Problem 1:
find and hasDuplicates
+1 if find isn’t explained as O(n), recurrence is best but something else might fly -- just O(n) not enough
+1 solve the recurrence you write
+2 have a recurrence, +1 for T(n-1), +1 for O(n)
With no recurrence, just reasoning that it’s n^2, -4

------

power: +3 recurrence +1 solution
for +3, T(n/2) is +2, O(1) is +1

arrange
+1 for O(n log n) sort
+1 for O(n) other work
+1 for that’s O(n log n), i.e., -1 if big-Oh terms not combined
+2 for T(n/2) in recurrence

----------------

problem 2
bool operator < (const Student& a, const Student& b) {
  if (a.gpa == b.gpa) return a.name < b.name; // +2 points
  return a.gpa < b.gpa                         // +1 point
}

Possible to get -1 if logic messed up but idea is correct for equal gpas.
return a.name < b.name
gets no credit, there’s no comparison of gpa’s

---
for honorRoll
+1 for initialization and return (either/both missing is -1)
+2 for calling new with state that makes sense: element of vector and something linking
+3 for building list that’s linked in the right order, if list is built backwards and can be fixed by looping down instead of up that’s -3.
+2 for comparing vector elements with honor. Comparing with 3.8 is -1

----------------

Problem 3:
Part A:
ant ant bat bat dog dog
+2 2 of each
+1 perfect (so -1 if dog dog bat bat ant ant)

---
ant ant ant bat bat bat dog dog dog
+2 four of each, looks like related to earlier answer
+1 perfect
T(n) = T(n-1) + O(1) --> O(n)
+1 for O(n)
+2 justification (non recurrence is ok)

Part B:
-----
ant bat dog +1 all or nothing
ant bat bat dog dog +2, based on response to part A/1.
+1 something close
+1 perfect
8 of each is worth +2
+1 7 or 8 of each in some order
+1 perfect

Part C:
n2^n with justification about doubling.
+1 for initial n
+2 for 2^n
+1 for justification.

A recurrence leading to n^2 gets +1 point (or -3)

----------------

Problem 4:
Part A:
must mention insert/delete at front for +3
and must mention O(n) or n-elements shifted, so not O(1)
-1 if O(n) or n not mentioned

Part B
+1 insert/delete at back
+2 delete <--- lose -2 if delete not singled out!

In other words, if only mention is find back and insert/delete both bad, this is -2. Must specifically cite delete as the problem to earn +3, no mention of delete specifically is -2.

Part C:
void Deque::push_back(const string& s) {
  if (myLast == 0) {
    myLast = myFirst = new Node(s,0,0);  // this case is +1/-1
  }
  else {  
    myLast->next = new Node(s,0,myLast);   // +1 assign to myLast->next
  }
Part D:

```cpp
    string Deque::pop_back() {
        string s = myLast->info; // +1/2 identify value to return
        myLast = myLast->prev; // +1 move to previous
        if (myLast) {
            myLast->next = 0; // +1/2 change next
        } else {
            myFirst = 0; // +1 identify this case
        }
        mySize--; // +1/2 decrement
        return s; // +1/2 return value
    }
```

Note: when size == 0 after pop, myFirst must be set to 0.

Part E:

+2 Must mention how to change body to return mySize (all or nothing)
+2 must mention O(n^2)
+2 must mention why it’s O(n^2)

if no n^2 mentioned, -2, if no reasoning, -2

----------

Question 5:

Hardly anybody messed up the simple link-list search. If people did it, they got it right. -1 if they failed to return 0 if not found.

Part B:

Here’s my solution. Note that *THIS WILL FAIL* if the team being moved is the first team in a conference list. In that case it’s hard to do the problem without special casing the removal and not calling removeNode. I had one student who noticed this, that student got extra credit and didn’t deduct for the solution below.

```cpp
    void ChangeConf(Conference* ncaa, string team, string fromConf, string toConf) {
        School * from = getSchools(ncaa, fromConf);
        School * to = getSchools(ncaa, toConf);
        School * change = findSchool(ncaa, from, team);
        removeNode(change);
        change->next = to->next;
        if (to->next != 0) {
            to->next->prev = change;
        }
        change->prev = to;
        to->next = change;
    }
```

+3 if all done right and used
(I didn’t deduct if getSchools code duplicated rather than calling)
+2 for calling findSchool and calling removeNode
+1 removeNode -- this could be lost if called after relinking which some did
+3 for relinking code. Basically there are four statements, missing one means -1, but you can’t lose more than 3. Missing the guard on to->next was -1/2, but most people didn’t do this, they ran to the end of the list to link the node. In that case, running off the end instead of stopping on the end is -1.