Analyze the big-oh run times (in terms of n) for each of the following recursive functions:

```
void A(int n)
{
	if (n < 10)
		time++;
	else
	{
		for (int i = 0; i < 7; i++)
			A(n / 2);
		for (int i = 0; i < n*n; i++)
			time++;
	}
}


void B(int n)
{
	if (n < 10)
		time++;
	else
	{
		for (int i = 0; i < 8; i++)
			B(n / 2);
		for (int i = 0; i < n*n; i++)
			time++;
	}
}

void C(int n)
{
	if (n < 10)
		time++;
	else
	{
		C(n / 5);
		C(7*n / 10);
		for (int i = 0; i < n; i++)
			time++;
	}
}

void D(int n)
{
	if (n < 15)
		time++;
	else
	{
		D(n / 2);
		for (i = 0; i < n; i++)
			time++:
		D(n / 2);
	}
}
```

```
void E(int n)
{
	if (n < 15)
		time++;
	else
	{
		E(n / 3);
		for (i = 0; i < n; i++)
			time++:
		E(2 * n / 3);
	}
}

void F(int n)
{
	if (n < 20)
		time++;
	else
	{
		time++;
		F(n / 2);
		F(n / 2);
	}
}


void G(int n)
{
	if (n < 20)
		time++;
	else
	{
		time++;
		G(n / 2);
	}
}


void H(int n)
{
	if (n < 20)
		time++;
	else
	{
		for(int i=0; i<n; i++)
			time++;
		G(n - 5);
	}
}
```

SUS Pokemon: Design an efficient algorithms to find an honest pokemon.

You are a pokemon trainer and have just ordered a fresh batch of *n* pokemon to help you with various daily tasks. However, you know that some pokemon are liars. Your first task is to figure out which of your *n* pokemon are honest (never lie), and which are SUS (may sometimes lie). Fortunately, the pokemon themselves know who are SUS, and thus you have the following tool available to you: You may combine any two pokemon and have them state whether the other pokemon is honest or SUS. An honest pokemon will always report correctly what the other pokemon is, but SUS pokemon may or may not tell the truth. Thus, for any pair of pokemon X and Y that you test, you may conclude the following:

If X and Y each say the other is honest, then either both are honest, or both are SUS.

If X says Y is honest, but Y says X is SUS (or vice-versa), then at least one is SUS.

If X and Y each say the other is SUS, then at least one is SUS.

1. Show that if half or more of the pokemon are SUS, it is impossible to determine who the honest pokemon are.
2. Assume that more than half of the pokemon are honest. Now consider the problem of finding one single honest pokemon from the given *n* pokemon. Design a way to reduce this problem to at most half the size by applying roughly $n/2$ pokemon comparisons.
3. Apply the above technique to design a divide-and-conquer algorithm for identifying all the honest pokemon in $O(n)$ pokemon comparisons. To analyze your run time, state and solve a recurrence equation describing the maximum number of comparisons made by your algorithm.