Practice Final Solutions

1. Heap

Diagram after inserting 15:

```
   15
```

Diagram after inserting 10:

```
  10
 /  \
15  15
```

Diagram after inserting 13:

```
  10
 /  \
15  13
```

Diagram after inserting 8:

```
  8
 /  \
10  13
 /    \
15  15
```

Diagram after inserting 2:

```
  2
 /  \
 8  13
 /    \
15  10
```

Diagram after inserting 9:

```
  2
 /  \
 8  9
 /    \
15 10 13
```
Diagram after calling dequeue once:

```
  8
  /|
 /  |
10  9
  |
  \
15
```

Diagram after calling dequeue twice:

```
  9
  /|
 /  |
10  13
  |
  \
15
```

2. Linked Lists

```cpp
def switchPairsOfPairs(ListNode* &front):
    if (front && front->next && front->next->next && front->next->next->next):
        # have: front -> 1 -> 2 -> 3 -> 4 -> 5...; want: front -> 3 -> 4 -> 1 -> 2 -> 5...
        ListNode* curr = front->next->next;  // curr -> 3
        front->next->next = curr->next->next;  // 2 -> 5
        curr->next->next = front;  // 4 -> 1
        front = curr;  // front -> 3
        curr = curr->next->next->next;  // curr -> 2
        # have: curr -> 2 -> 5 -> 6 -> 7 -> 8 -> 9 ...; want: curr -> 2 -> 7 -> 8 -> 5 -> 6 -> 9 ...
        while (curr->next && curr->next->next && curr->next->next->next && curr->next->next->next->next):
            ListNode* temp = curr->next->next->next->next;  // temp -> 7
            curr->next->next->next->next = temp->next->next;  // 6 -> 9
            temp->next->next = curr->next;  // 8 -> 5
            curr->next = temp;  // 2 -> 7
            curr = temp->next->next->next;  // curr -> 6
    }
```

3. Recursive Backtracking

Part A:

```cpp
bool placeHoriz(Grid<char>& board, int size, int row, int col)
{
    for (int i = 0; i < size; i++) {
        if (!board.inBounds(row, col + i) || board[row][col + i] != '?') {
            return false;
        }
    }
    for (int i = 0; i < size; i++) {
        if (board.inBounds(row, col + i)) {
            board[row][col + i] = 'B';
        }
    }
    return true;
}
```
Part B:

```cpp
bool canPlaceShips(Grid<char>& board, Vector<int> shipSizes) {
    if (shipSizes.size() == 0) {
        return true;
    }
    int shipSize = shipSizes[0];
    shipSizes.remove(0);
    for (int row = 0; row < board.numRows(); row++) {
        for (int col = 0; col < board.numCols(); col++) {
            if (placeHoriz(board, shipSize, row, col)) {
                if (canPlaceShips(board, shipSizes)) {
                    return true;
                }
                unplaceHoriz(board, shipSize, row, col);
            }
            if (placeVert(board, shipSize, row, col)) {
                if (canPlaceShips(board, shipSizes)) {
                    return true;
                }
                unplaceVert(board, shipSize, row, col);
            }
        }
    }
    return false;
}
```

4. Trees

```cpp
// Constructor (2pts)
KTree::KTree() {
    root = NULL; // or this->root = NULL;
}

// Destructor (6pts)
KTree::~KTree() {
    deleteHelper(root); // they add this, can be named anything
}

void deleteHelper(Node* curr) {
    if (curr != NULL) {
        deleteHelper(curr->left);
        deleteHelper(curr->right);
        delete curr;
    }
}

// Inserts key into the tree in the proper place, and updates all tree
// counts appropriately. Your solution must be recursive, using the
// provided helper. (7pts)
void KTree::addKey(int key) {
    if (root == NULL) {
        root = new Node(key);
    } else {
        addKeyHelper(key, root);
    }
}```
Recursive helper function for addKey. Returns true if node was added, false if key was duplicate so no add was done. The code for a standard BST insert is already provided here for you. You should edit this code to make it work for k-ordered tree. Write your additional line(s) of code to the right and use arrows to indicate where to insert your addition(s). Cross out any code you want to delete.

