BST

```
class Node {
   int key;
   int value;
   Node leftChild;
   Node rightChild;

   public Node(int key, int value) {
      this.key = key;
      this.value = value;
   }

   public void displayNode() {

   }
}

class Tree {
   Node root;

   public Node find(int key) {
      Node currentNode = root;
      while (currentNode != null && currentNode.key != key) {
         if (key < currentNode.key) {
            currentNode = currentNode.leftChild;
         } else {
            currentNode = currentNode.rightChild;
         }
      }
      return currentNode;
   }

   public void insert(int key, int value) {
      if (root == null) {
         root = new Node(key, value);
         return;
      }
      Node currentNode = root;
      Node parentNode = root;
      boolean isLeftChild = true;
      while (currentNode != null) {
         parentNode = currentNode;
         if (key < currentNode.key) {
            currentNode = currentNode.leftChild;
            isLeftChild = true;
        } else {
            currentNode = currentNode.rightChild;
            isLeftChild = false;
        }
    }
    Node newNode = new Node(key, value);
    if (isLeftChild) {
        parentNode.leftChild = newNode;
    } else {
        parentNode.rightChild = newNode;
    }
}

public boolean delete(int key) {
    Node currentNode = root;
    Node parentNode = root;
    boolean isLeftChild = true;
    while (currentNode != null && currentNode.key != key) {
        parentNode = currentNode;
        if (key < currentNode.key) {
            currentNode = currentNode.leftChild;
            isLeftChild = true;
        } else {
            currentNode = currentNode.rightChild;
            isLeftChild = false;
        }
    }
    if (currentNode == null) {
        return false;
    }
    if (currentNode.leftChild == null && currentNode.rightChild == null) {
        //要删除的节点为叶子节点
        if (currentNode == root)
            root = null;
        else if (isLeftChild)
            parentNode.leftChild = null;
        else
            parentNode.rightChild = null;
    } else if (currentNode.rightChild == null) {//要删除的节点只有左孩子
        if (currentNode == root)
            root = currentNode.leftChild;
        else if (isLeftChild)
            parentNode.leftChild = currentNode.leftChild;
```

```
      else
        parentNode.rightChild = currentNode.leftChild;
    } else if (currentNode.leftChild == null) {//要删除的节点只有右孩子
      if (currentNode == root)
        root = currentNode.rightChild;
      else if (isLeftChild)
        parentNode.leftChild = currentNode.rightChild;
      else
        parentNode.rightChild = currentNode.rightChild;
    } else { //要删除的节点既有左孩子又有右孩子
      //思路：用待删除节点右子树中的 key 值最小节点的值来替代要删除的节点的值,然后删除右子树中 key 值最小的节点
      //右子树 key 最小的节点一定不含左子树,所以删除这个 key 最小的节点一定是属于叶子节点或者只有右子树的节点
      Node directPostNode = getDirectPostNode(currentNode);
      currentNode.key = directPostNode.key;
      currentNode.value = directPostNode.value;
    }
    return true;
  }

  private Node getDirectPostNode(Node delNode) {//方法作用为得到待删除节点的直接后继节点

    Node parentNode = delNode;//用来保存待删除节点的直接后继节点的父亲节点
    Node direcrPostNode = delNode;//用来保存待删除节点的直接后继节点
    Node currentNode = delNode.rightChild;
    while (currentNode != null) {
      parentNode = direcrPostNode;
      direcrPostNode = currentNode;
      currentNode = currentNode.leftChild;
    }
    if (direcrPostNode != delNode.rightChild) {//从树中删除此直接后继节点
      parentNode.leftChild = direcrPostNode.rightChild;
      direcrPostNode.rightChild = null;
    }
    return direcrPostNode;//返回此直接后继节点

  }

  public void preOrder(Node rootNode) {
    if (rootNode != null) {
      System.out.println(rootNode.key + " " + rootNode.value);
      preOrder(rootNode.leftChild);
      preOrder(rootNode.rightChild);
    }
  }

  public void inOrder(Node rootNode) {
    if (rootNode != null) {
      inOrder(rootNode.leftChild);
      System.out.println("key: " + rootNode.key + " " + "value: " + rootNode.value);
      inOrder(rootNode.rightChild);
    }
  }

  public void postOrder(Node rootNode) {
    if (rootNode != null) {
      postOrder(rootNode.leftChild);
      postOrder(rootNode.rightChild);
      System.out.println(rootNode.key + " " + rootNode.value);
    }
