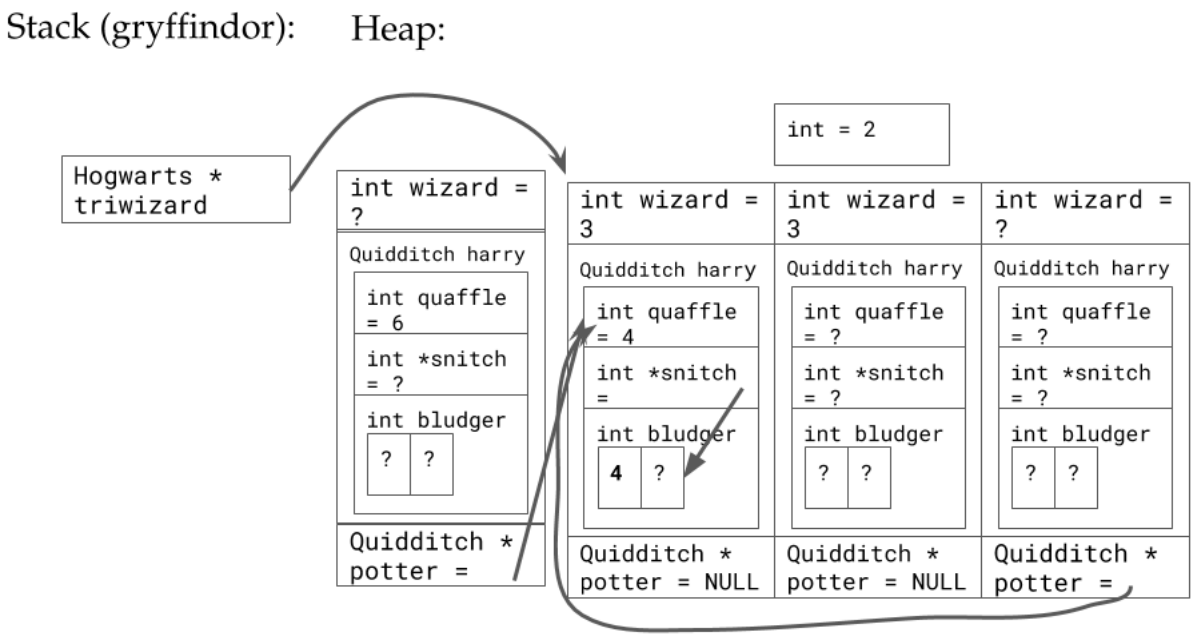
Pointers and Linked Lists: Solutions

1. Pointer Trace



2. Mirror (Arrays)

```
void ArrayIntList::mirror() {
    if (mysize * 2 > capacity) {
        // grow array to fit
        int* bigger = new int[2 * capacity];
        for (int i = 0; i < mysize; i++) {
            bigger[i] = elements[i];
        }
        delete[] elements;
        elements = bigger;
        capacity *= 2;
    }
    for (int i = 0; i < mysize; i++) {
        elements[2 * mysize - 1 - i] = elements[i];
    }
    mysize *= 2;
}
```

3. Sorted (Linked Lists)

```
bool isSorted(ListNode* front) {
    if (front != nullptr) {
        ListNode* current = front;
        while (current->next != nullptr) {
            if (current->data > current->next->data) {
                return false;
            }
            current = current->next;
        }
    }
    return true;
}
```

4. Stutter (Linked Lists)

```
void stutter(ListNode*& front) {
    ListNode* current = front;
    while (current != nullptr) {
        current->next = new ListNode(current->data, current->next);
        current = current->next->next;
    }
}
```

5. Braid (Linked Lists)

```
void braid(ListNode*& front) {
    // build reversed list
    ListNode* reverse = nullptr;
    for (ListNode* curr = front; curr != nullptr; curr = curr->next) {
        ListNode* newNode = new ListNode(curr->data);
        newNode->next = reverse;
        reverse = newNode;
    }

    ListNode* rest = reverse;   // part that has yet to be braided in
    for (ListNode* curr = front; curr != nullptr; curr = curr->next->next) {
        ListNode* next = rest->next;
        rest->next = curr->next;
        curr->next = rest;
        rest = next;
    }
}
```