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:

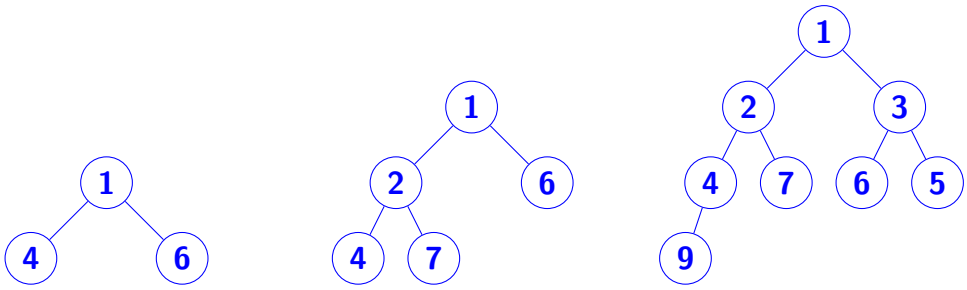
Correction slot — Please do not fill out the part below

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.

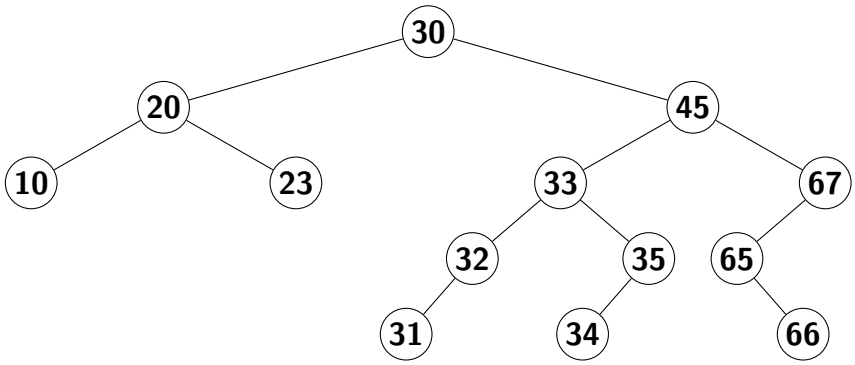
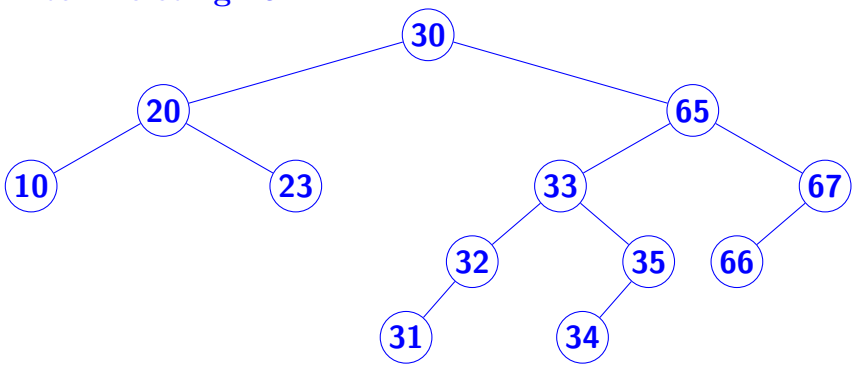


Figure 1: Binary Search Tree BST1

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

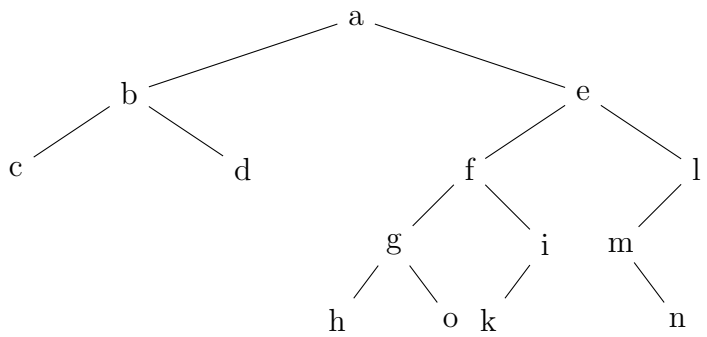
c. 9, 8, 63, 0, 4, 3, 2, 1, 45

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.

