Inheritance

Problem 1: What's the problem with the code?

```
#include <iostream>
using namespace std;
class A
{
public:
    virtual void print(){cout << "A::print()";}
    virtual void doSomeThing() const = 0;
};

class B:public A
{
public:
    void print(){cout << "B::print()";}
};
```

Problem 2: What's the output?

```
#include <iostream>
using namespace std;
class A
{
public:
    virtual void print(){cout << "A::print()"<<endl;}
    virtual ~A(){};
};

class B:public A
{
public:
    void print(){cout << "B::print()"<<endl;}
};

void printSomething(A a){ a.print();}
void printSomething2(A& a){ a.print();}
void printSomething3(A* a){ a->print();}

int main() {
    B b;
    A* c = new B;
    printSomething(b);
    printSomething2(b);
    printSomething3(&b);
    printSomething(*c);
    printSomething2(*c);
    printSomething3(c);
}
```

Problem 3: What's the output?

```
#include <iostream>
using namespace std;
class A
{
public:
    A(){cout << "A()" << endl;}
    A(int x){cout<< "A(" << x << ")" << endl;}
    ~A(){cout << "~A()" << endl;}
};

class B
{
public:
    B(){cout << "B()" << endl;}
    B(int x):m_a(x){cout << "B(" << x << ")" << endl;}
    ~B() {cout << "~B()" << endl;}
private:
    A m_a;
};

class C:public A
{
public:
    C():A(10), m_b2(5){ cout << "C()" << endl;}
    ~C(){ cout << "~C()"<< endl;}
private:
    B m_b1;
    B m_b2;
};

int main() {
    C c;
}
```

Recursion

Example 1:

```
// Return the factorial of n using recursion
// Assume n is a nonnegative integer
int fact(int n)
{







}
```

Example 2:

```
// Return whether the array of size n contains the target
bool contains(const int a[], int n, int target)
{






}
```

Example 3: a = {4, 3, 5, 2, 1, 8, 7, 6}

```
void merge(int a[], int b, int mid, int e)
{
// Assume we can implement this function such that we can merge 2
sorted passes within e computations
// e.g. int a[] = {1, 3, 5, 2, 4, 6}, merge(a, 0, 3, 6), we get a[] =
{1, 2, 3, 4, 5, 6}
}

// Sort the array elements a[b] through a[e-1]
void MergeSort(int a[], int b, int e)
{
    if (e - b > 1){
        int mid = (b+e)/2;
        MergeSort(a, b, mid);
        MergeSort(a, mid, e);
        merge(a, b, mid, e);
    }
}
```

Example 4:

```
// Return a^b
// Assume b is a nonnegative integer
int expon(int a, int b)
{










}
```

Example 5:
```
// Return fibonacci(n)
int fab(int n)
{










}
```

Example 6:
```
// You can go either 1 or 2 steps each time.
// How many ways are there for you to go n steps?
int step(int n)
{










}
```

Example 7:
```
//Only one disk can be moved at a time.
//Each move consists of taking the upper disk from one of the stacks
and placing it on top of another stack i.e. a disk can only be moved
if it is the uppermost disk on a stack.
//No disk may be placed on top of a smaller disk.
void solveHanoi(int n, int source, int dest, int buffer)
{










}
```