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

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.

```c
struct node* rearrange(struct node* h) {
  struct node *dummy1 = malloc(sizeof(struct node));
  struct node *dummy2 = malloc(sizeof(struct node));
  struct node * p1 = dummy1;
  struct node * p2 = dummy2;
  struct node *p = h;

  while (p != NULL) {
    if(p->val < h->val){
      p1->next = p;
      p1 = p1->next;
    } else {
      p2->next = p;
      p2 = p2->next;
    }
    p = p->next;
  }

  p1->next = dummy2->next;
  p2->next = NULL;

  struct node *newh = dummy1->next;
  free(dummy1);
  free(dummy2);
  return newh;
}
```

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.

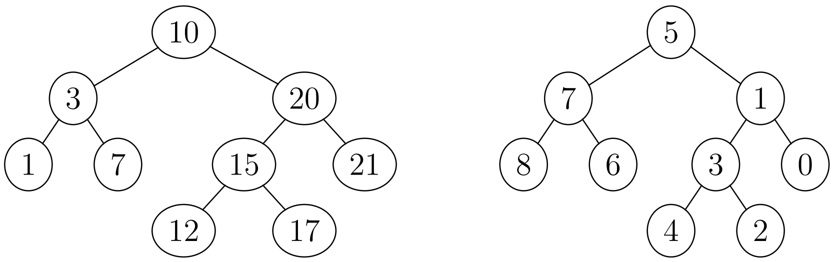
Call `rearrange` function and get the new head. Recursively call `rearrange` function for sub link list from the new head to `h`, and for sub link list from the `h` to the end of the sub linked list.

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:

```c
struct TreeNode{
  int val;
  struct TreeNode* left;
  struct TreeNode* right;
};
```

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.

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

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:

```c
typedef struct stackADT {
    int *arr;
    int size;
    int top;
} 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.

```
void insert(stack *A,stack *B,int val){
  if(capacity(A)>(size(A)/2) || capacity(B)>(size(B)/2)){
    if(capacity(A)!=size(A)){
      if(val<peek(A)){
        while(val<peek(A)){
          push(B,pop(A));
        }
        push(A,val);
      }else{
        while(val>peek(B) && capacity(B)<size(B)){
          push(A,pop(B));
        }
        push(B,val);
      }

      printf("Out: %d, %d\n ",capacity(A),capacity(B));
      while(capacity(B)-capacity(A)>1){
        printf("1: %d, %d\n ",capacity(A),capacity(B));
        push(B,pop(A));
        printf("1: %d, %d\n ",capacity(A),capacity(B));
      }
      while(capacity(A)>capacity(B)){
        printf("2: %d, %d\n ",capacity(A),capacity(B));
        push(A,pop(B));
        printf("2: %d, %d\n ",capacity(A),capacity(B));
      }

    }else{
```

code/Stack.c