1)
Vemos que sintácticamente las operaciones son idénticas y no se podría saber si la operación es local o remota dado que el los nombres de las operaciones son exactamente los mismos. sin embargo a la hora de llamar a la clase hay que hacer un lookup del nombre de esta, al no estar en el espacio de nombres local, teniendo el cuenta el hostname remoto.
Por lo que si bien a nivel invocación de métodos nunca no enteramos si es local o remoto y no se hace ninguna diferencia en la invocación de ambos, a la hora de obtener la clase remota si.
RMI si ofrece una transparencia de acceso al permitir invocar tanto un procedimiento local como remoto de la misma manera. Pero esa transparencia no es total ya que hay que tener en cuenta otras cuestiones como el lookup de la clase remota.

b)
StartRemoteObject: presente en el servidor, se encarga de la instanciación y registro de la clase invocable en el espacio de nombres.
IFaceRemoteClass: presente en el servidor, es la interface que define los métodos y la clase del servidor
RemoteClass: presente en el servidor, implementa los métodos y la clase de la IFaceRemoteClass (interface)
AskRemoteClass: presente en el cliente, hace los llamados a los métodos remotos

2)
Diferencias clave entre RPC y RMI:

- RPC admite paradigma procedural, porque está basado en C, mientras que RMI admite paradigmas de programación orientada a objetos y está basado en Java.

- Los parámetros pasados a los procedimientos remotos en RPC son las estructuras de datos ordinarias. Por el contrario, RMI transita objetos como un parámetro para el método remoto.

- RPC se puede considerar como la versión anterior de RMI, y se usa en los lenguajes de programación que admiten la programación de procedimientos, y solo puede usar parámetros por valor. Por el contrario, RMI está diseñada en base a un enfoque de programación moderno, que podría usar pasar por valor o referencia.

- El protocolo RPC genera más overheads que RMI.

Conclusión:

La diferencia común entre RPC y RMI es que RPC solo admite programación procedural, mientras que RMI admite programación orientada a objetos.
Otra diferencia importante entre los dos es que los parámetros pasados a la llamada a procedimientos remotos consisten en estructuras de datos ordinarias. Por otro lado, los parámetros pasados al método remoto consisten en objetos.

3)
en **RMI** puede ser que los métodos de un objeto remoto admitan como parámetros clases y a su vez devuelvan clases. Por lo que un método remoto podría devolver un objeto o admitir un parámetro de una clase desconocida para el cliente. En ese caso se podría generar una ClassNotFoundException
La solución sería que tanto el servidor como el cliente tengan sus propias copias de las clases que implementen estas interfaces. Esto no sería una solución deseada, ya que se deben saber todas las clases que utiliza el servidor y copiarlas del lado del cliente, quitando dinamismo y escalabilidad a RMI.
Una solución provista por RMI es RMISecurityManager, el cual habilita la carga dinámica de clases.

5)
RMI es multihilos, lo que permite que sus servidores exploten los hilos de Java para un mejor procesamiento concurrente de las solicitudes de los clientes.
Pero un inconveniente es que no los maneja de forma segura y puede generar colisiones de recursos del servidor.
Para evitar esto se puede utilizar mecanismos conocidos para sincronización, aunque RMI no provee ninguno por su cuenta.

6
a)
Tiempo de respuesta ejecutado 20 veces
0,624823
0,593689
0,598252
0,610289
0,589310
0,592321
0,610181
0,591367
0,609691

0,591576
0,561543
0,639406
0,586642
0,595342
0,688994
0,685917
0,579678
0,579702
0,592792
0,597834

Desviación estándar (s): 0.031636727347531
Media aritmética ($\bar{x}$): 0.60624952380952

b)
El paquete java.rmi no cuenta con timeout por defecto, ni tampoco configurable, pero existe el paquete sun.rmi el cual cuenta con una propiedad que permite configurar el timeout.
Se podría generar a mano con timers un timeout pero por defecto RMI no provee manejo de timeout a no ser que utilices sun.rmi