Linnaeus University
Department of Computer Science and Media Technology
Diego Perez

Exam in Algorithms and Advanced Data Structures, 1DV516, 4.5 cr.
Thursday 31 October 2019, 14.00–19.00

The exam consists of **3** questions. To pass you need 30 points.
Answer the questions in English.
Answer the questions using a pen. Do not use a pencil for your final answers.
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, and eraser.
Not allowed: Consultation of any media outside of this document.

1. (a) Formally show whether $O(\log n)$ means the same as $O(\log(n^2))$ (2p)

(b) Formally show that the time complexity of a search operation in a red-black tree is $O(\log N)$. (4p)

(c) Explain why an algorithm of time complexity $O(n \log n)$ might not always be faster than one of time complexity $\Theta(n^2)$. (3p)

(d) Assuming that addition, multiplication and print methods take constant time. Write the asymptotic time complexity of the three 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)

If you cannot give the Θ time complexity, give for each method the most restrictive O time complexity you can that is correct (for example, $O(1)$ is more restrictive than $O(log\ n)$; in turn, $O(log\ n)$ is more restrictive than $O(n)$, etc.).

```java
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)

```java
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)

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

Optional (for another 2p)

```java
public int alg4(int n) {
    if(n<=1){return 1;}
    else {
        int sum=0;
        for(int i=1; i<=n; i++) {
            sum+= alg4(n-1);
        }
        return sum;
    }
}
```
(Optional)

2. You have to propose Data Structures and the Algorithms for the system of a company that sells tickets for a sport event. The company stores the people who want to purchase tickets in a *set of prioritized public*. There might not be enough tickets for all the people who want to purchase one of them. Therefore, the company prioritizes the people. Each person in the set is represented as a tuple `<key, priority>`. Assume that everybody in the city has a different name, which allows you to use their names as `key`. The `priority` is the priority of the person who wants to buy a ticket (people have different priorities for buying tickets, for instance: league player, VIP, club member, club staff, children, general public, visitor public,...) and it is represented as an Integer value. There can be more than one person with the same priority who want to purchase a ticket. The amount of different priorities in the system is not limited. The higher the `priority` value of a person, the higher the priority of the person to purchase a ticket. The company needs to execute the following operations in their system.

- When a person with a given priority applies for a ticket, the system executes $insertPerson(key, priority)$. This operation inserts a new person with *key* (that is, his/her name) and *priority* values in the ticket system.
- When the company administrators want to check who is the person with maximum priority in the system, they execute $getPersonWithMaximumPriority()$. This operation returns the *key* of a person with maximum priority in the system (it does not delete the person from the system).
- When the person with maximum priority gets his/her ticket assigned, administrators immediately execute $deletePersonWithMaximumPriority()$. This operation deletes the person with maximum priority from ticket system.

  Note: Other people can still apply for tickets after some of the tickets have already been assigned and some people already deleted from the system. However, a person who has got his/her ticket and has been deleted from the system will NEVER have his/her ticket cancelled, even if another person with higher priority applies for a ticket later.
- People who are in the system (that is, they have been inserted and they have not been deleted yet) can swap their priority values with other people in the system. Sometimes a person wants to transfer their priority to another person (for instance, a VIP person could want to transfer his/her priority in the system to a child who wants to watch the sport event). In that case, the system executes $swapPriority(key1, key2)$. This operation swaps the priority values of applicants with keys $key1$ and $key2$.

  Example: after operations *insertPerson(`"Alice", 5`)*, *insertPerson(`"Bob", 10`)*, *swapPriority(`"Alice","Bob"`)*,

  the system contains `<"Alice", 10>` and `<"Bob", 5>`

It happens frequently that two or more people in the system have the same priority. In this case, operations *getPersonWithMaximumPriority()* and *deletePersonWithMaximumPriority()* must return or delete, among the people in the system with maximum priority, the person who has been the longest time in the system with his/her current priority value.

Example: after operations *insertPerson("Alice", 5)*, *insertPerson("Bob", 10)*, *insertPerson("Carol", 10)*, *swapPriority("Alice","Carol")*,

operation *getPersonWithMaximumPriority()* should return `"Bob"`. Even though `"Alice"` arrived to the system first, the selected person is `"Bob"` because he has been more time in the system with priority `10`.

