

## 1. Insertion Sort with Recursion (15 points)

### Solution:

```
#include <vector>
using namespace std;

void ISRecur (int A[], int n)
{
    if (n>1){
        ISRecur (A, n-1); // Sort array[0..n-1]

        int posn= n-2;    // posn is index where last will be inserted
        int key = A[n-1];

        // Search for first array (from rear) <= array[last]
        while (posn >= 0 && key < A [posn]){
            A[posn+1] = A[posn]; //shift larger element
            posn--;
        }

        // insert element into proper position
        A[posn] = key;
        // now array[0..last] are in order
    }
}
```

## 2. Validity Checking for n-Queen Problem Using STL vector(17 points)

a) (6 points)

**Solution:**

```
bool ValidRow(const Vect& r){

    int size = r.size();    // get the size of vector

    for(int i=0;i< size;i++)
        for (int j=i+1; j< size; j++)
            if(r[i]==r[j])
                return false;

    return true;

}
```

b) (7 points)

**Solution:**

```
bool ValidDiag(const Vect& r){

    int size = r.size();    // get the size of vector

    for(int i=0;i< size;i++)
        for (int j=i+1; j < size; j++)
            if(j-i==abs(c[j]-c[i]))
                return false;

    return true;

}
```

c) (4 points)

**Solution:**

```
bool isValid(const Vect& r) {

    return (ValidRow(r) && ValidDiag(r) );

}
```

### 3. Implementing Stacks and Queues Using Inheritance and Polymorphism

(23 points)

(a) (7 points)

Solution:

```
MyArray::MyArray():size(0), data(0){
}

MyArray::~~MyArray()
{
    delete[] data;
}

void MyArray::push(int elem){
    int* newData=new int[size+1]; // allocate a larger array
    for (int i=0; i<size; i++)
        newData[i] = data[i]; // copy elements over
    newData[size]=elem;
    if(data)
        delete[] data; // release the old memory
    data = newData;
    size+=1;
}
```

(b) (6 points)

Solution:

```
class MyStack : public MyArray{
public:
    int pop()
    {
        if(size==0)
            return -1; // pop error
        int* newData=new int[size-1]; //allocate new array of smaller
size
        int value = data[size-1];
        size-=1;
        for(int i=0;i<size;i++)
        {
            newData[i]=data[i];
        }
        delete[] data;
        data=newData;
        return value;
    }
};
```

(c) (6 points)

Soution:

```
class MyQueue : public MyArray{
public:
    int pop()
    {
        if(size==0)
            return -1;
        int* newData=new int[size-1];
        int value = data[0];
        size-=1;
        for(int i=0; i < size; i++)
        {
            newData[i] = data[i+1];
        }
        delete[] data;
        data=newData;
        return value;
    }
};
```

(d) (4 points)

Solution:

```
Void printAndClear(MyArray* stkQ){
    int value = stkQ->pop();
    while (value!=-1){
        cout << value << " ";
        value = stkQ->pop();
    }
}
```

#### 4. List Implementation Using Template and Overloading (25 points)

(a) (6 points)

**Solution:**

In List class, add:

```
List<T> operator+(T) const;

template<typename T>
List<T> List<T>::operator+(T newValue) const{
List<T> result = *this;
    result.list.push_back(newValue);
    return result;
}
```

(b) (6 points)

**Solution:**

In List class, add:

```
template<class U>
friend List<U> operator+(U, const List<U>&);

//global function
template<class T>
List<T> operator+(T newValue, const List<T>&list){
List<T> result = list;
    result.list.push_front(newValue);
    return result;
}
```

(c) (6 points)

**Solution:**

```
In List class, add:
    List<T> operator+(const List<T>&) const;

template<typename T>
List<T> List<T>::operator+(const List<T>&lst) const{

    List<T> result = *this;
    typename deque<T>::const_iterator it = lst.list.begin();

    while(it != lst.list.end()){
        result.list.push_back(*it);
        it++;
    }
    return result;
}
```

(d) (7 points)

**Solution:**

```
In List class, add:
List<T>& operator--();

template<typename T>
List<T>& List<T>::operator--(){
    if(!list.size()){//NULL list
        // cout<<"NULL List"<<endl;
        exit(-1);
    }
    else
        list.pop_front();
    return *this;
}
```

## 5. Binary tree (20 points)

(a) (5 points)

**Solution:**

```
void BT::deleteNodes(Node* node) {
    if( node ){
        deleteNodes(node->left);
        deleteNodes(node->right);
        delete node;
    }
}
```

(b) (5 points)

**Solution:**

```
void BT::copyNodes(const Node const* src, Node* &dest){
    if ( src ==NULL )
        return;

    dest = new Node(src->data, src->level);
    copyNodes(src->left, dest->left);
    copyNodes(src->right, dest->right);
}
}
```

(c) (5 points)

**Solution:**

```
int BT::CountINodes(Node* nptr){

    if ( nptr == NULL )
        return 0;
    if (nptr->left == NULL&& nptr->right == NULL) // a leave
        return 0;
    return ( CountINodes(nptr->left)
            + CountINodes(nptr->right) + 1); //add itself as it is
                                           //an internal node
}
```

(d) (5 points)

**Solution:**

```
void BT::assignNodeLevel( Node* nptr, int level){
    nptr->level = level;
    if (nptr->left)
        assignNodeLevel(nptr->left, level+1);
    if (node->right)
        assignNodeLevel(nptr->right, level+1);
}
```

Note: Another non-recursive implementation is to use queue and level-order traversal