

# Linnaeus University

Department of Computer Science and Media Technology

Diego Perez

## Exam in Algorithms and Advanced Data Structures, 1DV516, 4.5 cr.

Thursday 06 December 2018, 08.00–13.00

The exam consists of 4 questions. To pass you need 30 points.

Answer the questions in English.

When an algorithm is asked for, it should be understood that it should be as efficient as possible, and it should be expressed in such a way that it is understandable.

*Allowed aids:* Pen, pencil, eraser, and paper.

*Not allowed:* Consultation of any media outside of this document.

1. (a) Formally show that  $O(N^2)$  means the same as  $O(3N^2)$ . (2p)
- (b) Formally show that the time complexity of a search operation in an AVL tree is  $O(\log N)$ . (4p)
- (c) What is the minimum and maximum number of nodes *at depth*  $d$  in a binary search tree. The root is at depth 0. Motivate your answer. (3p)
- (d) Assuming that addition and print methods take constant time. Write the asymptotic time complexity of the following methods. Motivate your answer (for example, write the recurrence relations and solve them using one of the methods studied during the lectures). (6p: 2p each)

```
public void alg1(int n) {  
    if(n<=1){System.out.print(n);}   
    else {  
        for(int i=1; i<=n; i++) {  
            System.out.print(i+" ");  
        }  
        System.out.println(" ");  
        alg1(n/2);  
        alg1(n/2);  
    }  
}
```

(a)

```
public void alg2(int n) {  
    if(n<=1){System.out.print(n);}   
    else {  
        for(int i=1; i<=n; i++) {  
            System.out.print(i+" ");  
        }  
        System.out.println();  
        alg2(n/2);  
    }  
}
```

(b)

```
public int alg3(int n) {  
    if(n>=2){  
        return 2*(alg3(n-2)+alg3(n-2));  
    }  
    return n+1;  
}
```

(c)

- (e) In a max-heap with  $n$  distinct elements, where can the smallest element reside? And where can the largest element reside? Give both the location in the array and the location in the implicit tree structure. (5p)

2. The University registration system processes students applications. Students submit their applications to the system through a method *submit(StudentInfo information)*. The *StudentInfo* class includes, among other information: 1) the ID of the student (we assume that the ID an alphanumeric value which concatenates the student's surname, name, birthdate; for instance Berling-Tim-19890908), 2) a public method *getID()* which returns the ID value, 3) a **real** value *averageGrade* between 0 and 10 that represents the average of the student grades in his/her previous courses, and 4) a method *getAverageGrade()* which returns the *averageGrade* value.

Administrators at the University obtain the next application to process through method *retrieveNext()*, which gives them an object of class *StudentInfo*. In the following,  $N$  represents the number of applications in the system; that is, the number of applications that have been submitted to the system but have not been processed by administrators yet. The University gives more priority to the registration of students with higher average grade. Therefore, method *retrieveNext()* returns the student application that has the highest *averageGrade* value among the  $N$  applications in the system.

During the time interval between the moment when the student submits his/her information and the moment when the information is retrieved by an administrator, administrators can check the existence of an application. There is a method *approximateSearch(String partialID)* that returns the lowest and highest IDs in the system which match with the partialID. For example, if the system contains  $N=4$  elements, with IDs [Fry-Philip-19800101, Simpson-Bart-18850505, Skinner-Seymour-19600606, Sinclair-Jack-19730808], then:

*approximateSearch("S")* returns [Simpson-Bart-18850505, Skinner-Seymour-19600606]  
*approximateSearch("Si")* returns [Simpson-Bart-18850505, Sinclair-Jack-19730808]  
*approximateSearch("Sin")* returns [Sinclair-Jack-19730808]

- (a) Describe a solution for the University registration process where all the three *submit*, *retrieveNext*, and *approximateSearch* operations execute in logarithmic time. You must motivate that your solution works and argue that your solution actually executes in  $O(\log N)$ . You do not need to explain the internal execution of operations that you have studied during the lectures; you can just use them.

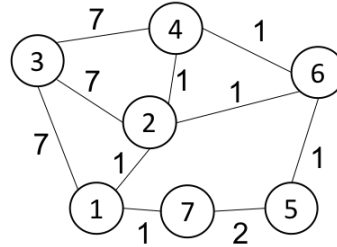
(14p)

- (b) The **Advanced Data Structures expert** (ADSe) who is in charge of the registration system receives a system upgrade request. Now, at any point in time administrators would like to obtain an array with the  $N$  applications in the submission system. Moreover, administrators would like to receive this array sorted by the *ID* value of *StudentInfo* class. For this, they will use a method *getApplicationsSortedByID()*. This new method *getApplicationsSortedByID()* should run in  $O(N)$ .

Describe an extension to your previous solution that includes the operation *getApplicationsSortedByID()* and satisfies the mentioned requirements: it executes in  $O(N)$  and the three other operations still run in  $O(\log N)$ . You must motivate that your solution works and argue that your new solution actually has the required time complexity.

(6p)

3. Given a weighted, undirected graph  $G = (V, E)$  with no negative-weight in edges. The *diameter* of the graph is the maximum-weight shortest path between any two vertices in the graph. For example the diameter of the graph in next figure is 9 (weight of path between vertices 3 and 5). Give a polynomial-time algorithm for finding the diameter of the graph. What is the running time of your algorithm? Does it matter whether it's a dense or sparse graph? Motivate! (5p)



4. Suppose there is a coffee machine, which works with cash, and that you have to implement the internal software that computes the coins returned to the customer as change. Given the amount of money to return  $M$ , you have to implement a method that always returns the minimum number of coins.

The typical *change-making problem* addresses the question of finding the minimum number of coins (of certain denominations) that add up to a given amount of money. The most appropriate algorithmic technique to use for solving this problem depends on the set of values of coins.

- (a) Assuming that we can use only 3 types of coins, with values 1, 5, and 10; given array  $Coins = [coin_1, \dots, coin_C] = [1, 5, 10]$  and the amount of money to return  $M$ , write an algorithm that executes in  $O(C)$  time (see that  $C$  is the number of elements in array  $Coins$ ) that computes the minimum necessary number of coins. *Note 1:* You do not need to compute the concrete number of coins of each type, just compute the total number of coins. For instance, for  $M=16$  you should return the value 3 (a coin of value 10, a coin of value 5 and a coin of value 1). *Note 2:* Assume that all arithmetic operations “+”, “-”, integer multiplication, integer division, and modulo operation execute in constant time (4p)
- (b) Assuming that we can use only 4 types of coins, with values 1, 5, 8 and 10; given array  $Coins = [coin_1, \dots, coin_C] = [1, 5, 8, 10]$  and the amount of money to return  $M$ , write an algorithm that computes the minimum number of coins that are necessary and executes in  $O(M \cdot C)$  time. *Note 1:* You do not need to compute the concrete number of coins of each type, just compute the total number of coins. For instance, for  $M=16$  you should return the value 2 (two coins of value 8). *Note 2:* Dynamic Programming strategy can be applied to this problem. *Note 3:* Assume that all arithmetic operations execute in constant time. (8p)
- (c) Given the same information  $(M, [coin_1, \dots, coin_C])$  and the decision problem, “*Is there a way in which I can give the change using less than  $S$  coins?*”, to which class can you say that the decision problem belongs? (P, NP, NP-complete, NP-Hard). You must motivate your answer. (3p)