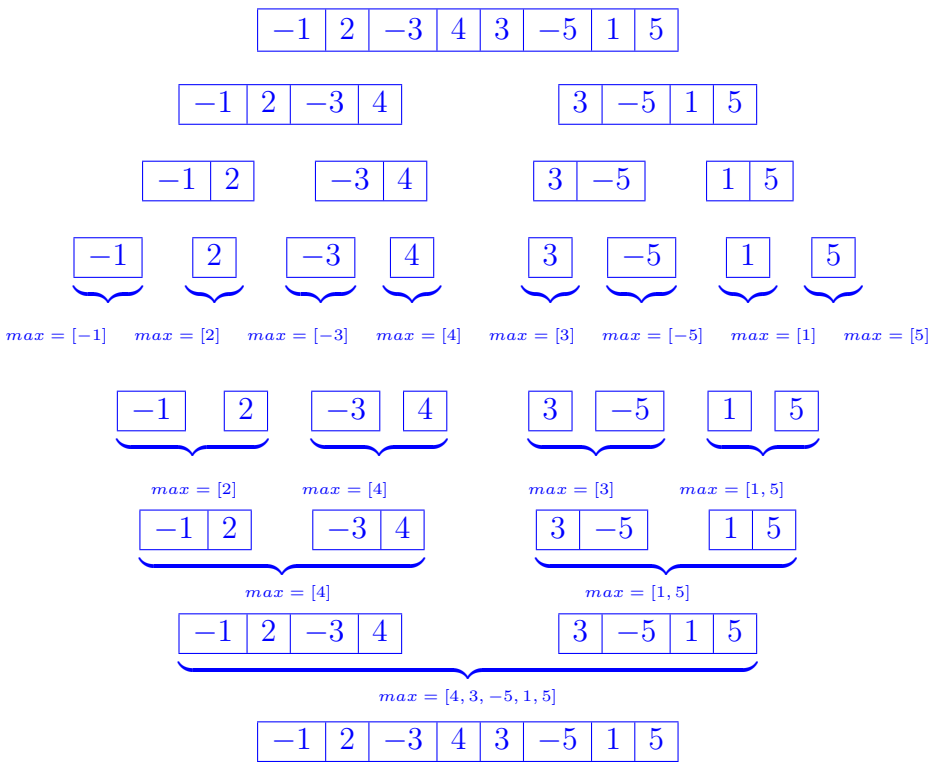
Name: Matriculation number:

Exercise 4 14 Points

Divide and Conquer

The maximum-subarray algorithm finds the contiguous subarray that has the largest sum within an **unsorted** array `A[0...n-1]` of integers. For example, for array `A = [-2, -5, 6, -2, -3, 1, 5]`, the maximum subarray is `[6, -2, -3, 1, 5]`.
The algorithm works as follows:
Firstly, it divides the input array into two equal partitions: **I** (A[0]...A[mid]) and **II** (A[mid+1]...A[n-1]). Afterwards, it calls itself recursively on both partitions to find the maximum subarray of each partition. The combination step decides the maximum-subarray by comparing three arrays: the maximum-subarray from the left part, the maximum-subarray from the right part, and the maximum-subarray that overlaps the middle. The maximum-subarray that overlaps the middle is determined by considering all elements to the left and all elements to the right of the middle.

4.1 [4 points] Based on the above algorithm description, draw a tree that illustrates the process of determining the maximum subarray in array `A = [-1, 2, -3, 4, 3, -5, 1, 5]`.



4.2 [10 points] Design a **divide and conquer** algorithm that finds and returns the maximum subarray of an unsorted array. Use either C or pseudocode for your solution.

```
Algorithm: MaxCrossSubarray(A, low, mid, high)
1  leftSum = −∞;
2  sum = 0;
3  for i = mid to low do
4      sum = sum + A[i];
5      if sum > leftSum then
6          leftSum = sum;
7          maxLeft = i;
8  rightSum = −∞;
9  sum = 0;
10 for j = mid + 1 to high do
11     sum = sum + A[j];
12     if sum > rightSum then
13         rightSum = sum;
14         maxRight = j;
15 return (maxLeft, maxRight, leftSum+rightSum);
```

```
Algorithm: FindMaxSubarray(A, low, high)
16 if high == low then
17     return (low, high, A[low]);
18 else
19     mid = ⌊(low+high)/2⌋;
20     (leftLow, leftHigh, leftSum)=FindMaxSubarray(A, low, mid);
21     (rightLow, rightHigh, rightSum)=FindMaxSubarray(A, mid+1, high);
22     (crossLow, crossHigh, crossSum)=MaxCrossSubarray(A, low, mid, high);
23     if leftSum ≥ rightSum and leftSum ≥ crossSum then
24         return (leftLow, leftHigh, leftSum);
25     else if rightSum ≥ leftSum and rightSum ≥ crossSum then
26         return (rightLow, rightHigh, rightSum);
27     else
28         return (crossLow, crossHigh, crossSum);
```

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
Spring 2018

Midterm 2
27.04.2018

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.
  - 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:

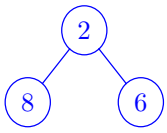
Correction slot — Please do not fill out the part below

Exercise	1	2	3	Total
Points Achieved				
Maximum Points	16	22	22	60

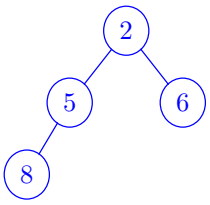
Exercise 1 16 Points

Heaps

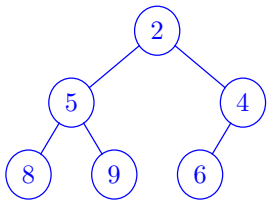
1.1 [3 points] The values **8 6 2 ; 5 ; 9 4 ;** are inserted in the given order into an empty `min-heap`. Draw the `min-heap` at the positions marked by a semicolon.



[1 point]



[1 point]



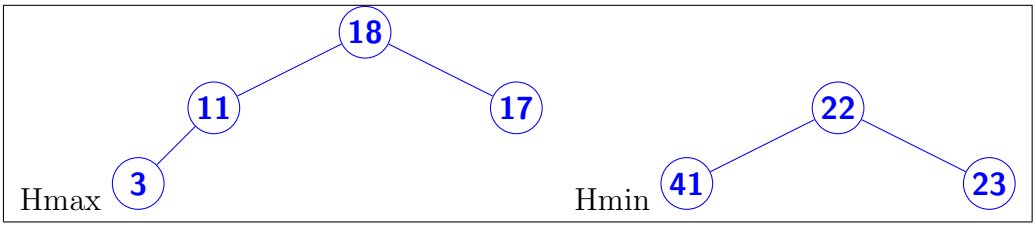
[2 points]

1.2 [4 points] A possibility to efficiently compute the median of a set S with n values is to maintain a so-called `median heap` data structure that consists of two heaps: A max heap for the smaller half of the elements and a min heap for the larger half of the elements.

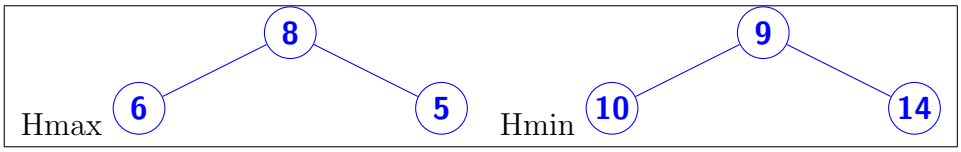
Assume a sorted array $A[1..n]$ that includes the n values of S. Let $i = \lceil n/2 \rceil$. Hmax is a max heap that includes values $A[1], ..., A[i]$. Hmin is a min heap that includes values $A[i+1], ..., A[n]$. The median of S is defined as follows:

$$median(S) = \begin{cases} A[i] & \text{iff } odd(n) \\ (A[i] + A[i+1])/2 & \text{iff } even(n) \end{cases}$$

Assume $S_1 = \{23, 11, 41, 18, 17, 22, 3\}$ and $S_2 = \{9, 6, 8, 10, 5, 14\}$. Show the median heap for S_1 and S_2 respectively.



Median Heap for S_1



Median Heap for S_2

1.3 [3 points]

Assume a `median heap S` consisting of Hmin and Hmax heaps. Use `S` to define the median.

$$median(S) = \begin{cases} Hmax[1] & \text{iff } size(Hmax) > size(Hmin) \\ Hmin[1] & \text{iff } size(Hmin) > size(Hmax) \\ (Hmin[1] + Hmax[1])/2 & \text{iff } size(Hmin) = size(Hmax) \end{cases}$$

1.4 [6 points] Assume the `median-heap` data structure. Use C or Pseudocode to implement the functions **insert**, which inserts a new value into the `median-heap`, and **getMedian**, which returns the current median value.

You can use any of the following helper functions:

- `void minInsert(int val)`
  inserts a new value in a `min-heap`
- `int minDelete()`
  deletes and returns the root value of a `min-heap`
- `int getMinSize()`
  returns the size of `min-heap`
- `void maxInsert(int val)`
  inserts a new value in a `max-heap`
- `int maxDelete()`
  deletes and returns the root value of a `max-heap`
- `int getMaxSize()`
  returns the size of `max-heap`

```
void balanceHeap() {
    if ((getMinSize() - getMaxSize()) > 1) {
        maxInsert(minDelete());
    } else if ((getMaxSize() - getMinSize()) > 1) {
        minInsert(maxDelete());
    }
}

void insert(int val) {
    if (getMaxSize() == 0) {
        maxInsert(val);
        return;
    }

    if (maxHeap[1] > val) {
```

```
        maxInsert(val);
    } else {
        minInsert(val);
    }

    balanceHeap();
}

float getMedian() {
    float sum;
    if (getMaxSize() > getMinSize()) {
        return maxHeap[1];
    }

    if (getMaxSize() < getMinSize()) {
        return minHeap[1];
    }

    sum = maxHeap[1] + minHeap[1];

    return sum/2;
```

Name: Matriculation number:

Exercise 2 22 Points

Lists and Quicksort

Consider a doubly-linked lists *dll* defined as follows:

```
struct list {
  struct Node* head;
  struct Node* tail;
};

struct Node {
  int val;
  struct Node* next;
  struct Node* prev;
};

struct list* dll;
```

We refer to the elements in the list by their position. For example, in a doubly-linked list with three elements the first element is element number 1 and the last element is element number 3.

Below we consider positional insertions into the list. If an element is inserted at position n then it is inserted directly after the element at position n. If an element is inserted at position $n = 0$ then the new node is added at the beginning of the list.