  }

      private void destroy(Node tree) {
         if (tree==null)
           return ;

         if (tree.left != null)
           destroy(tree.leftChild);
         if (tree.right != null)
           destroy(tree.rightChild);

        tree=null;
      }

         public void destory() {
              destory(root);
         }
}
public class BinarySearchTreeApp {
   public static void main(String[] args) {
           Tree tree = new Tree();
       tree.insert(6, 6);//插入操作,构造图一所示的二叉树
       tree.insert(3, 3);
         tree.insert(14, 14);
        tree.insert(16, 16);
        tree.insert(10, 10);
        tree.insert(9, 9);
         tree.insert(13, 13);
        tree.insert(11, 11);
         tree.insert(12, 12);
        System.out.println("删除前遍历结果");
         tree.inOrder(tree.root);//中序遍历操作
        System.out.println("删除节点 10 之后遍历结果");
         tree.delete(10);//删除操作
          tree.inOrder(tree.root);
         }
}
```

```
/**
 * Java 语言: 二叉查找树
 *
 * @author skywang
 * @date 2013/11/07
 */

public class BSTree<T extends Comparable<T>> {

    private BSTNode<T> mRoot;    // 根结点

    public class BSTNode<T extends Comparable<T>> {
        T key;                // 关键字(键值)
        BSTNode<T> left;    // 左孩子
        BSTNode<T> right;    // 右孩子
        BSTNode<T> parent;    // 父结点

        public BSTNode(T key, BSTNode<T> parent, BSTNode<T> left, BSTNode<T> right) {
            this.key = key;
            this.parent = parent;
            this.left = left;
            this.right = right;
        }

        public T getKey() {
            return key;
        }
        public String toString() {
            return "key:"+key;
        }
    }

    public BSTree() {
        mRoot=null;
    }

    /*
     * 前序遍历"二叉树"
     */
    private void preOrder(BSTNode<T> tree) {
        if(tree != null) {
            System.out.print(tree.key+" ");
            preOrder(tree.left);
            preOrder(tree.right);
        }
    }

    public void preOrder() {
        preOrder(mRoot);
    }

    /*
     * 中序遍历"二叉树"
     */
    private void inOrder(BSTNode<T> tree) {
        if(tree != null) {
            inOrder(tree.left);
            System.out.print(tree.key+" ");
            inOrder(tree.right);
        }
    }

    public void inOrder() {
        inOrder(mRoot);
    }


    /*
     * 后序遍历"二叉树"
     */
```

```
private void postOrder(BSTNode<T> tree) {
   if(tree != null)
   {
      postOrder(tree.left);
      postOrder(tree.right);
      System.out.print(tree.key+" ");
   }
}

public void postOrder() {
   postOrder(mRoot);
}


/*
 * (递归实现)查找"二叉树 x"中键值为 key 的节点
 */
private BSTNode<T> search(BSTNode<T> x, T key) {
   if (x==null)
      return x;

   int cmp = key.compareTo(x.key);
   if (cmp < 0)
      return search(x.left, key);
   else if (cmp > 0)
      return search(x.right, key);
   else
      return x;
}

public BSTNode<T> search(T key) {
   return search(mRoot, key);
}

/*
 * (非递归实现)查找"二叉树 x"中键值为 key 的节点
 */
private BSTNode<T> iterativeSearch(BSTNode<T> x, T key) {
   while (x!=null) {
      int cmp = key.compareTo(x.key);

      if (cmp < 0)
         x = x.left;
      else if (cmp > 0)
```

```
            x = x.right;
        else
            return x;
    }

    return x;
}

public BSTNode<T> iterativeSearch(T key) {
    return iterativeSearch(mRoot, key);
}

/*
 * 查找最小结点：返回 tree 为根结点的二叉树的最小结点。
 */
private BSTNode<T> minimum(BSTNode<T> tree) {
    if (tree == null)
        return null;

    while(tree.left != null)
        tree = tree.left;
    return tree;
}

public T minimum() {
    BSTNode<T> p = minimum(mRoot);
    if (p != null)
        return p.key;

    return null;
}

/*
 * 查找最大结点：返回 tree 为根结点的二叉树的最大结点。
 */
private BSTNode<T> maximum(BSTNode<T> tree) {
    if (tree == null)
        return null;

    while(tree.right != null)
        tree = tree.right;
    return tree;
}
```

```
    public T maximum() {
        BSTNode<T> p = maximum(mRoot);
        if (p != null)
            return p.key;

        return null;
    }

    /*
     * 找结点(x)的后继结点。即，查找"二叉树中数据值大于该结点"的"最小结点"。
     */
    public BSTNode<T> successor(BSTNode<T> x) {
        // 如果 x 存在右孩子，则"x 的后继结点"为 "以其右孩子为根的子树的最小结点"。
        if (x.right != null)
            return minimum(x.right);

        // 如果 x 没有右孩子。则 x 有以下两种可能：
