Problem 1.

Option 2 and 4 are not possible because the order of the values in the array do not fit any possible order of inputs.

Problem 2.

```
public BST(double a[])
{
	Node* root = null;

	Arrays.sort(a); //nlogn
	insertmidpoint(root, a, a.length);
}

public void insertmidpoint(Node* root, double* arr, int size)
{
	if(arr.length == 1){
		root = new Node(a[0]);
	}

	midpoint = size/2;
	root =  new Node(a[midpoint]);
	insertmidpoint(root.left, arr, midpoint);
	insertmidpoint(root.right, &arr[midpoint+1], size-midpoint-1);
}
```

Problem 3.

This can be implemented efficiently with a List data structure because it contains all the functions that Josie needs.

Problem 4.

We need a Data Structure such as a BST, where each node stores an interval. Each node however also includes the farthest left and farthest right interval of their children. Using this special BST,

we can easily search the count of intervals containing a value x by parsing through the tree with ease. We check each node if its interval contains x, and then by knowing the extreme values of the child nodes can determine if we check said interval for the number x.

Problem 5.

The insertion of 1, 2 would lead to a red-black binary tree to have the height 2 (*2*log(2)=2*). It also works to insert any two numbers because if the tree has two nodes it has it's maximum height for its number of n.