2.1 [6 points] Draw pictures that illustrate the details of the following insertion operations: `Your illustration must clearly work out the modification of pointers`.
Assume `nNode` is a pointer to the new node to be inserted and `cNode` is a pointer to the node at position n. Use the tables to list all pointers that will be modified as a result of insertion.

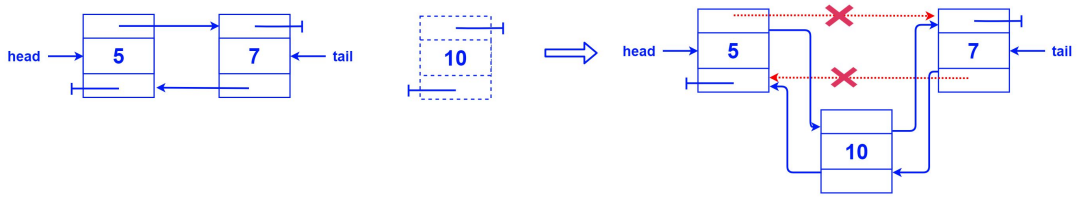
1. insert value 5 at position 0 into an empty list

Pointers that must be modified
head
tail

2. insert value 7 at position 1 into a list with one node

Pointers that must be modified
tail
cNode− >next
nNode− >prev

3. insert value 10 at position 1 into a list with two nodes



Pointers that must be modified
cNode− >next
cNode− >next− >prev
nNode− >next
nNode− >prev

2.2 [9 points] Implement function `void insert(struct list* dll, int val, int n)` that inserts a new node with value *val* at position *n* into the doubly linked list *dll*.

```
void insertAt(struct list* dll, int val, int n){
  struct Node* current = dll->head;
  int i = n;
  while (i > 1) {
    current = current->next;
    i--;
  }
  struct Node* y = (struct Node*)malloc(sizeof(struct Node));
  y->val = val;
  if (n == 0) {
    y->next = current;
    dll->head = y;
  } else {
    y->next = current->next;
    current->next = y;
    y->prev = current;
  }
  if (y->next == NULL) {
    dll->tail = y;
  } else {
    y->next->prev = y;
  }
```

2.3 [4 points] Consider the following Hoare partitioning algorithm:

```
1 Algorithm: HoarePartition(A,left,right)
2 x = floor((left+right)/2);
3 exchange A[left] and A[x];
4 x = A[left]; i = left-1; j = right+1;
5 while true do
6     repeat j = j-1 until A[j]≤x;
7     repeat i = i+1 until A[i]≥x;
8     if i<j then exchange A[i] and A[j] else return ???;
```

Use array `A = [6,9,2,7,5,4,0,3,8]` to illustrate the Hoare partitioning algorithm for $l = 0$ and $r = 8$. Complete the following matrix by showing the state after each execution of an exchange operation.

i	**j**	**A[0]**	**A[1]**	**A[2]**	**A[3]**	**A[4]**	**A[5]**	**A[6]**	**A[7]**	**A[8]**
-1	9	5	9	2	7	6	4	0	3	8
0	7	3	9	2	7	6	4	0	5	8
1	6	3	0	2	7	6	4	9	5	8
3	5	3	0	2	4	6	7	9	5	8

2.4 [3 points] Specify what must be returned on `line 8` of algorithm Hoare Partition (what to put instead of `???`). Specify the quicksort algorithm that uses the above partitioning function.

In this case, as pivot is first element, so j should be returned.

```
1 Algorithm: QuickSort(A,l,r)
2 if l<r then
3     m = LomutoPartition(A,l,r);
4     QuickSort(A,l,m);
5     QuickSort(A,m+1,r);
```

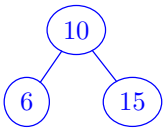
Exercise 3 22 Points

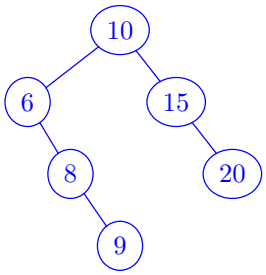
Binary Search Trees and ADTs

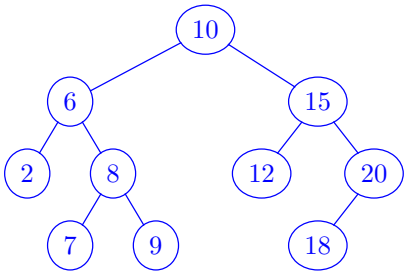
3.1 [9 points] The LIFO behaviour of a `stack` shall be simulated with FIFO queues. Assume a FIFO `queue` data structure with the following operations `enQueue(queue q, int val)`, `int deQueue(queue q)`, and `int size(queue q)`. Use one or more FIFO queues to implement the `pop()` and `push()` functions of a LIFO stack.

```
void push(int x) {
  enQueue(q1, x);
}

int pop() {
  if (size(q1) == 0) return -1;
  while (size(q1) != 1) { enQueue(q2, deQueue(q1)); }
  int val = deQueue(q1);
  while (size(q2) != 0) { enQueue(q1, deQueue(q2)); }
  return val;
}
```

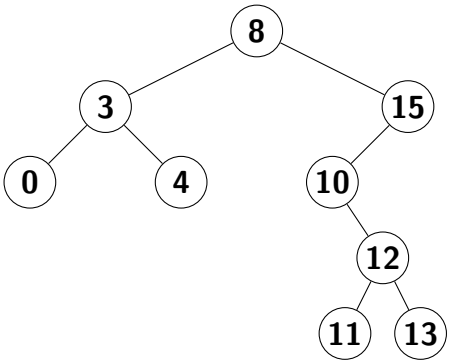
3.2 [3 points] The values **10 6 15 ; 8 9 20 ; 7 18 2 ;** are inserted in the given order into an empty `binary search tree`. Draw the binary search tree at the positions marked by a semicolon.



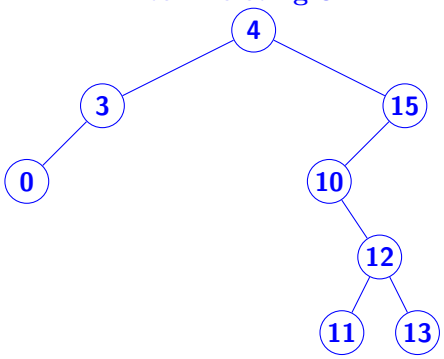




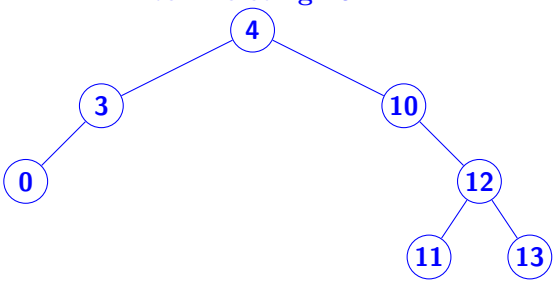
2.3 [4 points] First delete **8** from the given `binary search tree`. Then remove **15** from the result of tree obtained by deleting 8. Draw the binary search tree after each deletion.



After Deleting 8:



After Deleting 15:



3.4 [6 points] Consider a binary tree defined as follows:

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

struct node* root;
```

Implement a function `isBST(struct node* root)` that checks if a tree is a `Binary Search Tree`.

```
int isBST(struct node* r) {
  if (r == NULL) return 1;
  if (r->left != NULL && r->val < r->left->val) return 0;
  if (r->right != NULL && r->val > r->right->val) return 0;
  return isBST(r->left) && isBST(r->right);
}
```

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 — Midterm1
Spring 2019 — 22.03.2019

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 (20 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 notes (handwritten/ printed/ photocopied).
  - A foreign language dictionary is allowed. The dictionary will be checked by a supervisor.
  - No additional items are allowed except 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	Total
Points Achieved					
Maximum Points	10	17	15	8	50

Exercise 1 — 5+5 = 10 Points

Arrays and Sorting

1.1 Consider an array `A[1,...,n]` of integers. The array contains values between 0 and `m-1` and may contain duplicate elements. Implement an algorithm that prints the elements of array `A` in **ascending** order. The complexity of your solution must be $O(n+m)$. Use either C or pseudocode for your solution.

Example:
```
Input:  n=16, m=7, A=[5,0,2,4,3,6,1,1,5,5,0,6,0,0,2,4]
Output:  0 0 0 0 1 1 2 2 3 4 4 5 5 5 6 6
```

Hint: Use an auxiliary data structure for the frequencies of the elements in `A`.

Algorithm: CountingSort (A,n,m)

```
1 frequency[1..m] = [0,..,0];
2 for i = 1 to n do
3 |   val = A[i];
4 |   frequency[val+1]= frequency[val+1] + 1;
5 for i = 1 to m do
6 |   for j = 1 to frequency[i] do
7 |   |   print "i-1 ";
```

- 2pt for frequency.
- 3pt for correct print.

Name: Matriculation number:

1.2 Consider algorithm B1 shown below. The input is an array `A[1,...,n]` of integers and $n \geq 2$.

Algorithm: B1(A,n)