        // (01) x 是"一个左孩子"，则"x 的后继结点"为 "它的父结点"。
        // (02) x 是"一个右孩子"，则查找"x 的最低的父结点，并且该父结点要具有左孩子"，
找到的这个"最低的父结点"就是"x 的后继结点"。
        BSTNode<T> y = x.parent;
        while ((y!=null) && (x==y.right)) {
            x = y;
            y = y.parent;
        }

        return y;
    }

    /*
     * 找结点(x)的前驱结点。即，查找"二叉树中数据值小于该结点"的"最大结点"。
     */
    public BSTNode<T> predecessor(BSTNode<T> x) {
        // 如果 x 存在左孩子，则"x 的前驱结点"为 "以其左孩子为根的子树的最大结点"。
        if (x.left != null)
            return maximum(x.left);

        // 如果 x 没有左孩子。则 x 有以下两种可能：
        // (01) x 是"一个右孩子"，则"x 的前驱结点"为 "它的父结点"。
        // (01) x 是"一个左孩子"，则查找"x 的最低的父结点，并且该父结点要具有右孩子"，
找到的这个"最低的父结点"就是"x 的前驱结点"。
        BSTNode<T> y = x.parent;
    while ((y!=null) && (x==y.left)) {
        x = y;
        y = y.parent;
    }

    return y;
}

/*
 * 将结点插入到二叉树中
 *
 * 参数说明：
 *     tree 二叉树的
 *     z 插入的结点
 */
private void insert(BSTree<T> bst, BSTNode<T> z) {
    int cmp;
    BSTNode<T> y = null;
    BSTNode<T> x = bst.mRoot;

    // 查找 z 的插入位置
    while (x != null) {
        y = x;
        cmp = z.key.compareTo(x.key);
        if (cmp < 0)
            x = x.left;
        else
            x = x.right;
    }

    z.parent = y;
    if (y==null)
        bst.mRoot = z;
    else {
        cmp = z.key.compareTo(y.key);
        if (cmp < 0)
            y.left = z;
        else
            y.right = z;
    }
}

/*
```

```
 * 新建结点(key)，并将其插入到二叉树中
 *
 * 参数说明：
 *     tree 二叉树的根结点
 *     key 插入结点的键值
 */
public void insert(T key) {
    BSTNode<T> z=new BSTNode<T>(key,null,null,null);

    // 如果新建结点失败，则返回。
    if (z != null)
        insert(this, z);
}

/*
 * 删除结点(z)，并返回被删除的结点
 *
 * 参数说明：
 *     bst 二叉树
 *     z 删除的结点
 */
private BSTNode<T> remove(BSTree<T> bst, BSTNode<T> z) {
    BSTNode<T> x=null;
    BSTNode<T> y=null;

    if ((z.left == null) || (z.right == null) )
        y = z;
    else
        y = successor(z);

    if (y.left != null)
        x = y.left;
    else
        x = y.right;

    if (x != null)
        x.parent = y.parent;

    if (y.parent == null)
        bst.mRoot = x;
    else if (y == y.parent.left)
        y.parent.left = x;
    else
```

```
      y.parent.right = x;

   if (y != z)
      z.key = y.key;

   return y;
}

/*
 * 删除结点(z)，并返回被删除的结点
 *
 * 参数说明：
 *     tree 二叉树的根结点
 *     z 删除的结点
 */
public void remove(T key) {
   BSTNode<T> z, node;

   if ((z = search(mRoot, key)) != null)
      if ( (node = remove(this, z)) != null)
         node = null;
}

/*
 * 销毁二叉树
 */
private void destroy(BSTNode<T> tree) {
   if (tree==null)
      return ;

   if (tree.left != null)
      destroy(tree.left);
   if (tree.right != null)
      destroy(tree.right);

   tree=null;
}

public void clear() {
   destroy(mRoot);
   mRoot = null;
}
```

```
    /*
     * 打印"二叉查找树"
     *
     * key        -- 节点的键值
     * direction  --  0，表示该节点是根节点;
     *               -1，表示该节点是它的父结点的左孩子;
     *                1，表示该节点是它的父结点的右孩子。
     */
    private void print(BSTNode<T> tree, T key, int direction) {

        if(tree != null) {

            if(direction==0)    // tree是根节点
                System.out.printf("%2d is root\n", tree.key);
            else                // tree是分支节点
                System.out.printf("%2d is %2d's %6s child\n", tree.key, key, direction==1?"right" :
"left");

            print(tree.left, tree.key, -1);
            print(tree.right,tree.key,  1);
        }
    }

    public void print() {
        if (mRoot != null)
            print(mRoot, mRoot.key, 0);
    }
}
```

Balance tree

```
public static class Node{
        public int value;
        public Node left;
        public Node right;
        public Node(int data){
                this.value = data;
        }
}
public static boolean isBalance(Node head){
        return getHeight(head, 0) ! = -1;
}
```

```
public static int getHeight(Node head, int level){
    if(head == null){
        return level;
    }
    int lh = getHeight(head.left, level + 1);
    int rh = getHeight(head.right, level + 1);
    if(lh == -1 || rh == -1 || Math.abs(lh - rh) > 1){
        return -1;
    }
}
```

Complete binary tree

```
public static boolean isComplete(Node head){
    if(head == null){
        return true;
    }
