Problem 1:

No. the first element could have value x but it doesn’t check the first element at all.

deleteAll(x)

a=head

while(a->next!=Null){

if(a->value==x){

a=a->next}}

Problem2:

Queue(stack s1, stack s2):

dequeue(){//let’s say dequeue/pop an element from s1

if(s1.empty()==True){

exit;}

else{

while(s1.empty()!=True){

s2.push(s1.peek());

s1.pop();

}

S2.pop();

While(s2.empty()!=True){

S1.push(s2.peek());

S2.pop();

}

}

}

enqueue(int x){

while(s1.empty()!=True){

s2.push(s1.peek());

s1.pop();}

s1.push(x);

while(s2.empty()!=True){

s1.push(s2.peek());

s2.pop();}

}

Problem 3:

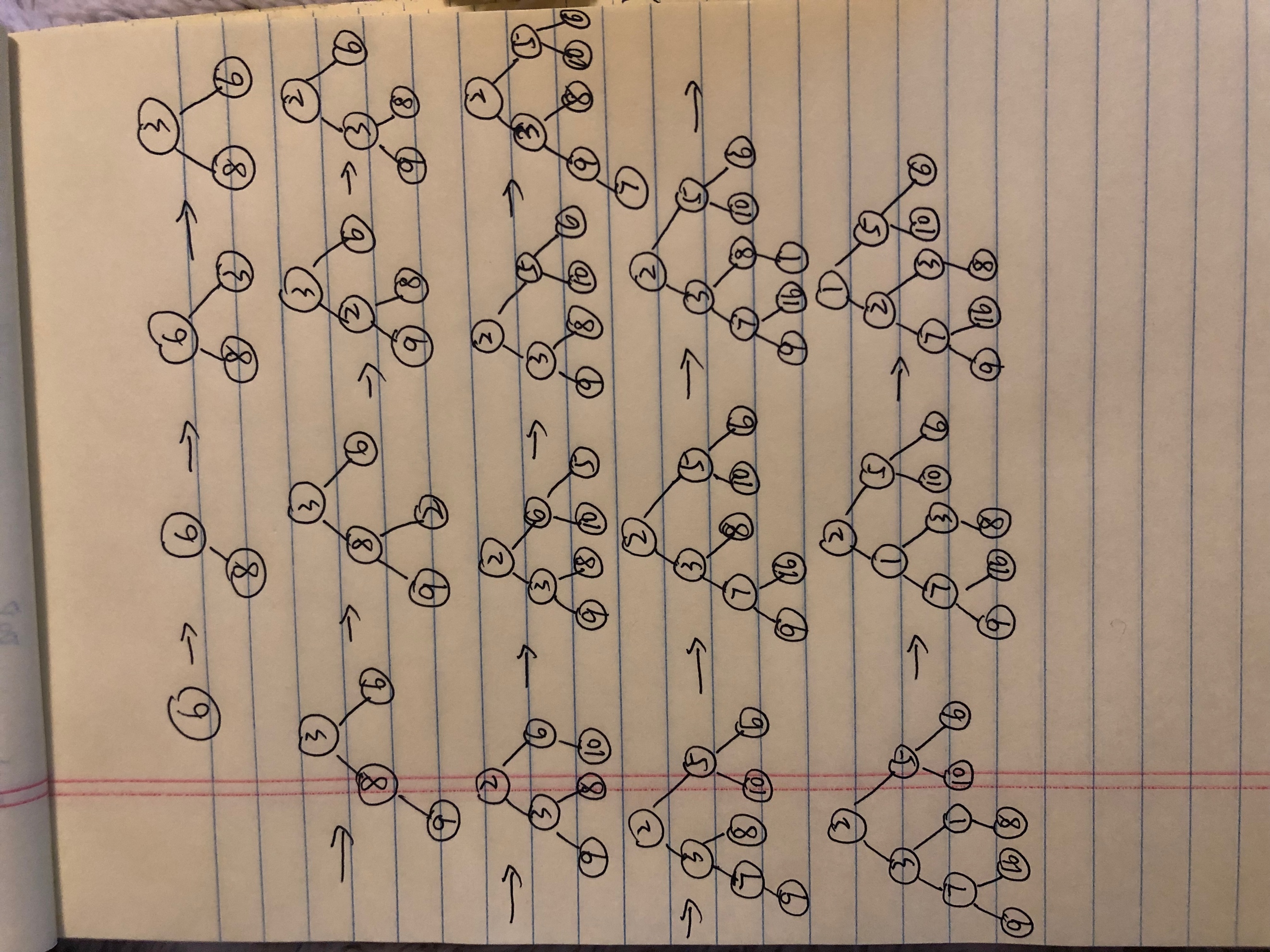
To implement 3 stacks with only one array, make one stack starts from the start of the array and another from the end of the array. The third on from the middle of the array.

If pushing a new element into the left stack where there’s already another element of the middle stack, simply move the entire middle stack to the right side by one, and then push the new element into the left stack.

If pushing a new element into the middle stack where there’s already another element of the left stack or the right stack, simply move the middle stack to the opposite side by one, and then push the new element into the middle stack.

If pushing a new element into the right stack where there’s already another element of the middle stack, simply move the middle stack to the left by one and then push the new element into the middle stack.

Problem 4:



Problem 5:

BST(int arr[], int start, into end){

If(end<start){

Return null;}

Int mid= (start+end)/2;

Node node=new Node(arr[mid])

Root->left=BST(arr,start,mid-1)

Root->right=MakeSubTree(arr, mid+1,end)

Return node;

}