```
1  index = n-1;
2  while index ≥ 1 do
3      if index == n then
4          index = index - 1;
5      if A[index] ≥ A[index + 1] then
6          index = index - 1;
7      else
8          tmp = A[index];
9          A[index] = A[index+1];
10         A[index+1] = tmp;
11         index = index + 1;
```

Answer the questions below. Write your solution into the answer boxes.

(a) What does algorithm B1 do?

Sorts the elements in descending order

(b) What is the asymptotic complexity of algorithm B1 in the worst case?

$\Theta(n^2)$

(c) What is the asymptotic complexity of algorithm B1 in the best case?

$\Theta(n)$

(d) What will algorithm B1 do if **line 5** is changed as follows?

if $A[index] \leq A[index + 1]$ **then**

Sorts the elements in ascending order

(a) 1 pt

(b) 1.5 pt

(c) 1.5 pt

(d) 1 pt

Name: Matriculation number:

Exercise 2 6+5+6 = 17 Points

Asymptotic Complexity

2.1 Calculate the simplest possible asymptotic tight bound for the following functions. Write your solution into the answer boxes.

a) $f_1(n) = \sqrt{n} + n \log_2 n + \log_2 n^2$

$f_1(n) = \sqrt{n} + n \log_2 n + \log_2 n^2$
$f_1(n) = \sqrt{n} + n \log_2 n + 2 \log_2 n$
$f_1(n) \in \Theta(n \log_2 n)$

$\Theta(n \log_2 n)$

b) $f_2(n) = \log_2 (8n^3 log_2 4) + \sqrt{n}$

$f_2(n) = \log_2 (8n^3 log_2 4) + \sqrt{n}$
$f_2(n) = \log_2 8 + \log_2 n^3 + \log_2 (\log_2 4) + \sqrt{n}$
$f_2(n) = 3 + 3 \log_2 n + \log_2 2 + \sqrt{n}$
$f_2(n) = 3 + 3 \log_2 n + 1 + \sqrt{n}$
$f_2(n) = 4 + 3 \log_2 n + \sqrt{n}$
$f_2(n) \in \Theta(\sqrt{n})$

$\Theta(\sqrt{n})$

c) $f_3(n) = 2^{7n} + 10n + \log_2 24$

$f_3(n) = 2^{7n} + 10n + \log_2 24$
$f_3(n) \in \Theta(128^n)$

$\Theta(128^n)$

d) $f_4(n) = 7n^{max(\log_2 n^{\log_2 4 \cdot 2}, \sqrt{n})}$

$max(\log_2 n^{\log_2 4 \cdot 2}, \sqrt{n}) = max(\log_2 n^{\log_2 8}, \sqrt{n})$

$max(\log_2 n^3, \sqrt{n}) = max(3 \log_2 n, \sqrt{n}) => max = \sqrt{n}$

$f_4(n) = 7n^{max(\log_2 n^{\log_2 4 \cdot 2}, \sqrt{n})}$

$f_4(n) = 7n^{\sqrt{n}}$

$f_4(n) \in \Theta(n^{\sqrt{n}})$

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

e) $f_5(n) = 1 + n^{\frac{\log_3 32}{\log_3 2}} + n^2 + n^3 + n^4$

$f_5(n) = 1 + n^{\frac{\log_3 32}{\log_3 2}} + n^2 + n^3 + n^4$

$f_5(n) = 1 + n^5 + n^2 + n^3 + n^4 \in \Theta(n^5)$

$\Theta(n^5)$

f) $f_6(n) = \log_{\log 5}(\log^{\log 100} n)$

$f_6(n) = \log_{\log 5}(\log^{\log 100} n)$

$f_6(n) = \log_{\log 5}(\log 100 \cdot \log n)$

$f_6(n) = \log_{\log 5}(\log 100) + \log_{\log 5}(\log n)$

$f_6(n) \in \Theta(\log(\log n))$

$\Theta(\log(\log n))$

– 1 pt each for correct solution in the box

2.2 Calculate the asymptotic tight bound of the following recurrences. If the Master Theorem can be used write down the case (1-3). Write your solution into the answer boxes. Assume T(1) = 0 for all cases.

a) $T(n) = 5T(\frac{n}{7}) + (\log n)^2$
a=5, b=7, f(n)= $(\log n)^2$
Case 1: $T(n) = \Theta(n^{log_7 5})$

Case:	Complexity:
1	$\Theta(n^{log_7 5})$

b) $T(n) = T(n-3) + \frac{n}{3}$

$$\begin{aligned}
T(n) &= T(n-3) + \frac{n}{3} \\
&= T(n-6) + \frac{n-3}{3} + \frac{n}{3} \\
&= T(n-6) + \frac{2n-3}{3} \\
&= T(n-9) + \frac{n-6}{3} + \frac{2n-3}{3} \\
&= T(n-9) + \frac{3n-9}{3} \\
&= T(n-12) + \frac{n-9}{3} + \frac{3n-9}{3} \\
&= T(n-12) + \frac{4n-18}{3} \\
&= \dots \\
\Rightarrow T(n) &= T(n-3k) + \frac{k \cdot n}{3} - \sum_{i=1}^{k} i - 1
\end{aligned}$$

k grows until it reaches $k = \lfloor \frac{n}{3} \rfloor$, thus we have:

$$T(n) = \frac{n}{3} \cdot \frac{n}{3} - \sum_{i=1}^{\frac{n}{3}} i - 1 = \frac{n^2}{9} - \frac{n-3}{18} \cdot n = \frac{n^2}{9} - \frac{n^2}{18} + \frac{1}{6} = \frac{n^2}{18} + \frac{1}{6} \in \Theta(n^2)$$

Case:	Complexity:
-	$\Theta(n^2)$

c) $T(n) = 2T(\frac{n}{3}) + n \log n$
a=2, b=3, f(n)= $n \log n$
Case 3: $T(n) = \Theta(n \log n)$

Case:	Complexity:
3	$\Theta(n \log n)$

d) $T(n) = 8T(\frac{n}{2}) + n^3$
a=8, b=2, f(n)= n^3
Case 2: $T(n) = \Theta(n^3 \log_2 n)$

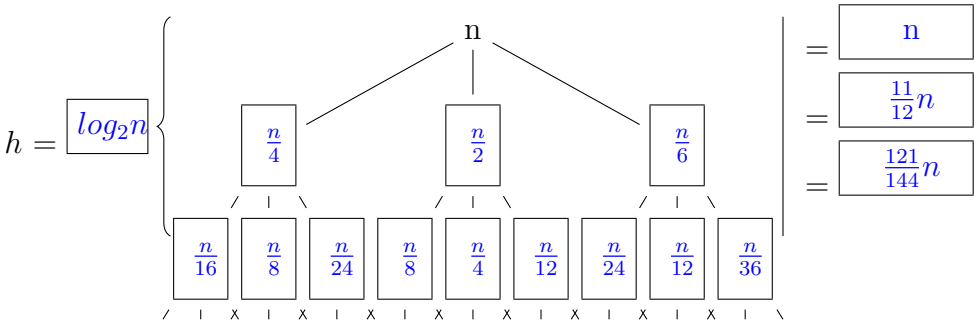
Case:	Complexity:
2	$\Theta(n^3 \log_2 n)$

- 0.5 pt for correct Case
- 0.5 pt for correct complexity
- 2 pt for correct complexity for part b

2.3 Consider the recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/4) + T(n/2) + T(n/6) + n & \text{if } n > 1 \end{cases}$$

Finish a recursion tree and use it to estimate the asymptotic upper bound of $T(n)$. Write your solutions into the answer boxes.



Complexity: $\Theta(n)$

Tree grows until $\left(\frac{1}{2}\right)^h n = 1 \implies h = \log_2 n$ To get an upper bound, we can use the sum $n \sum_{h=0}^{h} (\frac{11}{12})^h$.
Guess: $O(n)$

- 1 pt for 'h'
- 1 pt for complexity
- 0.2 pt for all other correct boxes in tree

Name: Matriculation number:

Exercise 3 1+3+4+7 = 15 Points

Runtime Analysis

Suppose `A[1,...,n]` is a **sorted** array of **unique integers**. A *fixed point* of an array is an index $i \in \{1, ..., n\}$ so that $A[i] = i$.

The goal of the following program is to return `True` if there is a fixed point, and to return `False` otherwise.

Function Call: isThereAFixedPoint(A,1,n)

Algorithm: isThereAFixedPoint(A, lower, upper)

```
1 mid = ⌊(lower + upper)/2⌋;
2 if A[mid] == mid then
3     return True;
4 if lower == upper then
5     return False;
6 if A[mid] > mid then
7     return isThereAFixedPoint(A, lower, mid);
8 if A[mid] < mid then
9     return isThereAFixedPoint(A, mid + 1, upper);
```

3.1 What is the fixed point in array $B = [-1, -2, 0, 4, 5, 7]$

4 or 5

– 1 pt for correct answer

3.2 Suppose the algorithm `isThereAFixedPoint` is applied to input array [-2,-1,2,3,4,5,7,14]. What are values of `lower`, `upper`, `mid`, `A[mid]` each time after `isThereAFixedPoint` has executed `line 1`?

No.	lower	upper	mid	A[mid]
1	1	8	4	3
2	5	8	6	5
3	7	8	7	7

– 0.25 pt for each correct value except No. column

3.3 Consider the worst-case runtime of `isThereAFixedPoint`. Specify the recurrence relation and calculate the asymptotic complexity of `isThereAFixedPoint` for this case.

Reccurence relation:

$T(n) = T(n/2) + \Theta(1)$

Complexity:

$O(\log n)$

The algorithm satisfies recurrence relation:
$T(n) = T(n/2) + \Theta(1)$

because in each call to isThereAFixedPoint we recur on a set of size about n/2, and then there is $\Theta(1)$ overhead to increment the pointer. By, for example, Master Theorem, this means
$T(n) \in O(\log n)$

- 2 pt for recurrence relation
- 2 pt for complexity

3.4 Do an exact analysis and calculate the asymptotic tight bound of Algorithm D2.

Algorithm: D2(n)