Hint: Keep in mind that the Priority Queue ADT based on heaps that you studied during the lessons DOES NOT ensure the ordering of elements with the same priority. For example, using directly the Priority Queues you saw during the lessons, after the following code:
insertPerson("Alice", 5), *insertPerson("Bob", 10)*, *insertPerson("David", 5)*, *deletePersonWithMaximumPriority()*, it is NOT ensured that a next operation *getPersonWithMaximumPriority()* returns `"Alice"`.

(a) Describe a solution for the system where all the four *insertPerson, getPersonWithMaximumPriority, deletePersonWithMaximumPriority,* and *swapPriority* operations execute at most in O(log N), where N is the number of people in the system. 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. (10p)

(b) You get 4 additional points for each of the four operations in the previous exercise that can execute in average faster than O(log N) . You must describe your solution, motivate that each operation works as expected, and argue that they execute faster than O(log N) in average. For example, let P be the number of different priority values in the system in a given moment, an operation which executes in O(1) or O(log P) is considered faster than O(log N). (note that $P \leq N$ is always true in this problem). (16p: 4p for each improved operation)

3. Suppose that you live in a neigborhood with N+1 houses. You have N neighbors, from n_1 to n_N. Your own house is n_0. You know the distances between each pair of houses in the neighborhood; $d(n_i, n_j)$ $\forall i, j \in [0..N]$ such that $i \neq j$; and $\forall i, j \;\; d(n_i, n_j) = d(n_j, n_i)$.

(a) Suppose that tonight is Halloween[1]. Assume that, despite your age, you want to bother ALL your neighbors with the "trick-or-treat" game. Each neighbor n_i will give you some units of candy c_i. Assume that you would like to walk as less as possible during your visits to all houses and collect all the candy ($\sum_{i=1}^{N} c_i$). Therefore, you would like to visit all your neighbors' houses and come back to your house walking the minimum possible distance.

In your Algorithms course you have studied a problem similar to this, the Traveling Salesperson Problem. Therefore, you know that nobody knows any efficient algorithm for calculating this minimum walking distance. However, you have also studied that you can give an approximate solution to this problem in polynomial time. Assume that walking double than the minimum possible distance is still a *good enough* solution for you. Assuming that the minimum distance to walk is M, give an algorithm in **pseudocode** that executes in polynomial time and computes a solution that requires, at most, walking $2M$ distance. The inputs to the algorithm are: the N+1 houses and all the distances $d(n_i, n_j)$. The output of the algorithm is a sequence of houses, starting and finishing in n_0, that represent a path of houses. Explain the time complexity of the algorithm. Argue that the algorithm produces a solution which requires, at most, walking double than the minimum distance. (8p)

(b) You have a friend who knows that you do not like walking. Assume that your friend lends you a *magic bicycle* which allows you to move from house to house immediately (in zero time). However, your friend only lends the magic bicycle to you for a limited time of T minutes. You know that each neighbor will want to chatter with you some minutes before giving his/her candy to you. Assume that you know how many minutes of chatter you will have to bear from each neighbor (each neighbor n_i blocks your candy-collection activity for t_i time). In this problem you do not need to worry about the distances, but you need to worry about the maximum amount of candy you can collect. Give an algorithm that executes in $O(N \cdot T)$ time and that, given the arrays $[c_1, ..., c_N]$, $[t_1, ..., t_N]$, and T, returns the maximum amount of candy you can obtain in time T if you visit the most convenient subset of neighbors. *Note 1:* You do not need to compute the concrete subset of neighbors that will give you the maximum amount of candy, just compute its value. *Note 2:* Dynamic Programming strategy can be applied to this problem. (8p)

(c) Given the same information as in the previous problem ($[c_1, ..., c_N]$, $[t_1, ..., t_N]$, and time limit T) and the decision problem, *"Is there any way in which I can obtain more than C units of candy?"*, to which class can you say that the decision problem belongs? (possible classes are P, NP, NP-complete, NP-Hard). You must motivate your answer. (3p)

[1]Halloween is an event where, in some cultures, young people put on a costume and visit their neighbors and they expect to receive candy from them.