    Queue<Node> queue = new LinkedList<Node>();
    boolean leaf = false;
    Node cur = null;
    Node l = null;
    Node r = null;
    queue.offer(head);
    while(!queue.isEmpty){
        cur = queue.poll();
        l = cur.left;
        r = cur.right;
        if((leaf && (l != null || r != null)) || (l == null && r != null)){
            return false;
        }
        if(l != null){
            queue.offer(l);
        }
        if(r != null){
            queue.offer(r);
        }else{
            leaf = true;
        }
    }
}
```

```
public void insertSort(int [] a){
   int len=a.length;//单独把数组长度拿出来，提高效率
   int insertNum;//要插入的数
   for(int i=1;i<len;i++){//因为第一次不用，所以从 1 开始
     insertNum=a[i];
     int j=i-1;//序列元素个数
     while(j>=0&&a[j]>insertNum){//从后往前循环，将大于 insertNum 的数向后移动
       a[j+1]=a[j];//元素向后移动
       j--;
     }
     a[j+1]=insertNum;//找到位置，插入当前元素
   }
 }
```

```
public void selectSort(int[]a){
     int len=a.length;
     for(int i=0;i<len;i++){//循环次数
       int value=a[i];
       int position=i;
       for(int j=i+1;j<len;j++){//找到最小的值和位置
         if(a[j]<value){
           value=a[j];
           position=j;
         }
       }
       a[position]=a[i];//进行交换
       a[i]=value;
     }
   }
```

```
public  void heapSort(int[] a){
       int len=a.length;
       //循环建堆
       for(int i=0;i<len-1;i++){
         //建堆
         buildMaxHeap(a,len-1-i);
         //交换堆顶和最后一个元素
         swap(a,0,len-1-i);
       }
     }
     //交换方法
```

```
private  void swap(int[] data, int i, int j) {
   int tmp=data[i];
   data[i]=data[j];
   data[j]=tmp;
}
//对 data 数组从 0 到 lastIndex 建大顶堆
private void buildMaxHeap(int[] data, int lastIndex) {
   //从 lastIndex 处节点（最后一个节点）的父节点开始
   for(int i=(lastIndex-1)/2;i>=0;i--){
      //k 保存正在判断的节点
      int k=i;
      //如果当前 k 节点的子节点存在
      while(k*2+1<=lastIndex){
         //k 节点的左子节点的索引
         int biggerIndex=2*k+1;
         //如果 biggerIndex 小于 lastIndex，即 biggerIndex+1 代表的 k 节点的右子节点存在
         if(biggerIndex<lastIndex){
            //若果右子节点的值较大
            if(data[biggerIndex]<data[biggerIndex+1]){
               //biggerIndex 总是记录较大子节点的索引
               biggerIndex++;
            }
         }
         //如果 k 节点的值小于其较大的子节点的值
         if(data[k]<data[biggerIndex]){
            //交换他们
            swap(data,k,biggerIndex);
            //将 biggerIndex 赋予 k，开始 while 循环的下一次循环，重新保证 k 节点的值大于其左右子节点的值
            k=biggerIndex;
         }else{
            break;
         }
      }
   }
}
```

```
public void bubbleSort(int []a){
        int len=a.length;
```

```
      for(int i=0;i<len;i++){
        for(int j=0;j<len-i-1;j++){//注意第二重循环的条件
          if(a[j]>a[j+1]){
            int temp=a[j];
            a[j]=a[j+1];
            a[j+1]=temp;
          }
        }
      }
    }
```

```
public void quickSort(int[]a,int start,int end){
      if(start<end){
        int baseNum=a[start];//选基准值
        int midNum;//记录中间值
        int i=start;
        int j=end;
        do{
          while((a[i]<baseNum)&&i<end){
            i++;
          }
          while((a[j]>baseNum)&&j>start){
            j--;
          }
          if(i<=j){
            midNum=a[i];
            a[i]=a[j];
            a[j]=midNum;
            i++;
            j--;
          }
        }while(i<=j);
         if(start<j){
           quickSort(a,start,j);
         }
         if(end>i){
           quickSort(a,i,end);
         }
      }
    }
```

```
public  void mergeSort(int[] a, int left, int right) {
      int t = 1;// 每组元素个数
      int size = right - left + 1;
      while (t < size) {
        int s = t;// 本次循环每组元素个数
        t = 2 * s;
        int i = left;
        while (i + (t - 1) < size) {
          merge(a, i, i + (s - 1), i + (t - 1));
          i += t;
        }
        if (i + (s - 1) < right)
          merge(a, i, i + (s - 1), right);
      }
    }

    private static void merge(int[] data, int p, int q, int r) {
      int[] B = new int[data.length];
      int s = p;
      int t = q + 1;
      int k = p;
      while (s <= q && t <= r) {
        if (data[s] <= data[t]) {
          B[k] = data[s];
          s++;
        } else {
          B[k] = data[t];
          t++;
        }
        k++;
      }
      if (s == q + 1)
        B[k++] = data[t++];
      else
        B[k++] = data[s++];
