Computer Engineering Department
PhD Qualifier Exam

11 September 2015

- Time: 09:30 - 12:30
- The exam contains **8** questions in total. You are to **choose 6** of these questions and answer **only** them.
- By submitting this exam, you agree to fully comply with Koç University Student Code of Conduct, and accept any punishment in case of failure to comply. If you do not submit this exam on time, you will directly fail the qualifier.
- The exam is **open-book** and **open-notes**.
- You are **not** allowed to use any electronic equipment such as computers and mobile phones.
- In general, you are **not** allowed to ask questions during the exam.
- Good luck!

1. Consider the following pseudocode:

```
function Compute(n : nonnegative integer)
if n = 0 then
    p = 1;
else if n = 1 then
    p = 2;
else
    p = Compute(n − 1)*Compute(n − 2);
return p;
```

   a) Verify **by induction** that this algorithm computes $2^{f(n)}$, where $f(n)$ is the n-th Fibonacci number. (Recall that $f(n) = f(n-1) + f(n-2),\ n \geq 2$, and $f(0) = 0,\ f(1) = 1$.)

   b) Write down the complexity function $T(n)$ of the above algorithm as a recurrence relation.

   c) Comment **as precisely as possible** on the worst-case complexity. Explain your answer.

2. Suppose that you are given a `Point` class with two integer fields `x` and `y` representing the x and y coordinates of a point in the two-dimensional plane. Use an object-oriented language you are comfortable with to write the `Point` class and answer the following questions.

   (a) Write a method `in3rdButNot1stOr2nd` that takes as input an *ordered list* `lst1` of `Point` objects, a *set* `s2` of `Point` objects, and a collection `c3` of `Point` objects. Recall that lists and collections may contain multiple `Point` objects with exactly the same coordinates. The `in3rdButNot1stOr2nd` method should return a *set* of `Point` objects `result` such that every `Point` object in `result` is in `c3` but *not* in `lst1` and *not* in `s2`.

   (b) Devise a type (such as an interface, an abstract class in Java or an abstract class with a pure virtual function in C++) named `T`. For instance, in Java, the type `T` may be `Comparable<Point>`. The type `T` should be such that

   - `Point` is a subtype of `T`, and
   - this fact forces `Point` to have a method `compareTo` that takes another `Point` object `other` as argument, and returns an integer based on whether the `Point` object `this` comes before (returns a negative integer), after (returns a positive integer), or is the same as the `Point` object `other` (returns 0).

   Implement the `compareTo` method such that `Point` object `p1` precedes `Point` object `p2` if `p1.x` is smaller than `p2.x`. If the x coordinates of the two objects are equal, then `p1` precedes `p2` if `p1.y` is smaller than `p1.y`. Otherwise the two `Point` objects are equal.

3. **a) Process synchronization.**

Suppose processes P, Q, R and S execute concurrently, and there exists a shared data item d. Processes perform the following operations (update or read) on data item d, once during their execution.

- Process P updates d (in code part A).
- Process Q reads d (in code part B).
- Process R reads d (in code part C).
- Process S updates d (in code part D).

Show how these processes can use **semaphores** to ensure that the code parts are always executed in the following order: A, C, D, B.

Write the pseudo-codes for the processes showing how the semaphores are used and their initial values.

b) Virtual memory.

Given the reference string 5, 3, 4, 3, 1, 2, 1, 4, 3, 4, 5, 3, 2 which indicates the sequence of page accesses by a process, consider the following scenarios for **page replacement algorithms**. Assume the number of frames allocated for the process in the physical memory is four.

- How many page faults will occur with **LRU (Least Recently Used)** page replacement algorithm? Remember all frames are initially empty, so your first unique pages will cost one page fault each. Show the contents of frames for each page access, and indicate newly loaded pages.

- How many page faults will occur with the **Optimal** page replacement algorithm? Remember all frames are initially empty, so your first unique pages will cost one page fault each. Show the contents of frames for each page access, and indicate newly loaded pages.