```cpp
bool addKeyHelper(int key, Node* curr) {
    if (key < curr->key) {
        if (curr->left == NULL) {
            curr->left = new Node(key);
            curr->count++;
            return true;
        } else {
            if (addKeyHelper(key, curr->left)) {
                curr->count++;
                return true;
            } else {
                return false;
            }
        }
    } else if (key > curr->key) {
        if (curr->right == NULL) {
            curr->right = new Node(key);
            return true;
        } else {
            return addKeyHelper(curr->right);
        }
    } else {
        return false;
    }
}
```

3 possible solutions for kthKey
O(logN) (full credit)
```cpp
int kthKeyHelper(int k, Node* curr) {
    if (k == curr->count) {
        return curr->key;
    }
    if (k < curr->count) {
        return kthKeyHelper(k, curr->left);
    }
    if (k > curr->count) {
        return kthKeyHelper(k - curr->count - 1, curr->right);
    }
}
```

O(N) no aux data structs (small deduction)
```cpp
void kthKeyHelper(int k, Node* curr, int& countSoFar, int& retVal) {
    if (curr != NULL) {
        kthKeyHelper(k, curr->left, countSoFar, retval);
        if (k == countSoFar) {
            retVal = curr->key;
        }
        countSoFar++;
        kthKeyHelper(k, curr->right, countSoFar, retVal);
    }
    return 0;
}
// O(N) with aux data structs (larger deduction in this version the wrapper
// will receive the Vector populated with keys of the tree, and then just
// pull out nums[k])
void kthKeyHelper(int k, Node* curr, Vector<int>& nums) {
    if (curr != NULL) {
        kthKeyHelper(k, curr->left, countSoFar);
        nums.add(curr->key);
        // "optimization" if (nums.size() < k)
        kthKeyHelper(k, curr->right);
    }
    return 0;
}

5. Searching and Sorting

(a) Indexes examined: 7, 3, 5, 6 Value returned: -1

(b) {29, 17, 3, 94, 46, 8, -4, 12}
    {-4, 17, 3, 94, 46, 8, 29, 12}
    {-4, 3, 17, 94, 46, 8, 29, 12}
    {-4, 3, 8, 94, 46, 17, 29, 12}

(c) {29, 17, 3, 94, 46, 8, -4, 12}
    {17, 29, 3, 94, 46, 8, -4, 12}
    { 3, 17, 29, 94, 46, 8, -4, 12}
    { 3, 17, 94, 46, 8, -4, 12}

(d) {29, 17, 3, 94, 46, 8, -4, 12}
    {29, 17, 3, 94} {46, 8, -4, 12} split
    {29, 17} { 3, 94} split
    {29} {17} split
    {17, 29} merge
    { 3} {94} split
    { 3, 94} merge
    { 3, 17, 29, 94} merge
    {46, 8} {-4, 12} split
    {46} { 8} split
    { 8, 46} merge
    {-4} {12} split
    {-4, 12} merge
    { -4, 8, 12, 46} merge
    {-4, 3, 8, 12, 17, 29, 46, 94} merge
6. Hashing

Rehashing triggers when $i = 2$. Load factor = 0.6 (6 elements over 10 buckets).
In the enlarged table of 30 buckets, entries 4:2 and 34:26 are both in bucket[4].

7. Graphs

a) directed
b) weighted
c) DFS: examines A, B, F, (backtracks to A), E, H, I
   returns path {A, E, H, I}
d) BFS: examines C, (neighbors of C) B, G, (neighbors of B) A, F, (neighbors of G) J,
   (neighbors of A) E, (neighbors of E) H
   returns path {C, B, A, E, H}

8. Big-Oh

add:
  sorted array: $O(N)$
  binary max-heap: $O(\log N)$
  balanced BST: $O(\log N)$

contains:
  sorted array: $O(\log N)$
  binary max-heap: $O(N)$
  balanced BST: $O(\log N)$

max:
  sorted array: $O(1)$
  binary max-heap: $O(1)$
  balanced BST: $O(\log N)$