      for (int i = p; i <= r; i++)
        data[i] = B[i];
    }
```

```
public void baseSort(int[] a) {
        //首先确定排序的趟数;
        int max = a[0];
        for (int i = 1; i < a.length; i++) {
            if (a[i] > max) {
                max = a[i];
            }
        }
        int time = 0;
        //判断位数;
        while (max > 0) {
            max /= 10;
            time++;
        }
        //建立 10 个队列;
        List<ArrayList<Integer>> queue = new ArrayList<ArrayList<Integer>>();
        for (int i = 0; i < 10; i++) {
            ArrayList<Integer> queue1 = new ArrayList<Integer>();
            queue.add(queue1);
        }
        //进行 time 次分配和收集;
        for (int i = 0; i < time; i++) {
            //分配数组元素;
            for (int j = 0; j < a.length; j++) {
                //得到数字的第 time+1 位数;
                int x = a[j] % (int) Math.pow(10, i + 1) / (int) Math.pow(10, i);
                ArrayList<Integer> queue2 = queue.get(x);
                queue2.add(a[j]);
                queue.set(x, queue2);
            }
            int count = 0;//元素计数器;
            //收集队列元素;
            for (int k = 0; k < 10; k++) {
                while (queue.get(k).size() > 0) {
                    ArrayList<Integer> queue3 = queue.get(k);
                    a[count] = queue3.get(0);
                    queue3.remove(0);
                    count++;
                }
            }
        }
    }
```

```java
package cstring;


/**
 * File Name: BigNumber.java
 * Infinite capacity Unsigned Number
 *
 * @author Jagadeesh Vasudevamurthy
 * @year 2018
 */
/*
 * To compile you require:  IntUtil.java RandomInt.java CharArray.java Cstring.java 
BigNumber.java
 */

class BigNumber {
   public Cstring d; //data
   static IntUtil u = new IntUtil();
   //YOU CANNOT add any data members
   //YOU CAN add any public or private function so that all the tests will pass

   public void pLn(String t) {
      d.pLn(t) ;
   }

   //WRITE ALL THE ROUTINES required to pass all the tests in BigNumberTester.java

   public BigNumber() {
      d = new Cstring();
   }

   public BigNumber(char ch) {
      d = new Cstring(ch);
   }

   public BigNumber(String str) {
      d = new Cstring(str);
   }

   public BigNumber(char[] arrChar) {
      d = new Cstring(arrChar);
   }

   public BigNumber(int num) {
      char[] chArray = String.valueOf(num).toCharArray();
      d = new Cstring(chArray);
   }

   public BigNumber(Cstring cstr) {
      d = cstr;
   }

   public BigNumber clone() {
      Cstring cstr = this.d.clone();
      BigNumber bigNum = new BigNumber(cstr);
      return bigNum;
   }

   public boolean isEqual(BigNumber bigNum) {
      return this.d.isEqual(bigNum.d);
   }
```

```java
public boolean isEqual(String str) {
   BigNumber bigNum = new BigNumber(str);
   return this.isEqual(bigNum);
}

public boolean isEqual(int num) {
   BigNumber bigNum = new BigNumber(num);
   return this.isEqual(bigNum);
}

public BigNumber add(BigNumber bigNum) {
   BigNumber r = new BigNumber();
   BigNumber n1 = this.clone();
   BigNumber n2 = bigNum.clone();
   int size1 = n1.size();
   int size2 = n2.size();
   int carry, digit;

   n1.d.reverse();
   n2.d.reverse();

   if (size1 > size2) {
      for (int i = 0; i < size1 - size2; ++i)
         n2.d.append("0");
   } else if (size1 < size2) {
      for (int i = 0; i < size2 - size1; ++i)
         n1.d.append("0");
   }

   carry = 0;
   for (int i = 0; i < n1.size(); ++i) {
      digit = n1.d.get(i) + n2.d.get(i) + carry;
      if (digit > 9) {
         digit -= 10;
         carry = 1;
      } else {
         carry = 0;
      }
      r.d.append("" + digit);
   }
   if (carry > 0)
      r.d.append("" + carry);

   r.d.reverse();

   return r;
}

public BigNumber sub(BigNumber bigNum) {
   BigNumber r = new BigNumber();
   BigNumber temp = new BigNumber();
   BigNumber n1 = this.clone();
   BigNumber n2 = bigNum.clone();
   int carry, digit;

   String s1 = new String();
   String s2 = new String();
   for (int i = 0; i < n1.size(); i++) {
      s1 += n1.d.get(i);
   }
```

```
for (int i = 0; i < n2.size(); i++) {
   s2 += n2.d.get(i);
}

boolean isNegative;

if (n1.isEqual(n2)) {
   isNegative = false;
} else {
   isNegative = compare(s1, s2);