```
1 result = 0;
2 for i = 1 to n do
3     sum = 0;
4     for j = 1 to i − 1 do
5         if i mod j == 0 then
6             sum = sum + j;
7     if sum == i then
8         result = result + 1;
9 return result;
```

line number	cost	number of times executed
line 1	c_1	1
line 2	c_2	$n+1$
line 3	c_3	n
line 4	c_4	$\sum_{i=1}^{n}(\sum_{j=1}^{i} 1) = n\frac{(n+1)}{2}$
line 5	c_5	$n\frac{(n-1)}{2}$
line 6	c_6	$\alpha\frac{n}{2}(n-1)$
line 7	c_7	n
line 8	c_8	βn
line 9	c_9	1

* $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$

$$\mathrm{T}(n) \in \boxed{\Theta(n^2)}$$

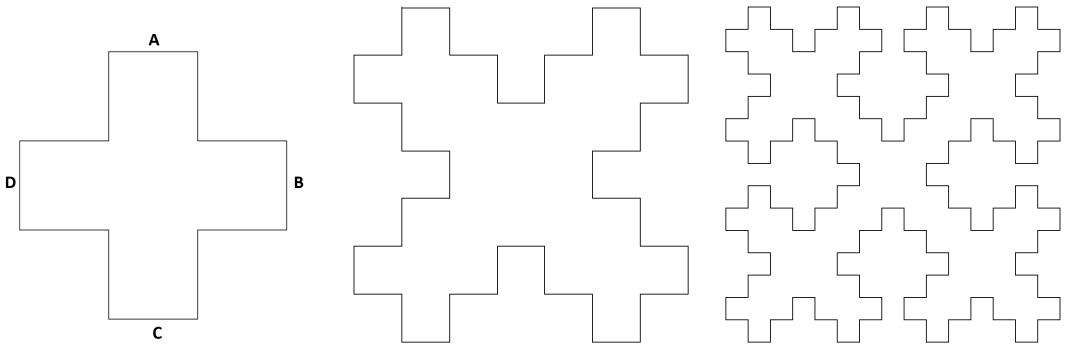
$$\mathrm{T}(n) = c_1 + (n+1)c_2 + nc_3 + n\tfrac{(n+1)}{2}c_4 + n\tfrac{(n-1)}{2}c_5 + \alpha\tfrac{n}{2}(n-1)c_6 + nc_7 + \beta nc_8 + c_9$$

- 0.25 pt - line 1 and line 9
- 0.5 pt - line 2 , line 3 and line 7, line 8
- 1 pt - line 5, line 6
- 2 pt - line 4
- 0.5 pt for T(n)

Exercise 4 2+4+2 = 8 Points

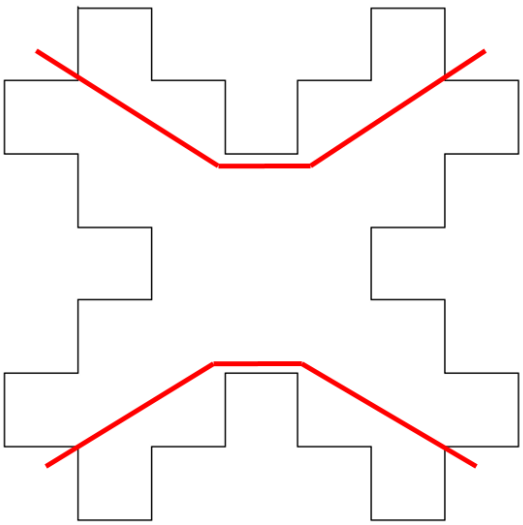
Divide and Conquer

Consider the space filling curves of, respectively, order 0, order 1 and order 2, illustrated below.



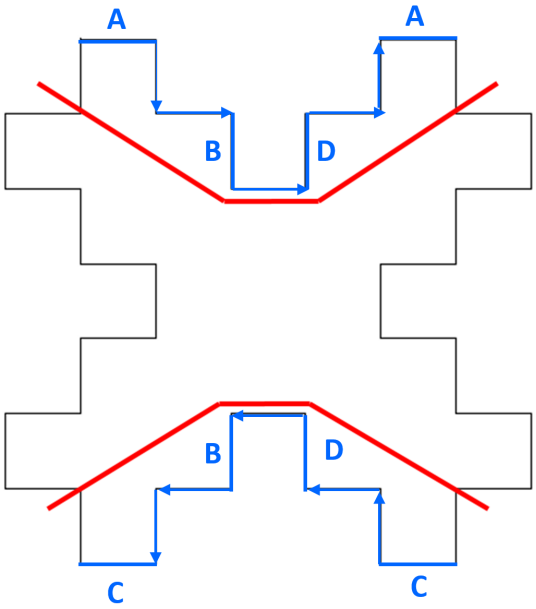
4.1 Illustrate in the indicated parts in figures below, how curves of order i are composed to curves of order i+1. Draw directly into the figures and label multi-line segments with, respectively, A, B, C and D to explain the solution. Precisely denote start and beginning of multi-line segments.

Order 1:

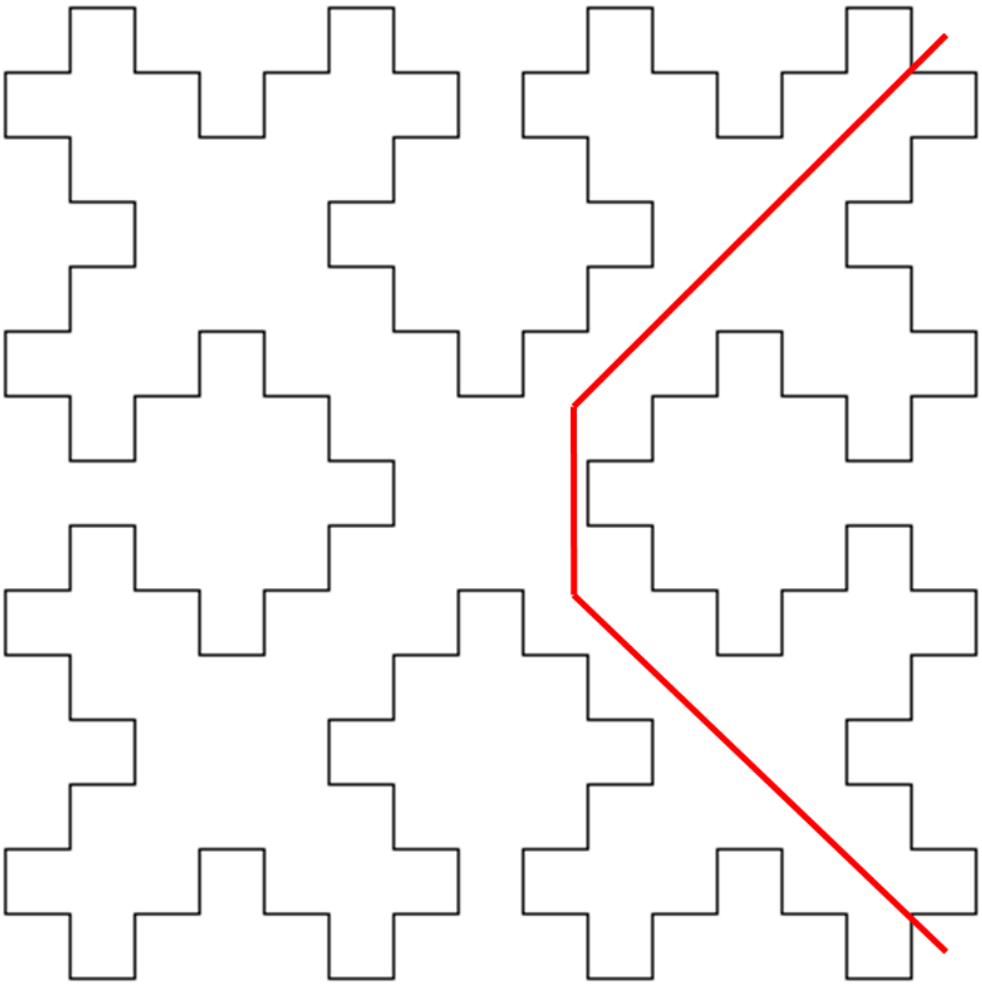


Name: Matriculation number:

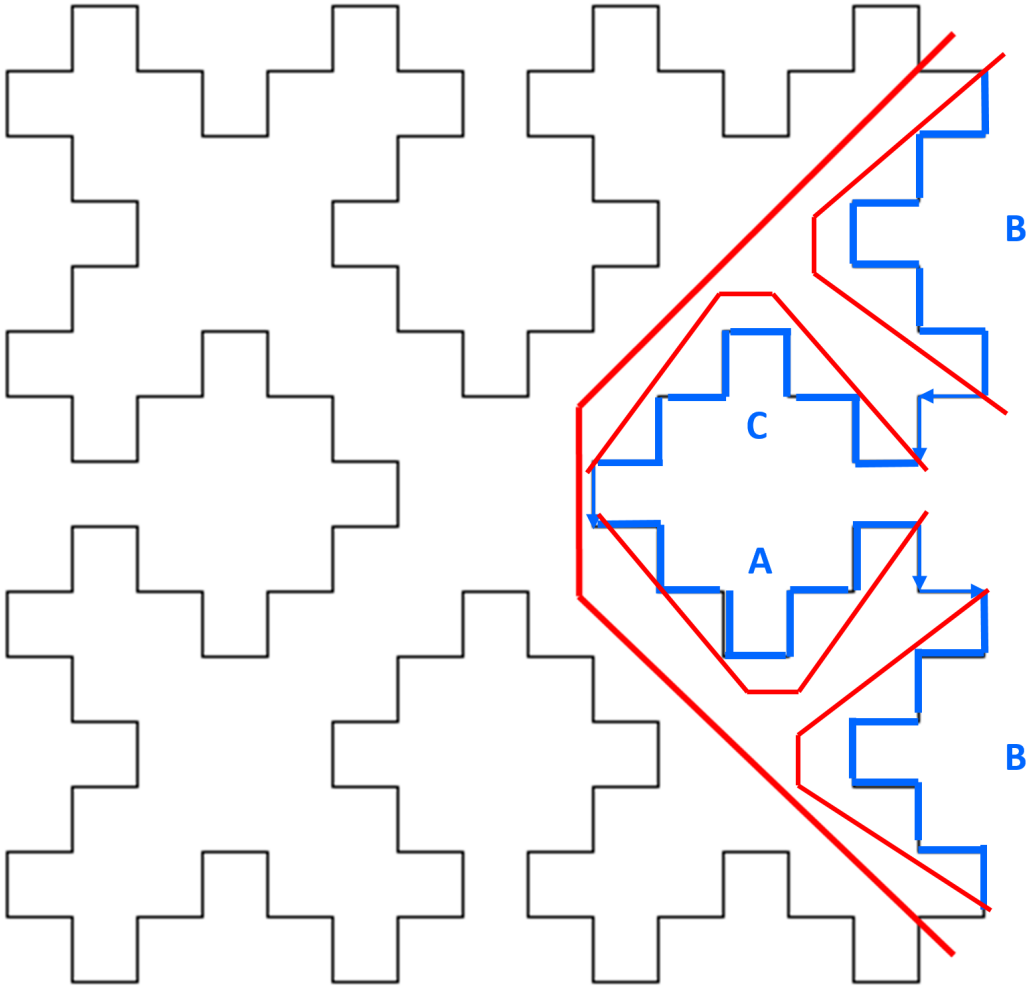
Order 1:



Order 2:



Order 2:



– 0.25 pt for each correct label

4.2 Recursively define curve C, i.e., define C of order `i+1` in terms of curves of order `i`.

$$C = C \uparrow \leftarrow D \leftarrow B \leftarrow \downarrow C$$

– 0.25 pt for each label of order 1

– 1 pt for each label of order 2

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 2019 — 03.05.2019

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 materials are 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.
  - No additional items are allowed except 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:

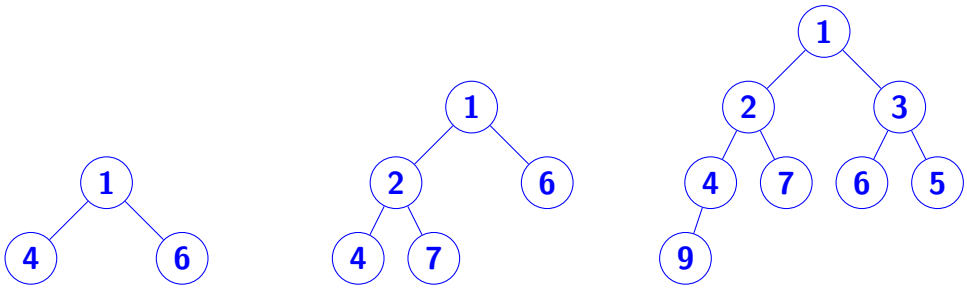
Correction slot — Please do not fill out the part below

Exercise	1	2	3	4	5	Total
Points Achieved						
Maximum Points	6	20.5	7.5	8	8	50

Exercise 1 — 2.5+0.5+3 = 6 Points

Heaps

1.1 The values **4 1 6 ; 2 7 ; 5 3 9 ;** are inserted in the given order into an empty `min-heap`. Draw the `min-heap` at the positions marked by a semicolon.



1.2 What is the worst case asymptotic time complexity for building a min-heap with n values by inserting the n values one-by-one as described in Task 1.1?

$\Theta(nlogn)$

Name: Matriculation number:

1.3 Consider an array $a[0...n-1]$. Write a function `int isMaxHeap(int a[], int i, int n)` that returns true if $array[i...n-1]$ represents max-heap, otherwise it returns false. You can write your implementation using C or pseudocode.