4. Database Design: you are asked to model a database that contains information on researchers, academic institutions, and collaborations among researchers. A researcher can either be employed as a professor or an assistant. The following should be stored:

   - For each researcher, his/her name, year of birth, and current position (if any).
   - For each institution, its name, country, and establishment year.
   - For each institution, the names of its schools (e.g. Humanities, Engineering, Medical School ...). A school belongs to exactly one institution.
   - An employment history, including information on all employments (start and end date, position, and what school).
   - Information about co-authorships, i.e., which researchers have co-authered a research paper. The titles of common research papers should also be stored.
   - For each professor, information on what research projects (title, start date, and end date) he/she is involved in, and the total amount of grant money.

   a) Draw an E/R (entity relationship) diagram for the data set described above. Make sure to indicate all cardinality constraints specified above. The E/R diagram should not contain redundant entity sets, relationships, or attributes. Also, use relationships whenever appropriate. If you need to make any assumptions, include them in your answer.

   b) Write an SQL query to find the total number of active projects from School of Engineering at Koç University.

5. Given an array A of n numbers, we call a number x the *frequent element* of A if it occurs at least $0.7n$ times in A (i.e., for at least 70% of values for $1 \leq i \leq n$ we have $A[i] = x$). Note that not all arrays have frequent elements and each array can have at most one frequent element. The *mode* of A is the value that occurs most frequently in A.

   - Give an algorithm that finds the *mode* of A in time $O(n \log n)$.
   - Give an *expected* $O(n)$ time *deterministic* algorithm that finds the *frequent element* of A or reports that none exists, but always returns the correct answer.

   For these algorithms, you do not need to provide a fully-working pseudocode. Instead, you can coarsely describe the algorithm.

6. **Data Structures**

Consider the following ADT for a binary search tree with an additional method, rank. For the sake of simplicity, the tree will keep only integer values with no duplicates.

```
public interface BST {
    // usual BST operations
    boolean isEmpty();
    void insert(int x);
    void remove(int x)
    int find(int x);

    // rank returns the number of elements that are smaller than x
    int rank(int x);

}
```

and its AVL tree implementation (partial) below:

```
public class Node {
     int data
     Node left;
     Node right;
     // ... and more elements,
     //     which are not needed to specify for this question
}

public class AVLTree implements BST {
     Node   root;
     // ..... implementation of the constructor, insert, remove, find etc.

     // in part 1 , you are asked to implement the rank operation
     int  rank(int x) {}
 }
```

Answer the following questions based on the BST interface and AVLTree implementation:

a) Implement the rank operation. If the rank of x is r, then your method should run in $\mathbf{O(r)}$ time worst case. You can use java-like pseudocode. You can have helper (private) methods if needed.

```
public int rank(int x) {
```

b) Consider the following problem: given a collection of intervals and an integer value q, a *count* query is the number of intervals that contains q. We want to design a data structure to perform the count query. For example if the intervals are (1,5), (1, 7), (3,4), then count(5) would return 2.

```
public class Interval {
        // .. some variables you might need here



        void insert(int xmin, int xmax);  // inserts an interval
        int count(int q); // returns the result of the count query
}
```

You are asked to describe how to implement insert and count methods using the AVLTree data structure (of Part 1). If there are n intervals, both insert and count should run in $\mathbf{O}(\log \mathbf{n})$ time worst case. Explain how you use AVLTree and provide a pseudocode for the methods insert and count.

7. The random variable X is uniformly distributed in the interval $[0, a]$. Suppose a is unknown, so we estimate a by the maximum value observed in n independent repetitions of the experiment; that is, we estimate a by $Y = max\{X_1, X_2, ..., X_n\}$.

   (a) Find the cumulative distribution function (CDF) function of Y, $P[Y \leq y]$.

   (b) Find the mean and variance of Y. Explain why Y is a good estimate for a when n is large.

8. The concept of state is central to programming languages. However, most functional programming paradigms do not have direct support for state.

   a) Describe how you would add support for state through functions. Assume no explicit support for pointers.

   b) Provide a diagram demonstrating the mechanism described in Part (a) in play for a simple scenario. Explain.