}

if (!isNegative) {
   BigNumber t = n1;
   n1 = n2;
   n2 = t;
}

n1.d.reverse();
n2.d.reverse();

int j = n1.size() - n2.size();
if (n1.size() > n2.size()) {
   int i = 0;
   for (i = 0; i < j; ++i) {
      n2.d.append("0");
   }
} else if (n1.size() < n2.size()) {
   for (int i = 0; i < n2.size() - n1.size(); ++i)
      n1.d.append("0");
}

// sub by digit
if (n2.isEqual(0)) {
   temp = n1;
} else {
   carry = 0;
   for (int i = 0; i < n1.size(); ++i) {
      digit = n1.d.get(i) - n2.d.get(i) + carry;
      if (digit < 0) {
         digit += 10;
         carry = -1;
      } else {
         carry = 0;
      }
      temp.d.append("" + digit);
   }
}
temp.d.reverse();

int n3 = temp.size();
boolean flag = true;
//check if the result is 0
for (int i = 0; i < n3; i++) {
   if (temp.d.get(i) != 0) {
      flag = false;
      break;
   }
}
```

```java
      int index = 0;
      char[] arrChar = new char[temp.size()];
      if (flag) {
         r = new BigNumber(0);
      } else {
         for (int i = delZero(temp); i < n3; i++) {
            int t = temp.d.get(i);
            arrChar[index++] = String.valueOf(t).charAt(0);
         }
         r = new BigNumber(arrChar);
      }
      r.d.reverse();
      if (!isNegative) {
         if (!n1.isEqual(n2)) {
            r.d.append("-");
         }
      }
      r.d.reverse();
      return r;
   }

   public BigNumber mult(BigNumber bigNum) {
      BigNumber newNum = new BigNumber(0);

      if (this.isEqual(0) || bigNum.isEqual(0))
         return newNum;

      BigNumber n1 = this.clone();
      BigNumber n2 = bigNum.clone();
      int carry = 0, digit;

      n2.d.reverse();

      for (int i = 0; i < n1.size(); ++i) {
         BigNumber tempNum = new BigNumber();
         carry = carry / 10;
         for (int j = 0; j < n2.size(); ++j) {
            digit = n1.d.get(i) * n2.d.get(j) + carry;
            carry = digit / 10;
            digit = digit % 10;
            tempNum.d.append("" + digit);
         }
         if (carry > 0)
            tempNum.d.append("" + carry);
         tempNum.d.reverse();

         if (n1.size() > 1 && !newNum.isEqual(0))
            newNum.d.append("0");
         newNum = newNum.add(tempNum);
      }
      return newNum;
   }

   public static BigNumber factorial(int n) {

      int res[] = new int[5000];

      res[0] = 1;
      int res_size = 1;

      for (int x = 2; x <= n; x++)
```

```java
         res_size = multiply(x, res, res_size);

      char[] arrChar = new char[res_size];
      for (int i = res_size - 1; i >= 0; i--) {
         arrChar[i] = String.valueOf(res[i]).charAt(0);
      }
      BigNumber f = new BigNumber(arrChar);
      f.d.reverse();
      return f;
   }

   static int multiply(int x, int res[], int res_size) {
      int carry = 0;
      for (int i = 0; i < res_size; i++) {
         int prod = res[i] * x + carry;
         res[i] = prod % 10;
         carry = prod / 10;
      }

      while (carry != 0) {
         res[res_size] = carry % 10;
         carry = carry / 10;
         res_size++;
      }
      return res_size;
   }
   public int size() {
      return this.d.size();
   }

   public boolean compare(String s1, String s2) {

      if (s1.length() < s2.length()) {
         return false;
      } else if (s1.length() > s2.length()) {
         return true;
      } else {
         if (s1.charAt(0) > s2.charAt(0)) {
            return true;
         } else if (s1.charAt(0) < s2.charAt(0)) {
            return false;
         } else {
            return compare(s1.substring(1), s2.substring(1));
         }
      }
   }
   public int delZero(BigNumber n) {
      int index = 0;
      for (int i = 0; i < n.size(); i++) {
         if (n.d.get(i) != 0) {
            index = i;
            break;
         }
      }
      return index;
   }
   public static void main(String[] args) {
      System.out.println("BigNumber.java");
      System.out.println("Done");
   }
}
```

```java
package cstring;

import java.util.regex.Pattern;

/**
 * File Name: Cstring.java
 * Implements C String
 *
 * @author Jagadeesh Vasudevamurthy
 * @year 2016
 */
/*
 * To compile you require: IntUtil.java RandomInt.java CharArray.java Cstring.java
 * WRITE CODE IN THIS FILE
 */