```
1  Algorithm: isMaxHeap(a,i,n)
2  if i > (n-2)/2 then
3      return 1;
4  leftCheck = 0;
5  rightCheck = 0;
6  if a[i] ≥ a[2*i+1] then
7      leftCheck = isMaxHeap(a,2*i+1, n) ;
8  if 2*i + 2 < n then
9      if a[i] ≥ a[2*i+2] then
10         rightCheck = isMaxHeap(a,2*i+2,n);
11 return leftCheck && rightCheck ;
```

Exercise 2 2+2+1.5+2+2+5+6 = 20.5 Points

Binary Trees

2.1 Consider the Binary Search Tree `BST1` in Figure 1. Delete **45** from `BST1`. Draw the binary search tree after the deletion.

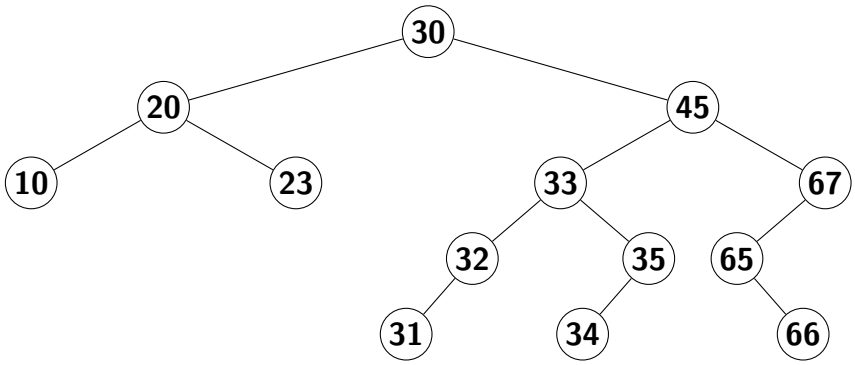
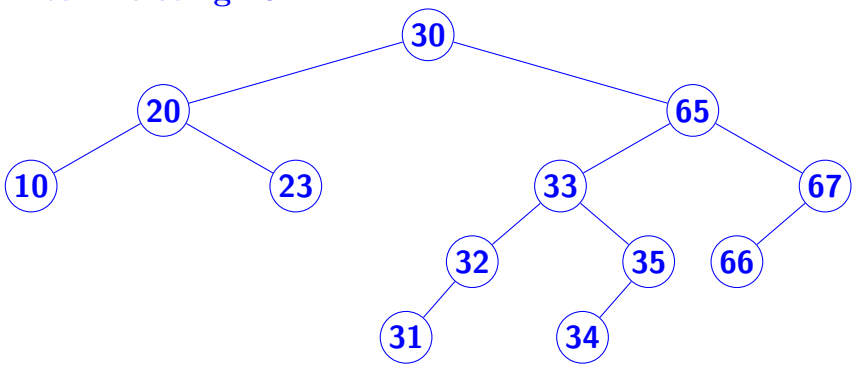


Figure 1: Binary Search Tree BST1

After Deleting 45:



2.2 Consider a binary search tree that includes nodes with keys between 1 and 100. Assume a successful search for node with key 45. Which of the following sequences could be sequences of nodes examined on the path from the root to the node with key 45?

a. 7, 3, 1, 11, 39, 34, 77, 63, 45

b. 1, 2, 3, 4, 5, 6, 7, 8, 45

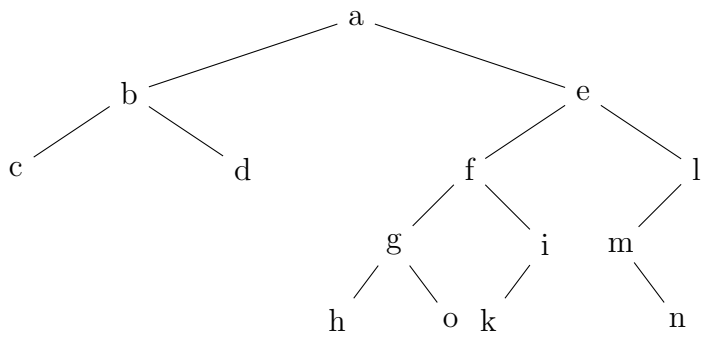
c. 9, 8, 63, 0, 4, 3, 2, 1, 45

d. 50, 25, 26, 27, 40, 44, 45

e. 99, 45

b, d, e

2.3 Consider the following binary search tree (note that the displayed node labels are independent of the search keys, which are not displayed in the figure).



Determine

1. The predecessor of e

   i

2. The predecessor of h

   a

3. The predecessor of m

   e

2.4 A radix tree can be used for storing bit string keys lexicographically. Given two bit strings $a = a_0a_1...a_p$ and $b = b_0b_1...b_q$, where each a_i and each b_j is either 0 or 1, we say that a is lexicographically less than b if either

- there exists an integer j, where $0 \leq j \leq min(p, q)$, such that $a_i = b_i$ for all $0 \leq i \leq j - 1$ and $a_j < b_j$, or
- $p < q$ and $a_i = b_i$ for all $0 \leq i \leq p$

In a radix tree:

- Each node's key can be determined by traversing the path from root to that node.
- There is no need to store the keys in the nodes, the keys in the Figure 2 are for illustrative purpose only.
- Nodes are shaded if the keys corresponding to them are not in the tree; such nodes are needed as part of paths to other nodes.

For example, the radix tree data structure shown in Figure 2 stores the bit strings 100, 10, 1011, 011, and 0.

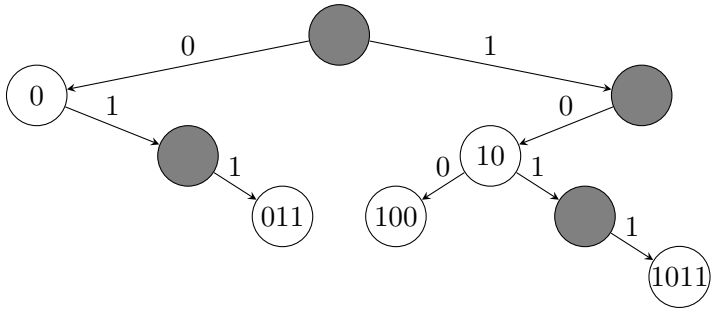


Figure 2: Radix Tree

When searching for a key we go left at a node at depth i if $a_i = 0$ and right if $a_i = 1$.

1. Define and initialize a C data structure for radix trees, such as the one illustrated in Figure 2. The nodes shall not store the actual bit string keys.

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

2. Assume the set of binary strings represented by the radix tree in Figure 2 shall be printed in the lexigrophaic order defined at the beginning of Task 2.4. State the order in which the nodes must be printed.

0, 011, 10, 100, 1011

3. Write a function that prints the set of binary strings represented by a radix tree in lexicographic order defined in the beginning of Task 2.4.

```
void preOrder(struct node* root, int i, int level){
  if (root == NULL) return;
  if (root->hasVal == 1) printf("%0*d\n", level, i);
  preOrder(root->left, 10*i, level+1);
  preOrder(root->right, 10*i+1, level+1);
}
```

Alternate Solution

First Call: preOrder(root, str,' ', -1)

```
void display(char str[], int N){
    int i;
    for(i= 0;i<=N;i++){
        printf("%c",str[i]);
    }
    printf("\n");
}
void preOrder(struct radixTreeNode* root,char str[], char v,
    int level){
  if(root == NULL) return;

  if(level >= 0){
    str[level] = v;
  }
  if(root->hasValue == 1) display(str,level);
  preOrder(root->left,str,'0',level+1);
  preOrder(root->right,str,'1',level+1);
}
```

4. Write a function that inserts a bit string B into a radix tree.

```
struct node* insert(struct node* p, char s[], int i, int n) {
    if (p == NULL) {
        p = malloc(sizeof(struct node));
        p->hasVal = 0;
        p->left = NULL;
        p->right = NULL;
    }
    if (i == n) p->hasVal = 1;
    else if (s[i] == '0') p->left = insert(p->left,s,i+1,n);
    else p->right = insert(p->right,s,i+1,n);
    return p;
}
```

Exercise 3	1 + 1 + 3.5 + 2 = 7.5 Points

Quicksort

3.1 What is the worst case runtime complexity of Quicksort?

Worst case Quicksort:

$\Theta(n^2)$

3.2 State the recurrence relation for the run time complexity of a randomized Quicksort algorithm that uses Lomuto's partitioning for the case when all elements of the array have the same value.

Recurence relation:

$T(n) = T(n-1) + \Theta(n)$

3.3 Consider an array `A[p..r]`. The `Partition(A, p, r)` procedure given below returns an index `q` such that each element of array `A[p..q - 1]` is less than or equal to `A[q]` and each element of `A[q + 1..r]` is greater than `A[q]`. Modify the `Partition` procedure such that it returns two indices `q` and `t` where p ≤ q ≤ t ≤ r and

- all elements of `A[q..t]` are equal,
- each element of `A[p..q - 1]` is less than `A[q]`, and
- each element of `A[t + 1..r]` is greater than `A[q]`.

```
1 Algorithm: PARTITION(A, p, r)
2 x = A[r];
3 i = p-1;
4 for j = p to r − 1 do
5     if A[j] ≤ x then
6         i = i + 1;
7         exchange A[i] with A[j];
8 exchange A[i+1] with A[r];
9 return i+1;
```

Note: You are allowed to return multiple variables in the `return` statement, e.g., `return (a, b)` returns the values of variables `a` and `b`.

