Solutions to Homework 1: Basic Data Structures and Trees

Solution 1:  See Fig. 1(a) and (b).

Figure 1: Solution to Problem 1.

Solution 2:  See Fig. 2.

Figure 2: Solution to Problem 2.

Solution 3:

(a) Given $i$, $j$, and $n$, if either $i$ or $j$ is even, we return zero. Otherwise, let $i' \leftarrow \lfloor i/2 \rfloor$ and $j' \leftarrow \lfloor j/2 \rfloor$. Observe that each odd row and each odd column stores $n' \leftarrow \lfloor n/2 \rfloor$ nonzero entries. The values $i'$ and $j'$ can be viewed as entries in an $n' \times n'$ matrix. We can map these values to a linear array by setting $k \leftarrow i'n' + j'$. In summary we have the following pseudocode solution.

```c
int getValue(int i, int j, int n) {
    if (i % 2 == 0 || j % 2 == 0) return 0;
    else {
```
n' = floor(n/2);
i' = floor(i/2);
j' = floor(j/2);
return L[i' * n' + j'];
}

(b) First, observe that if both \( i \) and \( j \) are odd, we may infer that the value is zero and return this. Otherwise, it will simplify matters to treat even rows and odd rows separately. If \( i \) is even, then all \( n \) entries of this row appear in \( L \). If \( i \) is odd, then only the \( \lfloor n/2 \rfloor \) entries (those in the even-indexed columns) appear in \( L \).

Letting \( n' = n + \lfloor n/2 \rfloor \), it follows that every two rows of \( M \) correspond to \( n' \) entries of \( L \). Given an even row index \( i \), we infer therefore that all the \( i/2 \) previous pairs of rows contain a total of \( n'' = n'i/2 \) entries. Therefore, the index \( k \) corresponding to \( M[i][j] \) is \( n'' + j \).

On the other hand, if \( i \) is odd, then observe that we may assume that \( j \) is even (since otherwise we would have returned a value of zero). The previous rows contain a total of \( n'' = n'(i−1)/2 + n \) entries. (As reasoned above, there are \( n'(i−1)/2 \) entries in the rows preceding \( i−1 \), and \( n \) elements in row \( i−1 \) itself.) Since we are skipping every other entry, the index \( k \) corresponding to \( M[i][j] \) is \( n'' + \lfloor j/2 \rfloor \). Since we know \( j \) is even, we can just write \( j/2 \) in place of \( \lfloor j/2 \rfloor \).

Thus, we obtain the following solution:

```c
int getValue(int i, int j, int n) {
    if (i % 2 == 1 && j % 2 == 1) return 0; // both indices odd - zero entry
    else {
        if (i % 2 == 0) { // even row index
            n'' = n' * i / 2; // number of entries in rows 0 ... i-1
            return L[n'' + j]; // skip over j columns
        } else { // odd row index (we may assume j is even)
            n'' = n' * (i-1) / 2 + n; // number of entries in rows 0 ... i-1
            return L[n'' + j/2]; // skip over j/2 columns
        }
    }
}
```

**Solution 4:** Our solution is based on the following useful observation about inorder successors and predecessors. If \( p \) is the left child of some node \( q \), then \( q \) is the inorder successor of the rightmost node in the subtree rooted at \( p \), which might be \( p \) itself (see Fig. 3(a)). Conversely, if \( p \) is the right child of some node \( q \), then \( q \) is the inorder predecessor of the leftmost node in the subtree rooted at \( p \) (see Fig. 3(b)). Since we don’t know \( p \)’s parent, we don’t know which case holds, so we will try both. We descend the tree until reaching rightmost/leftmost leaf in \( p \)’s subtree, and then follow the appropriate thread. If either leads to a node whose child is \( p \), we are done. If both lead to a \( \text{null} \) pointer, then it must be that \( p \) has no parent, and hence is the root.

```c
BinaryNode parent(BinaryNode p) {
    BinaryNode v = p;
```
while (!v.right.isThread) v = v.right; // find rightmost in p’s subtree
BinaryNode q = v.right; // follow v’s thread to q
if (q != null && p == q.left) return q; // p is q’s left child? q is parent

v = p;
while (!v.left.isThread) v = v.left; // find leftmost in p’s subtree
BinaryNode q = v.left; // follow v’s thread to q
if (q != null && p == q.right) return q; // p is q’s right child? q is parent

return null; // both failed? p must be root