### Web Application

Using the web application framework of your choice, write a web application that plays a number guessing game. The home page should ask the user to enter a number between 1 and 100. The user should be able to enter a guess and submit it to the web server. The server should respond by telling the user that the number is correct, or that the number is too high, or that the number is too low. If the last guess was not correct, the user should be allowed to guess again based on the new information. The application should track the number of guesses the user has made and display it to the user. The application should be able to gracefully handle guesses that are not numbers, are outside of the valid number range, etc.

Mike F.: See .html, js & css code files. This web app sends ajax PUTs of game moves to a webservice endpoint. This endpoint is not coded, since this was not in the spec, but the app includes a mock library, so the app should work normally in the absence of a working endpoint and should not throw any errors.

After writing the application, answer the following questions. Document your reasoning.

1. What is the best strategy for playing this game?

Mike: “Divide and conquer” binary search algorithm, halving the remaining space after each guess.

1. If that strategy is employed, what is the maximum number of guesses required in the worst case to guess the correct number?

Mike: O(log n), or 8 steps w/ 100 possible numbers.

### String Manipulation

Write a command line program in the language of your choice that performs a character frequency count on its input. The input text can be read from a command line argument. The output should be a list of characters and their frequencies sorted from highest to lowest.

Example Input: "This is some input"

Example Output:

i 3

s 3

t 2

h 1

o 1

m 1

e 1

n 1

p 1

u 1

Mike F.: See Program.cs code file.

### Database

**Use the attached schema to complete the following questions. Assume a SQL Server 2008 R2 database. Some assumptions about the database will have to be made to complete this exercise. Please note all assumptions.**

1. Create a T-SQL stored procedure that returns a listing of total revenue in 2011 by product for each sales rep in the Domestic Sales department. For example:

Bob ProductA $5000

Bob ProductB $10000

Jen ProductA $2000

Tim ProductB $1000

Tim ProductC $30000

Mike F.: See .sql file for sp code

2. Write a T-SQL script to alter the database structure and/or implement indices to maximally increase performance of the stored procedure you created in part 1. Detail server configuration settings that would impact the performance of your stored procedure (if applicable).

Mike F.: The short answer is that I would make sure there are indexes on:

product

id

sales\_order\_items

id & line\_id

id

sales\_order

order\_date

id

sales\_rep

employee

emp\_fname

emp\_id

I would also adjust down the isolation levels/lock hints, as this report data does not have to be immediately consistent, i.e. eventually consistent is acceptable, esp. for prior years.

3. Assume that usage of the database (both CRUD operations and calls to your stored procedure) has increased from 10 operations per hour to 10,000 operations per hour. Usage is expected to continue growing. Document your strategy for scalability, outlining specific technologies used and/or changes to the database design needed to accomplish your plan.

Mike F.: Things I would check/test/POC include:

* Server hardware, including: 64bit OS, RAM, Disk I/O, Disk RAID Levels, data & logs on separate partitions, Write-ahead cache with battery backup, SSD drives, gigabit ethernet, etc.
* Caching at the application layer, e.g. .Net cache, memcached
* Separating out OLTP/transactional vs Reporting queries into separate databases, the latter denormalized and populated with asynchronous jobs/queues.
* Profile and survey queries for use of stored procedures (as opposed to in-line SQL, indexes, query optimizations, etc.)
* Consider the use of NoSQL (e.g. MongoDB, Redis, DynamoDB), esp. for reporting data and/or data that can be eventually consistent, data that does not need to be normalized
* Consider sharding (e.g. Distributed Partitioned Views, Azure Federation, MongoDB sharding)