```
1 Algorithm: PARTITION2(A, p, r)
2 x = A[p];
3 i = h = p;
4 for j = p + 1 to r do
5     if A[j] < x then
6         y = A[j];
7         A[j] = A[h+1];
8         A[h+1] = A[i];
9         A[i] = y;
10        i = i+1;
11        h = h+1;
12    else if A[j] == x then
13        exchange A[h+1] with A[j];
14        h = h + 1;
15 return (i, h);
```

3.4 For an array `A[p..r]` and the original partitioning procedure `Partition(A, p, r)` the call of a `Quicksort(A, p, r)` is as follows:

```
1 Algorithm: QUICKSORT(A, p, r)
2 if p < r then
3     q = Partition(A, p, r);
4     Quicksort(A, p, q-1);
5     Quicksort(A, q+1, r);
```

Modify `Quicksort(A, p, r)` such that it uses the new partitioning procedure from Task 3.3.

```
1 Algorithm: QUICKSORT2(A, p, r)
2 if p < r then
3     (q, t) = Partition2(A, p, r);
4     Quicksort2(A, p, q-1);
5     Quicksort2(A, t+1, r);
```

Name: Matriculation number:

Exercise 4 2+2+4 = 8 Points

Pointers and Abstract Data Types

4.1 What is the output of the following program:

```
int f(int x, int *py, int **ppz) {
  int y, z;
  **ppz += 1;
  z = **ppz;
  *py += 2;
  y = *py;
  x += 3;
  return x + y + z;
}

void main() {
   int c, *b, **a;
   c = 4;
   b = &c;
   a = &b;
   printf("%d ", f(c, b, a));
   return 0;
}
```

19

4.2 A part of a C program that operates on a doubly linked list is given below. Specifically, function `void func(List *listA, void *p)` inserts the new element pointed to by `p` after element `curr` in the doubly linked list. The argument `listA` of `func` points to, respectively, the first element and element `curr` of the doubly linked list. Which code fragment can be substituted for `????`, such that function `func` correctly inserts the element? Write the correct answer(s) into the box.

```
#include <stdio.h>

struct elem {
    void *val;
    struct elem *next, *prev;
};

struct list { struct elem *first, *curr; };

void func(struct list *listA, void* p) {
    struct elem* newElem;
    if (listA->curr == NULL) return;
    newElem = malloc(sizeof(struct elem));
    newElem->val = p;
    newElem->prev = listA->curr;
    newElem->next = listA->curr->next;
    if (newElem->next) { ???? }
    listA->curr->next = newElem;
    listA->curr = newElem;
}
```

Answers:

1. `listA→curr→next→prev = newElem;`
2. `(*(*listA).curr).prev = newElem;`
3. `newElem→next→prev = newElem;`
4. `listA→curr→next = newElem→next→prev;`

Note: More than one answer is possible.

1, 3

4.3 Consider the following interface to stacks.

Method	Description
`stack newStack()`	create new, empty stack
`int isEmpty(stack S)`	returns 1, if stack is empty, 0 otherwise
`void push(int x, stack S)`	push element into stack
`int top(stack S)`	returns value of top element from stack, without removing it
`int pop(stack S)`	pop element from stack

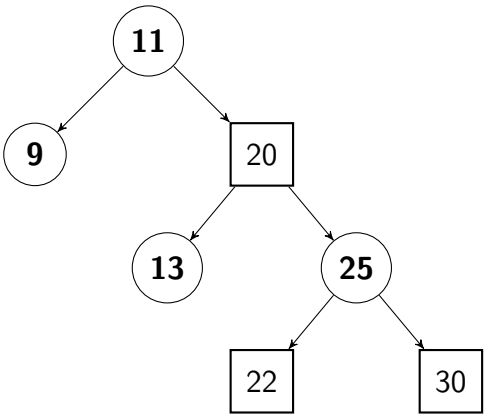
Consider that you are given `stack S` of integers. Use pseudocode to write a function `stack sortStack(stack* S)` that sorts the elements of `stack S` in ascending order. You are allowed to use auxiliary stacks in your implementation.

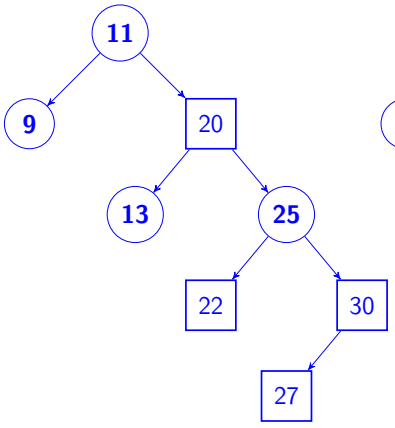
```
1 Algorithm: sortStack(S)
2 tmpStack = newStack();
3 while !isEmpty(S) do
4     tmp = pop(S);
5     while !isEmpty(tmpStack) and top(tmpStack) < tmp do
6         push(pop(tmpStack), S);
7     push(tmp, tmpStack);
8 return tmpStack;
```

Exercise 5 — 3+5 = 8 Points

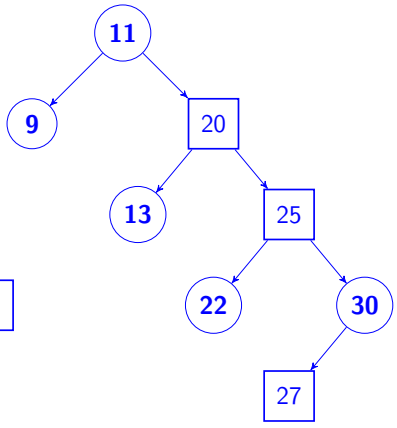
Red-black Trees

5.1 Consider the following red-black tree where black nodes are denoted with a circle and red nodes are denoted with a square. Add **27** and draw the resulting red-black tree. Clearly state the case(s) applied and draw separate tree for each case required to get the resulting red-black tree.

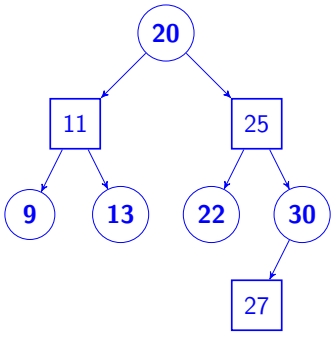




Case: 1 Mirrored



Case: 3 Mirrored



5.2 A **red-black node** is of the following type:

```
#define black 0
#define red 1

struct rb_node {
  int key, color;
  struct rb_node *left, *right, *parent;
};
```

Using a C or pseudocode write a function `int same_number_of_black_nodes(rb_node* node)` that recursively checks whether the property number 5 of red-black trees is satisfied: *For each node, all paths from the node to descendant leaves contain the same number of black nodes.* This function should return -1 if given tree does not satisfy this property otherwise it should return total number of black nodes from the node to descendant leaf node.

```
int same_number_of_black_nodes(rb_node* node) {
    if (node == NULL) return 0;

    int leftSubtree = same_number_of_black_nodes(node->left);
    int rightSubtree = same_number_of_black_nodes(node->right);
    int isBlackNode = 0;
    if(node->color == black ) isBlackNode++;

    if (leftSubtree == -1 || rightSubtree == -1 || leftSubtree != rightSubtree)
        return -1;
    else
        return leftSubtree + isBlackNode;
}
```

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):

```
1 int sum(int a[], int b[], int n) {
2   int s = 0;
3   int o = 0;
4   for (int i = n - 1; i >= 0; i--) {
5     s = s + ((a[i] + b[i] + o) % 10) * (int)pow(10,n-1-i);
6     o = (a[i] + b[i] + o) / 10 ;
7   }
8   return s + o * (int)pow(10,n);
9 }
```

code/task1b.c

2nd solution (with auxiliary array):

```
1  int sum(int a[], int b[], int n) {
2    int c[n + 1];
3    for (int i = n - 1; i >= 0; i--) {
4      c[i] = (c[i + 1] + a[i] + b[i]) / 10;
5      c[i+1] = (c[i+1] + a[i] + b[i]) % 10;
6    }
7    int sum = 0;
8    for (int i = 0 ; i <= n; i++) {
9      sum = 10 * sum + c[i];
10   }
11   return sum;
12 }
```

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)
1 index = 0;
2 while index ⩽ n − 1 do
3     while index ≠ A[index] do
4         temp = A[index];
5         A[index] = A[temp];
6         A[temp] = temp;
7     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

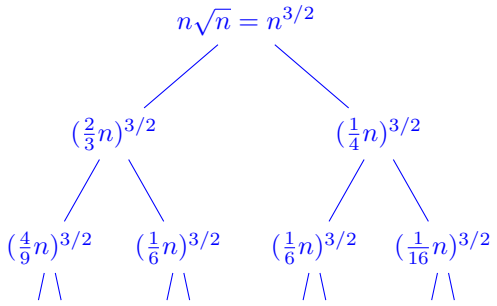
3.2 [10 points] Determine the asymptotic tight bound of the following recurrences using the indicated method.

(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:

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

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.

