\*\* 给定一个二叉搜索树T和给定的整数值V，找到T中值和V值最接近的节点。给出算法的时间和空间复杂度。

struct Node {

Node \*left;

Node \*right;

int value;

}

Node\* findNode(Node\* root, int value){

if(root == null){

return null;

}

Node\* leftValue = null;

Node\* rightValue = null;

Node \* target = null;

find(root,&leftValue, &rightValue, value,target);

return target;

}

void find(Node\* node, Node\* left, Node\* right, int value, Node\* rtv){

if(node.value == value){

rtv = node;

return rtv;

}else{

if(node == null){

if(left != null && right != null){

if(\*right.value - value < value - \*left.value){

return right;

}else{

return left;

}

}else if(left == null){

return right;

}else{

return left;

}

}else{

if(node.value < value){

left = node;

find(node.right,left,right,value,rtv);

}else{

right = node;

find(node.left,left,right,value,rtv);

}

}

return;

}

}

时间复杂度（O（logn）），空间复杂度 O（1）

\*\* 输入是一个非常大的英文文档（比如1T），单词之间用空格分开。请找到里面所有的具有如下特征的单词：该单词的字母顺序进行某种变化后，依然会出现在这个文档里面。比如如果post和stop都出现在文档中，就把他们找出来。

ArrayList<String> findWords(String cot){

HashSet<Word> checkContainer = new HashSet<Word>();

HashMap<String> rtvContainer = new HashMap<String>();

int index= 0;

while(findNextWord(cot,index)!= null){

String word = cot.subString(findNextWord(cot,index)[0],findNextWord(cot,index)[1]);

Word temp = new Word(word);

if(checkContainer.add(word) == false){

rtvContainer.add(word);

}

}

return rtvContainer;

}

int[] findNextWord(String cot, int index){

int[] rtv = new int[2];

int temp = index;

int start = 0;

String word = null;

while(temp < cot.length && cot.charAt(temp) == ' '){

temp++;

}

start = temp;

if(start == cot.length){

return null;

}

while(temp < cot.length && cot.charAt(temp) != ' '){

temp++;

}

word = cot.subString(start, temp-1);

rtv[0] = start;

rtv[1] = temp-1;

return rtv;

}

Word{

string value;

word(string value){

this.value = value;

}

public int hashCode(){

int hash = 0;

for(int i = 0 ; i < value.length;i++){

hash += value.charAt(i) - 'a';

}

return hash;

}

public boolean equals(Object obj){

if(obj.instanceOf(Word)){

if(obj.hashCode == this.hashCode){

return true;

}

}

return false;

}

}

\*\* 给定一个无序整数数组，找到所有满足以下条件的数字: 这个数字总是比它左边的数字大，比右边的数字小

ArrayList<Integer> findNumbers(int[] cot){

if(cot == null || cot.length == 0){

return null;

}

ArrayList<Integer> rtv = new ArrayList<Integer>();

int start = 0;

int end = cot.length-1;

rtv= findNumbers(cot,start,end);

retutn rtv;

}

ArrayList<Integer> findNumbers(int[] cot, int start, int end){

ArrayList<Integer> thisLayer = new ArrayList<integer>();

ArrayList<Integer> left;

ArrayList<Integer> right;

if(start == end){

thisLayer.add(cot[start]);

return;

}else if(end - start = 1){

if(cot[start] < cot[end]){

thisLayer.add(cot[start]);

thisLayer.add(cot[end]);

}else{

int temp = cot[start]; //sort 2 nums

cot[start] = cot[end];

cot[end] = temp;

}

return;

}

int mid = (start + end)/2;

left = findNumbers(cot, start, mid);

right = findNumbers(cot, mid+1,end);

for(int i = 0; i< left.size(),i++){

if(left.get(i) < cot[mid+1]){

thisLayer.add(left.get(i));

}

}

for(int i = 0; i< right.size(),i++){

if(right.get(i) > cot[mid]){

thisLayer.add(right.get(i));

}

}

sort(cot,start,mid,mid+1,end)； //完成对cot两段（start~mid 及 mid+1~end）已经有序的子数组的排序

return thisLayer;

}

时间复杂度 O(nlogn) 空间复杂度 O(n)——在sort函数中会用到；

请给出尽量优化的算法，并且给出时间复杂度的分析