# Cortica Testing Questions

| Revision | Date | Modified by | Comments |
|---|---|---|---|
| **1.0** | March 2016 | Yuval Woldman | |

## Table of Contents

## Overview

The following document lists questions to be asked Cortica Employee candidates.

### Important notes:

- Use coherent names in code.
- Make sure code compiles.
- For each class implement a test application which runs it.
- Deliver fully functioning application and the source code.
- Keep OOP in mind when designing the code.
- Keep performance in mind when designing the code.
- Keep boundary cases in mind when designing the code.
- Always validate user input.
- Handle exceptions.
- Use coherent notifications and readouts.
- Use any of the following programming languages:  C# / C++ / Java / Python
- All applications should be repeatable (Given the same input, the application should behave exactly the same always, and produce the same output).

# Task #1: Multithreaded matching

## Scenario:

We have 2 directories which contain CSV files.

We need to be able to determine which CSV files in the 1st directory are similar to CSV files in the 2nd directory.

## Definitions:

- Two directories, each containing CSV files.
- Each CSV files contains a sorted (Ascending) set of integer numbers (1 row only).
- A CSV file is considered “similar” to another CSV file if both files contain at least X of the same numbers.

## Desired function:

We want to copy all CSV files from directory “A”, which are similar to at least 1 csv file in directory “B”, to directory “C”

## Instruction:

Write a class which handles, in multi-threading, the loading, “similarity” testing and copying of files

The class should have a method which returns VOID and receives 3 strings (A,B & C, each a directory path) and an int (X, minimal amount of equal numbers).

# Task #2: Rush Hour

## Scenario:

In our server farm, there are certain times of day in which we get a higher number of requests. During these time-spans we want to respond by allocating more resources, to give better service. Outside these time-spans we want to remove resources, to save on energy.

We need to be able to know which times of day are such "Rush Hours"

## Instruction:

Write a class which can receive time-spans that are defined as "Rush Hours", remember them, and can also answer queries about a specific time of day - whether it is considered rush hour or not.

The class provides the following interfaces:
1. void AddTimeSpan(float start_time, float end_time);
2. boolean IsRushHour(float time);

## Assumptions:

- When the class is initialized, there are no rush hours at all. The user can then add time-spans of rush hours using the "AddTimeSpan" interface. The user can at any time query about a specific time using the "IsRushHour" interface.
- Assume that "IsRushHour" is called very frequently, and "AddTimeSpan" is called less frequently.
- Assume that the system will live forever, and performance should not degrade over time.
- float T is a valid time of day if (T>=0.00 and T<24.00). You can not assume anything about the input. You must support any valid time.
- If the input is invalid (meaning T<0 or T>=24.00), behvaiour is not defined.
- IMPORTANT: The internal representation of time spans must be minimal. For example, if the same time span is added twice, there should be no change to the internal data structures.

## Usage Example:

- Init() // System is initialized in an empty state
- IsRushHour(3.00) --> False
- IsRushHour(5.00) --> False

- AddTimeSpan(2.00, 4.00)
- IsRushHour(3.00) --> True
- IsRushHour(5.00) --> False

- AddTimeSpan(7.00, 9.00)
- IsRushHour(3.00) --> True
- IsRushHour(5.00) --> False
- IsRushHour(7.00) --> True
- IsRushHour(11.00) --> False

- AddTimeSpan(8.00, 12.00)
- IsRushHour(3.00) --> True
- IsRushHour(5.00) --> False
- IsRushHour(7.00) --> True
- IsRushHour(11.00) --> True

## Bonus:

How would your solution be different if we changed the definition of "time" to be an integer, representing the time of day in seconds? (answer in words)