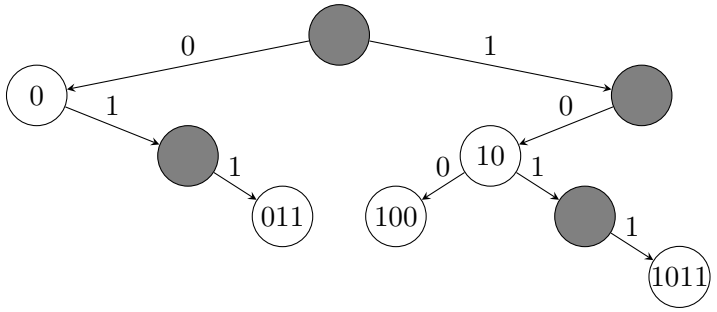


Figure 2: Radix Tree

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

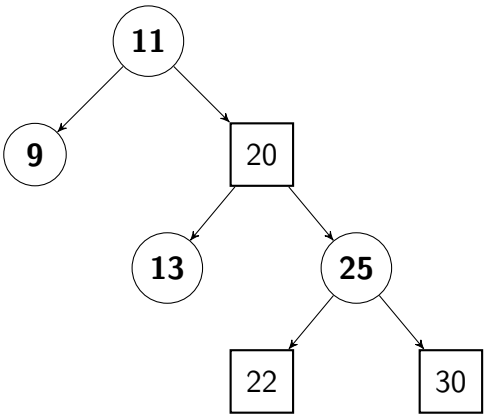
Consider that you are given `stack S` of integers. Use pseudocode to write a function `stack sortStack(stack* S)` that sorts the elements of `stack S` in ascending order. You are allowed to use auxiliary stacks in your implementation.

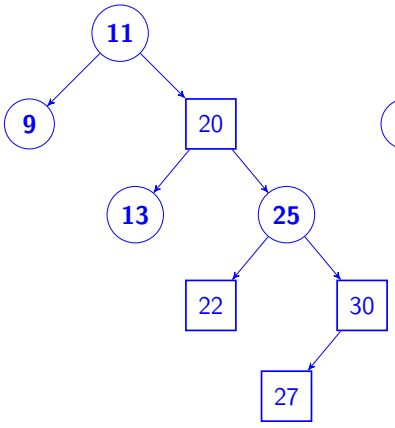
```
1 Algorithm: sortStack(S)
2 tmpStack = newStack();
3 while !isEmpty(S) do
4     tmp = pop(S);
5     while !isEmpty(tmpStack) and top(tmpStack) < tmp do
6         push(pop(tmpStack), S);
7     push(tmp, tmpStack);
8 return tmpStack;
```

Exercise 5 — 3+5 = 8 Points

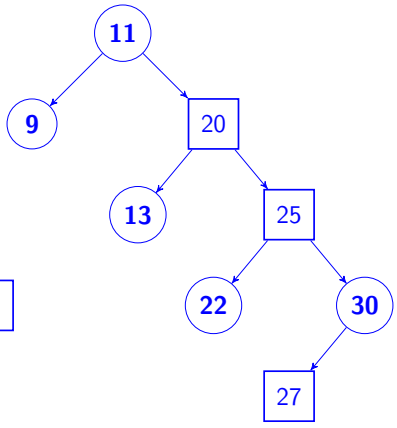
Red-black Trees

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.

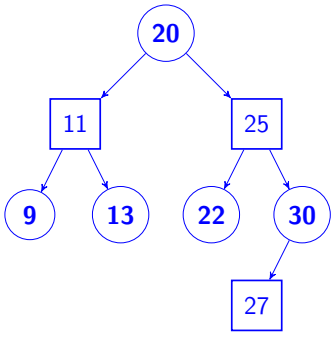




Case: 1 Mirrored



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;
}
```