```
1 for (i = 1; i <= ⌈(N + 1)/2⌉; i++) do
2     for (j = i; j <= N − i; j++) do
3         if (A[j] > A[j + 1]) then
4             exchange A[j] and A[j+1];
5     for (k = N − i; k > i; k−−) do
6         if (A[k] < A[k − 1]) then
7             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* $i \leq j < N-i$

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

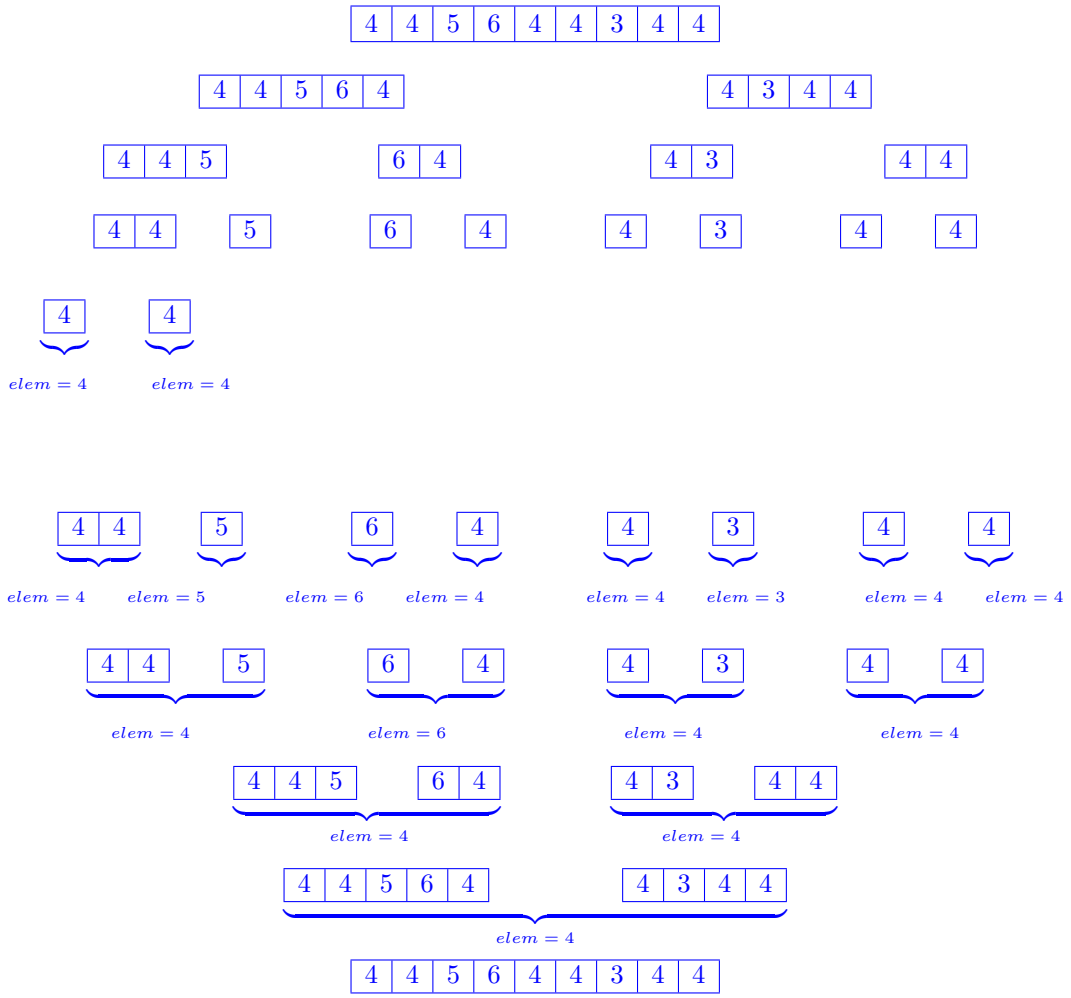
Exercise 5

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.

Name: Matriculation number:



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.

$2T(n/2) + 2n$

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
Spring 2020

Midterm 2
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:

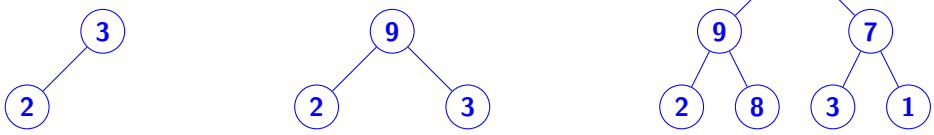
Correction slot — Please do not fill out the part below

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

Name: Matriculation number:

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 = [*l + 1*] **to** [*r*] **do**
    if *A[j]* [≤] x **then**
        i = [i + 1] ;
        exchange A[i] and A[j];
exchange [A[i]] and [A[l]] ;
return [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 each 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`

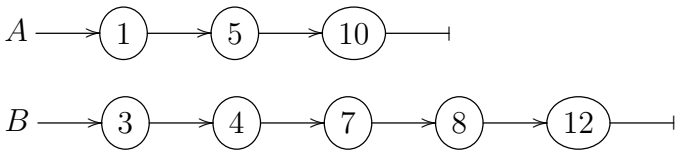
(e) `(h+1) mod n == t`

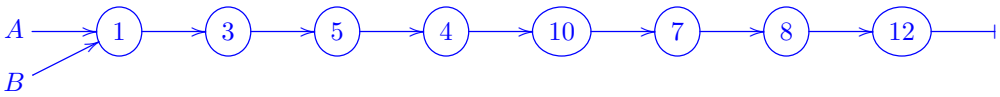
(f) `(t+1) mod n == h`

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 that are passed to the 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:

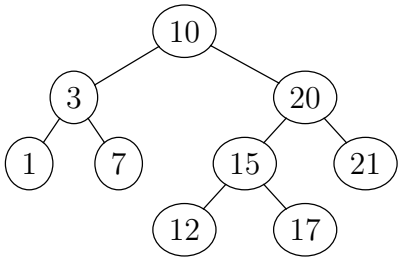
- *You may not use an auxiliary linked list or any other data structures.*
- *The merging should be implemented by modifying pointers.*
- *After merging, A and B must point to the head of the merged linked list.*

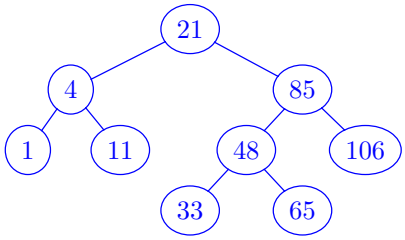
```
1  void merge(struct node** A, struct node** B){
2      if (*A == NULL && *B == NULL) return;
3      else if (*A == NULL) { *A = *B; return; }
4      else if (*B == NULL ) { *B = *A; return; }
5      struct node* curr = *A;
6      struct node* next = *B;
7      while (next != NULL) {
8          struct node* temp = next;
9          next = curr->next;
10         curr->next = temp;
11         curr = temp;
12     }
13     *B = *A;
14 }
```

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.

```
1 int MinAggregate(struct TreeNode* node, int sum){
2     if (node == NULL) { return sum; }
3     sum = node->val + MinAggregate(node->left,sum);
4     node->val = sum;
5     sum = MinAggregate(node->right,sum);
6     return sum;
7 }
```

Department of Informatics

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

University of Zurich[UZH]

Informatics II — Midterm 1
Spring 2021 — 29.03.2021

Name: _____ Matriculation number: _____

Advice

You have 60 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 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.
- There is no extra time for scanning and submission. The allotted time already includes the time for scanning and submission.
- Only submissions through EPIS are accepted.

Signature:

Correction slot — Please do not fill out the part below

Exercise	1	2	3	Total
Points Achieved				
Maximum Points	12	14	14	40

Exercise 1

For a positive integer i, $d(i)$ is the sum all digits of i plus i. For example, $d(75) = 7+5+75 = 87$. Given a positive integer i and a positive integer n, the *D-Numbers* is the sequence of n numbers such that $d(i)$, $d(d(i))$, $d(d(d(i)))$, ... $d(d(d(...)))$. In the *D-Numbers* for the integer i, the first number is $d(i)$; the second number is $d(d(i))$; the $n-th$ number is $d(d(d(...)))$.

a) [1 points] Let $i = 81$ and $n = 4$. Write down the *D-Numbers* for i.

90 99 117 126

b) [2 points] If i is always less than 1000, what is the asymptotic complexity of `void DNumbers(int i, int n)`? Explain.

$O(n)$. As i is always less than 1000, $d(i)$ is in constant time. The `void DNumbers(int i, int n)` calls $d(i)$ function n times, and the overall time complexity is $O(n)$.

c) [9 points] Write the C function `void DNumbers(int i, int n)` that prints the *D-Numbers* for i.

```
1  int d(int i) {
2    int sum = i;
3    while (i != 0) {
4      sum = sum + i % 10;
5      i = i / 10;
6    }
7    return sum;
8  }
9
10 void Dnumbers(int i, int n) {
11   int k, d_i;
12   for (k = 1; k <= n; k++) {
13     i = d(i);
14     printf("%d ", i);
15   }
16   printf("\n");
17 }
```

code/task01.c

Exercise 2

Consider the C function `WhatDoesItdo` shown below, where $n \geq 2$.

```
int WhatDoesItdo(int n) {
  int i;
  for (i = 1; i< n; i++){
    if (i * (i + 1) == n){
        return True;
    }
  }
  return False;
}
```

a) [1 point] What does `WhatDoesItdo(int n)` return for $n = 25$ and $n = 6$, respectively?

`WhatDoesItdo(25)` = False
`WhatDoesItdo(6)` = True

b) [2 points] What does `WhatDoesItdo(int n)` do? The function `WhatDoesItdo(int n)` checks whether n is the mutiplication of two consecutive positive integers.

c) [5 points]Write invariants for Initialization, Maintenance and Termination to prove the correctness of the loop.

(a) Loop Invariant: If the integer $i * (i + 1) = n$ in range$(1...n - 1)$ then the functions returned True.

(b) Initialization: i starting from 1 holds because if $1 * 2 = n$ exists it can only be in range$(1...n)$.

(c) Maintenance: Each i checks if $i * (i + 1)$ is equal to n, if it is then True is returned. Else i is increased by 1 and the same invariant holds for range$(i + 1...n - 1)$.

(d) Termination: if the integer $i * (i + 1) = n$ in range $(1...n)$ then the function returns True, else the loop terminates with returning False.

(e) If loop terminates without returning True, the False is returned to indicate that n is not the multiplication of two consecutive postive integers.

d) [2 points] What is the asymptotic complexity of the function? Explain.
$O(n)$. The dominating term of the function is the linear loop up to n, therefore the complexity of the algorithm is $O(n)$.

e) [4 points] Provide better solutions in terms of time complexity. Describe your solutions and show time complexity of your solutions.

Solution1: The function could be run faster if we use a Binary Search approach. This would reduce the asymptotic complexity to $O(\log n)$.

Solution2: The loop only needs to iterate as long as $i^2 < n$ holds. This would reduce the asymptotic complexity to $O(\sqrt{n})$.

Solutions like iterating up to $n/2$ are not accepted, since they are not correct (example: $n = 2$) and the asymptotic complexity is still $O(n)$.

