HITFOX GROUP ENGINEER CASE STUDY

Task A: Playing with data structures

- Given is an unordered list of numbers (which can appear repeatedly), and we would like to generate a data structure which has the following structure:

```
DivisorsHash { key: 'number' => value: 'divisors of the number in the list' }
```

- Moreover, there some extra requirements.
  - Keys must be stored ordered in the data structure, in a way that when iterated over its keys they will be returned in an ordered manner.
  - List of values for each key must be ordered.

```
Example Input:
[7,4,2,10,3,6,4,5]

Example Output:
{
      2 => [1],
      3 => [1],
      4 => [1,2],
      5 => [1],
      6 => [1,2,3],
      7 => [1],
      10 => [1,2,5],
}
```

- Code the defined data structure and write an algorithm which, given a list of numbers, returns a filled instance of it. Tasks:
  - Write class *DivisorsHash*.
  - Write *DivisorsHash generate(...)* method on class *ProblemA*.
  - Although the example code is written in Java, you can build the solution in the language you feel most comfortable, even in Pseudocode.
  - Do not write long comments, we will talk about the solution afterwards :).

```
public class DivisorsHash {
      // TODO
}

public class ProblemA {

      public static DivisorsHash generate(List<Integer> inputNumbers){
            // TODO
      }
}
```

- Advices
  - In order to provide an optimal solution, it is important to choose appropriate standard data structures as attributes of class *DivisorsHash*. We recommend to write the solution in Java (in order to use its standard data structures) but if you don't want to you can use predefined data structures of the language you use.
  - First, focus on an easy solution that works and, if you still have time, go for a better performance-driven one (you can provide a few solutions).

Task B: Find shortest path

- Given is a directed & weighted graph defined by the following interfaces:

```
public Interface IGraph {
      public Set<INode> getNodes();
}

public interface INode {
      public String getName();
      public List<IEdge> getEdges();
}

public interface IEdge {
      public INode getOriginNode();
      public INode getTargetNode();
      public int getWeight();
}
```

- Code an algorithm which finds the shortest path between two given nodes. Tasks:
  - Fill class *Path*.
  - Fill method *IPath getShortestPath(...)* on class *ProblemB*.
  - Although the example code is written in Java, you can build the solution in the language you feel most comfortable, even in Pseudocode.
  - Do not write long comments, we will talk about the solution afterwards :).
  - Do not add more code on interface *IPath*, and don't write more classes.

```
public interface IPath {
      public List<Node> getPath();
      public int getTotalWeight();
}

public class Path implements IPath(){
      // TODO
}

public class ProblemB {

      private IGraph graph;

      public ProblemB(IGraph graph){
            this.graph = graph;
      }

      public IPath getShortestPath(INode initNode, INode endNode){
            // TODO
      }
}
```

- Advices
  - Spend the first five minutes understanding the problem, and make up and draw a graph example to help you visualizing the structure.
  - Do not only implement the required methods on class *Path*, code as many helper methods as you need (i.e.: Methods to add nodes, to remove nodes, etc).
  - First, focus on an easy solution that works and, if you still have time, go for a better performance-driven one (you can provide few solutions).

Task C: SQL

- Concept: We have a client that has an outstanding mobile app and he wants to make some users download it. For that purpose, we are advertising that app displaying fancy banners inside some other applications. When we achieve that a user clicks on an ad (which rarely happens), he is redirected to a page where he can get it. If he finally downloads the advertised app (which does not happen so often), then we have succeeded in our goal, so we are paid by the owner of the app.

- Given is the following two relational tables and an example subset of their content:
  - CLICK: Information about the users that have clicked on an ad.
  - CONVERSION: Information about the users that, after clicking, finally downloaded the app (1 conversion maximum per click).

TABLE: CLICKS

click_id	timestamp	country_code	device
...	...	...	...
34545	2013-11-22 23:58:01	DE	ios
34546	2013-11-22 23:58:24	US	android
34547	2013-11-22 23:59:04	US	ios
34548	2013-11-23 00:00:01	US	ios
34549	2013-11-23 00:00:33	DE	android
...	...	...	...

TABLE: CONVERSIONS

conversion_id	click_id	timestamp	amount
...	...	...	...
167	34546	2013-11-23 00:10:03	1.23
158	34546	2013-11-23 02:56:32	0.27
...	...	...	...

- Perform the following queries:
  - List successful click ids (the ones that resulted in a conversion) for 'ios'.
  - Calculate the number of conversions achieved by country and device on November 22nd.
  - Calculate the revenue by device for 'US'.

- The following query returns the click and conversion ids that have resulted in a conversion, but not all. Why? How would you rewrite it to get all clicks in the same query?

```
SELECT A.click_id, B.conversion_id
FROM CLICKS AS A, CONVERSIONS AS B
WHERE A.click_id = B.click_id
```

- How would you write the following query without a subquery?

```
SELECT COUNT(*), device
FROM CLICKS AS A
WHERE A.click_id NOT IN (SELECT B.click_id FROM CONVERSIONS AS B)
GROUP BY device
```