注：

1. 请按个人的最佳习惯对下面进行答题

### 试题A1、地区表管理（SQLSERVER）

要点：

1. 创建一个地区表，带层级关系
2. 编写一个存储过程，根据一个节点标识获取该节点下的所有子节点（及子子级）

### 试题A2、编写一个搜索算法（C#）

要点：

1. 数据来源：List<int> listData;
2. 编写一个方法，对listData进行排序（非对象自带的Sort方法）
3. 编写一个方法，输入一个参数，获取该参数在列表中的位置（指定方法：二分法）

### 试题B1、编写一个登录验证页面（VS2012页面提交，下同）

要点：

1. 页面输入用户名、口令，对输入内容必要性检查，并将错误信息显示在右侧
2. 输入密码小于3位时，显示为“弱”，小于6位时，显示“中等”，超过6位时显示“强”
3. 点提交按钮，用户名小于3位时返回“用户名不存在”（代替数据库验证）

### 试题B2、编写一个页面：三个tab，鼠标点击切换下面内容页面（效果等同于tab控件）

要点：

1. 该试题采用css+div设计
2. Tab页水平分布
3. Tab页对应的内容通过点击tab标签来显示或隐藏

### 试题C、从面试者的项目经验中挑一个成功的流程案例（带角色扮演）

## 试题A1标准答案

CREATE TABLE HR\_AREA(ID INT , PID INT , NAME VARCHAR(50))

//考点1：名称（题目明确说明为与地区相关的信息）

//考点2：通过ID、PID建立层级关系

//考点3：考察递归算法的存储过程

存储过程参考答案1

CREATE PROCEDURE HR\_AREA\_GETALLSUB(

 @ID INT)

AS

BEGIN

 CREATE TABLE #TMP(ID INT) //或DECLARE @TMP TABLE(ID INT)

 INSERT INTO #TMP (ID) VALUES(@ID);

 WHILE @@ROWCOUNT > 0

 BEGIN

 INSERT INTO #TMP(ID)

 SELECT ID

 FROM HR\_AREA

 WHERE DELETED = 0

AND PID IN(SELECT ID FROM #TMP)

AND ID NOT IN(SELECT ID FROM #TMP)

 END

 SELECT ID

 FROM #TMP

END

存储过程参考答案2（CTE模式）

CREATE PROCEDURE HR\_AREA\_GETALLSUB(

 @ID INT)

AS

BEGIN

WITH cte(id) AS (SELECT ID

FROM HR\_AREA

WHERE (ID = @ID)

UNION ALL

SELECT HR\_AREA.ID

FROM cte INNER JOIN

HR\_AREA ON cte.id = HR\_AREA.PID)

SELECT id

FROM cte RETURN

END

## 试题A2标准答案

//考点2：基础算法的编写能力

//考点2：递归算法的实现

//考点3：命名习惯

### 对数据进行排序

### 二分法搜索数据

 /// <summary>

 /// 面试用题（标准答案）

 /// 1、冒泡排序

 /// 2、二分法搜索算法

 /// </summary>

 public class Test

 {

 /// <summary>

 /// 存储数值列表

 /// </summary>

 private List<int> list = new List<int>();

 public Test()

 {

 }

 /// <summary>

 /// 初始化实例对象

 /// </summary>

 public void Init()

 {

 list.Add(1);

 list.Add(3);

 list.Add(2);

 }

 /// <summary>

 /// 冒泡排序方法（升序编码）

 /// </summary>

 public void Sort()

 {

 for (int i = 0; i < list.Count - 1; i++)

 {

 for (int j = i + 1; j < list.Count; j++)

 {

 if (list[i] > list[j])

 {

 //经典数据交换算法

 list[i] += list[j];

 list[j] = list[i] - list[j];

 list[i] -= list[j];

 }

 }

 }

 }

 /// <summary>

 /// 从列表中搜索指定一个数值位置

 /// </summary>

 /// <param name="nValue">搜索对象</param>

 /// <returns>返回结果值在列表中的位置，-1表示没找到</returns>

 public int FindIndex(int nValue)

 {

 return FindIndex(nValue, 0, list.Count - 1);

 }

 /// <summary>

 /// 二分法递归搜索等值数据位置

 /// </summary>

 /// <param name="nValue">搜索对象</param>

 /// <param name="nStartPos">起始位置</param>

 /// <param name="nStopPos">结束位置</param>

 /// <returns></returns>

 public int FindIndex(int nValue, int nStartPos, int nStopPos)

 {

 if (nStopPos < nStartPos)

 return -1;

 int nMidPos = (nStartPos + nStopPos) / 2;

 if (list[nMidPos] == nValue)

 return nMidPos;

 else if (nValue < list[nMidPos])

 return FindIndex(nValue, nStartPos, nMidPos - 1);

 else

 return FindIndex(nValue, nMidPos + 1, nStopPos);

 }

 }