Name: Matriculation number:

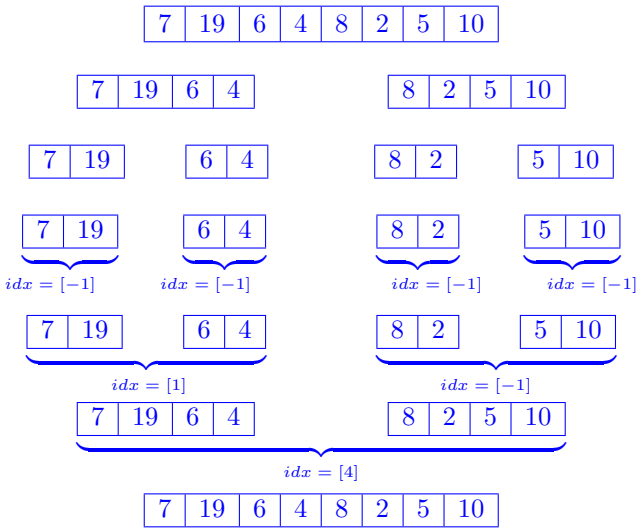
Exercise 3

Divide and Conquer

Consider an array `A[0...N-1]` consisting of N positive integers. Use the divide and conquer approach to find the index of `smallest apex element` in an array. An apex element is an element which is not smaller than its neighbours i.e. `A[i]` is an apex element if $A[i-1] \leq A[i] \geq A[i+1]$. For corner elements, you need to consider only one neighbour.

a) [1 point] What are the apex elements in array `A = [7,19,6,4,8,2,5,10]`.
19,8,10

b) [2 points] Draw a tree that illustrates the divide and conquer approach of determining the smallest apex element in array `A = [7,19,6,4,8,2,5,10]`.



c) [11 points] Write a C code to determine the index of `smallest apex element` for any input array of size N using **divide and conquer approach**. The complexity of your algorithm should be $O(N)$.

```
1  /*
2  * Function recursively only finds valid minimum apex
       elements, merges and checks the two middle elements, and
       checks corner cases only at the highest level.
3  */
4
5  int findApex(int A[], int low, int high, int n)
6  {
7    printf("low,high: %d,%d \n", low,high);
8    int smlstapex = -1;
9    if(high-low > 1)
10   {
11     /*
12     * DIVIDE IF SIZE IS LARGER THAN THREE
13     */
14     int mid = (low + high) / 2;
15     int lAIdx = findApex(A, low, mid, n);
16     int rAIdx = findApex(A, mid+1, high, n);
17     int check = mid;
18     printf("low, mid, high, lAIdx, rAidx: %d, %d, %d, %d, %d
          \n", low,mid,high,lAIdx,rAIdx );
19     /*
20     * MERGE AND CHECK MID BOUNDARY ELEMENTS
21     */
22
23     // check if mid is an apex candidate
24     if(A[check] >= A[check-1] && A[check] >= A[check+1])
25     {
26       if(lAIdx == -1 || A[lAIdx] > A[check])
27       {
28         lAIdx = check;
29       }
30     }
31     printf("low, mid, high, lAIdx, rAidx: %d, %d, %d, %d, %d
          \n", low,mid,high,lAIdx,rAIdx );
32     // check if mid+1 is an apex candidate
33     check = mid + 1;
34     if(high > mid+1 && A[check] >= A[check-1] && A[check] >=
          A[check+1])
35     {
36       if(rAIdx == -1 || A[rAIdx] > A[check])
37       {
38         rAIdx = check;
39       }
40     }
41     if(lAIdx == -1)
42       smlstapex = rAIdx;
43     else if(rAIdx == -1)
44       smlstapex = lAIdx;
45     else
46     {
```

```
47        if(A[lAIdx] < A[rAIdx])
48          smlstapex = lAIdx;
49        else
50          smlstapex = rAIdx;
51      }
52      printf("low, mid, high, lAIdx, rAidx: %d, %d, %d, %d, %d
            \n", low,mid,high,lAIdx,rAIdx );
53
54    }
55    /*
56    * CHECK CORNER CASES
57    */
58    if(high-low+1 == n)
59    {
60      // check corners
61      if(A[low] >= A[low+1])
62      {
63        if(smlstapex == -1 || A[low] < A[smlstapex])
64          smlstapex = low;
65      }
66      if(A[high] >= A[high-1])
67      {
68        if(smlstapex == -1 || A[high] < A[smlstapex])
69          smlstapex = high;
70      }
71    }
72    printf("smlst: %d \n",smlstapex);
```

code/Apex.c

Department of Informatics

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

University of Zurich[UZH]

Informatics II — Midterm 2
Spring 2021 — 03.05.2021

Name: _____ Matriculation number: _____

About Exam

You have 45 minutes to answer the questions; you have 15 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 15 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	Total
Points Achieved				
Maximum Points	20	5	15	40

Exercise 1

1.1 [3 points] Given an array A with n elements, the algorithm `BuildHeap(A,n)` taught in the lecutre converts A into a max-heap. Show for the following three cases the state of array A after `BuildHeap(A,n)`.

(a) A = [3, 6], n = 2.

(b) A = [4, 7, 1, 8], n = 4.

(c) A = [1, 7, 10, 2, 3], n = 5.

(a) $[6, 3]$

(b) $[8, 7, 1, 4]$

(c) $[10, 7, 1, 2, 3]$

1.2 [17 points] Consider a linked list defined as follows:

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

struct node* head;
```

(a) [12 points] Given a linked list `struct node* h`, implement the C function `struct node* rearrange (struct node* h)` that returns a new linked list where all elements whose `val` values are smaller than `h->val` are before `h` in the new linked list. Note that, `h` could be at any position in the new linked list returned by `rearrange` function.

```
1  struct node* rearrange(struct node* h) {
2    struct node *dummy1 = malloc(sizeof(struct node));
3    struct node *dummy2 = malloc(sizeof(struct node));
4    struct node * p1 = dummy1;
5    struct node * p2 = dummy2;
6    struct node *p = h;
7
8    while (p != NULL) {
9      if(p->val < h->val){
10       p1->next = p;
11       p1 = p1->next;
12     } else {
13       p2->next = p;
14       p2 = p2->next;
15     }
16     p = p->next;
17   }
18
19   p1->next = dummy2->next;
20   p2->next = NULL;
21
22   struct node *newh = dummy1->next;
23   free(dummy1);
24   free(dummy2);
25   return newh;
26 }
```

code/code01.c

(b) [2 points] What's the asymptotic complexity of your solution? Explain.

$O(n)$. All elements have to be visited.

(c) [3 points] Describe how to use `rearrange` function to sort a linked list such that `val` values are in an ascending order.

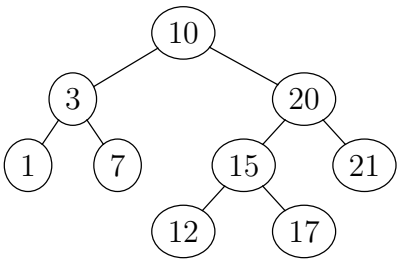
Call `rearrange` function and get the new head. Recursively call `rearrange` function for sub link list from the new head to `h`, and for sub link list from the `h` to the end of the sub linked list.

Exercise 2

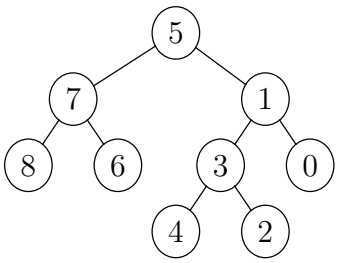
Consider a binary search tree with no duplicate values. Function `MaxRank` modifies the tree such that each node contains the number of values greater than that node in the binary search tree. The modified tree is the *Max-Rank Tree* for the binary search tree. A binary search tree is defined as follows:

```
1  struct TreeNode{
2    int val;
3    struct TreeNode* left;
4    struct TreeNode* right;
5  };
```

An example of Binary Search Tree and it corresponding *Max-Rank Tree* is given as:



Binary Search Tree



Max-Rank Tree

[5 points] Implement a C function `MaxRank` that transforms a binary search tree into its *Max-Rank 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.

```
1 int MaxRank(struct TreeNode* node, int rank){
2     if (node == NULL) { return 0; }
3     int rankR = MaxRank(node->right, rank);
4     node->val = max(rank,rankR);
5     int rankL = MaxRank(node->left,node->val+1);
6     return max(rankL,node->val+1);
7 }
```

code/Tree.c

Exercise 3

In a list of sorted numbers the median is the central number. If the list contains an odd number of elements, then the median corresponds to the value of the middle element. If the list contains an even number of elements, then the median is the average of the numbers from the two elements in the middle.

Example 1: A = [1,5,10,15,20], Median= 10
Example 2: B = [1,5,10,12,15,20], Median= 11

Given two stacks **A** and **B** of the following type:

```
1    typedef struct stackADT {
2      int *arr;
3      int size;
4      int top;
5    } stack;
```

[15 points] The task is to implement an `insert(stack *A, stack *B, int val)` function in C which inserts value `val` to either of the two stacks **A** or **B** in such a way that after *insert* function completes the following conditions hold.

1. The median value is the top element of stack A in case the total number of elements in both stacks is an odd number.
2. The median value is the average of the top elements of stack A and stack B in case the total number of elements in both stacks is an even number.
3. The difference between the number of elements in stack A and stack B must not be greater than 1.

   Note: You cannot use any additional data structure in the `insert` function. Both the stacks have the same size and consists of positive integers only. You can only insert the value in either of the stacks if there is sufficient space in stacks.

You can use the following stack operations in the `insert` function:

- **push**(stack *s, int val) - inserts `val` at the top of the stack s.
- **pop**(stack *s) - removes and returns the element at the top of the stack s. Returns -1 in case the stack s is empty.
- **peek**(stack *s) - only returns the element at top of the stack s. Returns -1 in case the stack s is empty.

- **size**(stack *s) - returns the size of stack s.
- **capacity**(stack *s) - returns the number of elements that can be added to stack s.

Hint: For the task the middle element(s) in sorted order need to be on top of the stack(s). Consider using the two stacks to maintain the sort order similar to insertion sort.

```
1  void insert(stack *A,stack *B,int val){
2    if(capacity(A)>(size(A)/2) || capacity(B)>(size(B)/2)){
3      if(capacity(A)!=size(A)){
4        if(val<peek(A)){
5          while(val<peek(A)){
6            push(B,pop(A));
7          }
8          push(A,val);
9        }else{
10         while(val>peek(B) && capacity(B)<size(B)){
11           push(A,pop(B));
12         }
13         push(B,val);
14       }
15
16       printf("Out: %d, %d\n",capacity(A),capacity(B));
17       while(capacity(B)-capacity(A)>1){
18         printf("1: %d, %d\n",capacity(A),capacity(B));
19         push(B,pop(A));
20         printf("1: %d, %d\n",capacity(A),capacity(B));
21       }
22       while(capacity(A)>capacity(B)){
23         printf("2: %d, %d\n",capacity(A),capacity(B));
24         push(A,pop(B));
25         printf("2: %d, %d\n",capacity(A),capacity(B));
26       }
27
28     }else{
```

code/Stack.c