class Cstring {
   //YOU CANNOT ADD ANYTHING HERE
   private CharArray d; //Infinite array of char
   static IntUtil u = new IntUtil();

   //WRITE ALL THE ROUTINES BELOW, so that all the tests pass

   public Cstring() {
      d = new CharArray();
   }

   public Cstring(char ch) {
      d = new CharArray();
      d.set(0, ch);
      d.set(1, '\0');
   }

   public Cstring(char[] arrChar) {
      d = new CharArray();
      int i = 0;
      for (char ch: arrChar) {
         d.set(i++, ch);
      }
      d.set(i, '\0');
   }

   public Cstring(String s) {
      d = new CharArray();
      int i = 0;
      for (char ch: s.toCharArray()) {
         d.set(i++, ch);
      }
      d.set(i, '\0');
   }

   public void pLn(String s) {
      for (char ch: s.toCharArray()) {
         System.out.print(ch);
      }
      for (int i = 0; i < this.size(); ++i) {
         System.out.print(this.d.get(i));
      }
      System.out.println();
   }
```

```
public Cstring clone() {
   char[] arrChar = new char[this.size()];
   for (int i = 0; i < this.size(); ++i) {
      arrChar[i] = this.d.get(i);
   }
   Cstring cs = new Cstring(arrChar);
   return cs;
}

public Cstring add(Cstring cs) {
   char[] arrChar = new char[this.size() + cs.size() + 1];
   for (int i = 0; i < this.size(); ++i) {
      arrChar[i] = this.d.get(i);
   }
   for (int i = 0; i < cs.size(); ++i) {
      arrChar[i + this.size()] = cs.d.get(i);
   }
   Cstring newCs = new Cstring(arrChar);
   return newCs;
}

public Cstring add(String s) {
   Cstring cs = new Cstring(s);
   return this.add(cs);
}

public Cstring append(Cstring cs) {
   int length = this.size();
   for (int i = 0; i < cs.size() ; ++i) {
      this.d.set(length + i, cs.d.get(i));
   }
   return this;
}

public Cstring append(String s) {
   Cstring cs = new Cstring(s);
   return this.append(cs);
}

public void reverse() {
   int i = 0;
   int j = this.size() - 1;
   while (i < j) {
      this.d.swap(i, j);
      ++i;
      --j;
   }
   d.set(this.size(), '\0');
}

public boolean isEqual(Cstring cs) {
   if (this.size() != cs.size()) {
      return false;
   }
   for (int i = 0; i < this.size() ; ++i) {
      if (this.d.get(i) != cs.d.get(i)) {
         return false;
      }
   }
   return true;
}
```

```
   public int size() {
      int length = 0;
      while (this.d.get(length++) != '\0');
      return length - 1;
   }

   public int get(int index) {
      return this.d.get(index) - '0';
   }

   public void set(int index, int value) {
      this.d.set(index, String.valueOf(value).charAt(0));
   }

   private static void testBasic() {
      Cstring a = new Cstring('b') ;
      a.pLn("a = ") ;
      Cstring b = new Cstring('7') ;
      b.pLn("b = ") ;
      Cstring c = new
Cstring("123456789012345678901234567890123456789012345678901234567890") ;
      c.pLn("c = ") ;
      Cstring d = c.clone() ;
      d.pLn("d = ") ;
      Cstring e = new Cstring("A quick brown fox junped over a lazy dog") ;
      e.pLn("e = ") ;
      Cstring f = new Cstring("Gateman sees name garageman sees nametag") ;
      f.pLn("f =  ") ;
      f.reverse() ;
      f.pLn("f' = ") ;
   }

   private static void testAdd() {
      Cstring a = new Cstring("UCSC") ;
      Cstring b = new Cstring("Extension") ;
      Cstring c = a.add(b) ;
      a.pLn("a = ") ;
      b.pLn("b = ") ;
      c.pLn("c = ") ;
      Cstring d = c.add("USA") ;
      d.pLn("d = ") ;
      a.append(b) ;
      a.pLn("a+b = ") ;
      a.append("World") ;
      a.pLn("a+b+World = ") ;
   }

   private static void testEqual() {
      Cstring a = new
Cstring("123456789012345678901234567890123456789012345678901234567890") ;
      a.pLn("a = ") ;
      Cstring b = new
Cstring("123456789012345678901234567890123456789012345678901234567890") ;
      b.pLn("b = ") ;
      u.myassert(a.isEqual(b)) ;
      Cstring c = new
Cstring("12345678901234567890123456789012345678901234567890123456789") ;
      c.pLn("c = ") ;
      u.myassert(a.isEqual(c) == false) ;
   }
```

```java
   private static void testBench() {
      System.out.println("-----------Basic----------------");
      testBasic() ;
      System.out.println("-----------Addition----------------");
      testAdd() ;
      System.out.println("-----------Equal----------------");
