CS106B Final ANSWER KEY, Spring 2023

Problem 1: Recursive Backtracking

```
void printTripletsHelper(Vector<int>& vec, int targetSum, Vector<int>& tripletVec, int sum, int index) {
    if (tripletVec.size() == 3) {
        if (sum <= targetSum) {
             cout << tripletVec << endl;
        }
        return;
    }

    if (index == vec.size()) {
        return;
    }
    // get the first value in vec
    int nextVal = vec[index];

    // recurse without the value
    printTripletsHelper(vec, targetSum, tripletVec, sum, index + 1);

    // use it
    tripletVec.add(nextVal);

    // recurse with the value
    printTripletsHelper(vec, targetSum, tripletVec, sum + nextVal, index + 1);

    // undo
    tripletVec.remove(tripletVec.size() - 1);
}

void printTriplets(Vector<int> vec, int targetSum) {
    Vector<int>tripletVec;
    printTripletsHelper(vec, targetSum, tripletVec, 0, 0);
}
```

Problem 2: Classes

2a.

```
private:
    void swapElements(int index1, index2);
    int _numFilled;
    int _numAllocated;
    string *_elements;
```

2b.

```
CuteQ::CuteQ(int size) {
    _numFilled = 0;
    _numAllocated = size;
    _elements = new string[_numAllocated];
}

CuteQ::~CuteQ() {
    delete[] _elements;
}

int CuteQ::size() const {
    return _numFilled;
}

void CuteQ::enqueue(string student) {
    _elements[_numFilled] = student;
    _numFilled++;
}

string CuteQ::dequeue() {
    if (_numFilled == 0) error("Can't dequeue from an empty CuteQ!");

    string res = _elements[0];

    for (int i = 1; i < _numFilled; i++) {
        swapElements(i, i - 1);
    }

    _numFilled--;
    return res;
}
```

Problem 3: Linked Lists

```
void moveToFront(Cell*& front, Cell* toMove) {

    // Check if toMove is at front of list
    if (front == toMove) {
        return;

    }

    // Check if toMove is at end of list
    else if (toMove->next == nullptr) {

        // We know toMove->prev is not nullptr since
        // if it was, toMove would be front
        toMove->prev->next = nullptr;
    }

    // toMove is somewhere in the middle
    else {

        Cell* before = toMove->prev;
        Cell* after  = toMove->next;

        // Slice out toMove
        before->next = after;
        after->prev = before;
    }

    // Move to front
    toMove->next = front;
    toMove->prev = nullptr;
    front->prev = toMove;
    front = toMove;
}
```

Problem 4: Trees

Highest product of siblings in a binary tree

```
int highestSiblingProduct(TreeNode *root) {
    if (root == nullptr) {
        return 0;
    }
    int childrenMax = max(highestSiblingProduct(root->left),
                          highestSiblingProduct(root->right));
    int childrenProduct;
    if (root->left == nullptr or root->right == nullptr) {
        childrenProduct = 0;
    } else {
        childrenProduct = root->left->data * root->right->data;
    }
    return max(childrenMax, childrenProduct);
}
```

Ternary Tree: sum of all nodes

```
struct TernaryNode {
    int data;
    TernaryNode *left;
    TernaryNode *middle;
    TernaryNode *right;
};

int sumNodes(TernaryNode *root) {
    if (root == nullptr) {
        return 0;
    }
    return root->data + sumNodes(root->left) + sumNodes(root->middle) + sumNodes(root->right);
}
```

Problem 5: Multiple Choice

5.1: A) Insertion Sort

5.2: D) Small changes in the input data produce a significant change in the resulting hash value.

5.3:

		A	B	C	D	E	F
D)	Distance to A	0	4	5	12	8	15
	From	-	A	A	B	C	E
	Visited?	Y	Y	Y	N	Y	N