1) Fill the array from the ends.

2)

```
public static Stack<Integer> reverseStack(Stack<Integer> in){
	Stack<Integer> out = new Stack<Integer>();
	Stack<Integer> tmp=new Stack<Integer>();
	while(!in.isEmpty()){
		Integer in1=in.pop();
		out.push(in1);
		tmp.push(in1);
	}
	while(!tmp.isEmpty()){
		in.push(tmp.pop());
	}
	return(out);
}
//Complexity is O(n),
```

3) Use a doubly linked list which maintains a tail pointer. Switch the head and tail pointers. Keep track of whether the list is in "reversed" state or not in order to know whether to follow the next or prev pointers when traversing to a certain position.

4) In the course of the algorithm we expect to see a certain kind of bracket on the stack and if we don't, then that's where the missing bracket should be. This is one such heuristic solution, there can be several others too.

5) Prints the binary representation of the given number.

Extra credit:

```
public static LLStack<Integer> transfer(LLStack<Integer> input){
	Integer tmp;
	Integer sizeinorig=0;//used to determine the size of the original input stack
	Integer sizein=0;//used to keep track of current size of input stack.
	LLStack<Integer> output=new LLStack<Integer>();
	while(!input.isEmpty()){
		output.push(input.pop());
		sizein++;
	}
	sizeinorig=sizein;//this is the original size of the input stack.
	tmp=output.pop();//this is the last element of the input stack.
	while(!output.isEmpty()){
		input.push(output.pop());
	}
	output.push(tmp);//now, output has one element, the last element, in the correct place.
	sizein--;//now the input stack has one less element than before.
	for(int i=0;i<sizeinorig;i++){
		while(!input.isEmpty()){
			output.push(input.pop());
		}
		tmp=output.pop();
		for(int j=0;j<sizein-1;j++){
```

```
                    input.push(output.pop());
                }
                output.push(tmp);
                sizein--;
        }
        return(output);
        }
//In total, this method uses a number of operations equal to
//8+2*n+n-1+n*(2*(1+2+...+n)+3) = 3*n+7+n*(n*(n+1)+3)
//which is O(n^3)
```