      testEqual() ;
   }

   public static void main(String[] args) {
      System.out.println("Cstring.java");
      testBench();
      System.out.println("Done");
   }

}
```

```java
package cstring;


/**
 * File Name: CharArray.java
 * Infinite capacity char array
 *
 * @author Jagadeesh Vasudevamurthy
 * @year 2016
 */
/*
 * To compile you require: IntUtil.java RandomInt.java CharArray.java
 */

/*
 * NOTHING CAN BE CHANGED IN THIS FILE
 */

class CharArray {
   /*
    * ALL PRIVATE DATA BELOW
    */
   private int capacity;
   private char[] darray;
   static private boolean display = false;
   static IntUtil u = new IntUtil();

   /*
    * ALL PUBLIC ROUTINES BELOW
    */
   static void setDisplay(boolean x) {
      display = x;
   }

   //Constructor that takes integer
   public CharArray(int s) {
      allocate(s);
      if (display == true) {
         System.out.println("Creating darray of int of capacity " + capacity);
      }
   }

   //Constructor that takes nothing
```

```
public CharArray() {
   this(16); // This must be a first line
}

public char get(int pos) {
   if (pos < 0) {
      u.myassert(false);
      return 'a' ; //Make compiler happy
   }
   if (pos < capacity) {
      return darray[pos];
   }
   grow(pos);
   return darray[pos];
}

public void set(int pos, char val) {
   if (pos < 0) {
      u.myassert(false);
   }
   if (pos >= capacity) {
      grow(pos);
   }
   darray[pos] = val;
}

public void swap(int a, int b) {
   char x = darray[a] ;
   darray[a] = darray[b] ;
   darray[b] = x ;
}

/*
 * ALL PRIVATES ROUTINES BELOW
 */

private void allocate(int s) {
   capacity = s;
   darray = new char[s];
}

private void grow(int s) {
   char[] ta = darray;
   int ts = capacity ;
   int ns = capacity;
   do {
      ns = ns * 2;
   } while (ns <= s);

   if (display == true) {
      System.out.println("Array grew from " + ts + " to " + ns);
   }
   u.myassert(s < ns);
   allocate(ns);
   for (int i = 0; i < ts; ++i) {
      darray[i] = ta[i];
   }
   ta = null;
}

/*
```

```
    * All test routines
    */

   private static void test1() {
      CharArray b = new CharArray();
      int s = 0 ;
      for (int i = 0; i < 8; ++i) {
         b.set(i, (char)('a'+i));
         ++s ;
      }
      CharArray a = new CharArray();
      a.set(3, 'Z');
      a.set(56, 'U');
      char x = a.get(3);
      char y = a.get(56);
      char z = a.get(100);
      System.out.println("a[3]= " + x + " a[56] = " + y + " a[100] = " + z);
   }

   private static void testBench() {
      CharArray.setDisplay(true);
      System.out.println("------------test1---------------------");
      test1();
   }

   public static void main(String[] args) {
      System.out.println("CharArray.java");
      testBench();
      System.out.println("CharArray.java Done");
   }
}
```

```
  private static void bfs(HashMap<Character, LinkedList<Character>>
graph,HashMap<Character, Integer> dist,char start)
  {
    Queue<Character> q=new LinkedList<>();
    q.add(start);//将 s 作为起始顶点加入队列
    dist.put(start, 0);
    int i=0;
    while(!q.isEmpty())
    {
      char top=q.poll();//取出队首元素
      i++;
      System.out.println("The "+i+"th element:"+top+" Distance from s is:"+dist.get(top));
      int d=dist.get(top)+1;//得出其周边还未被访问的节点的距离
      for (Character c : graph.get(top)) {
        if(!dist.containsKey(c))//如果 dist 中还没有该元素说明还没有被访问
        {
          dist.put(c, d);
          q.add(c);
        }
      }
    }
  }

  private static void dfs(HashMap<Character , LinkedList<Character>>
graph,HashMap<Character, Boolean> visited)
  {
    visit(graph, visited, 'u');//为了和图中的顺序一样，我认为控制了 DFS 先访问 u 节点
    visit(graph,visited,'w');
  }
  private static void visit(HashMap<Character , LinkedList<Character>>
graph,HashMap<Character, Boolean> visited,char start)
  {
    if(!visited.containsKey(start))
    {
      count++;
      System.out.println("The time into element "+start+":"+count);//记录进入该节点的时间
      visited.put(start, true);
      for (char c : graph.get(start))
      {
      if(!visited.containsKey(c))
      {
        visit(graph,visited,c);//递归访问其邻近节点
      }
      }
      count++;
      System.out.println("The time out element "+start+":"+count);//记录离开该节点的时间
    }
  }
```