HW4

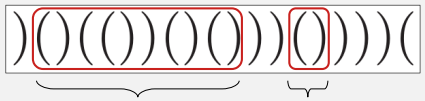
- approaching the problems
  - the stated running time goals and the section headers in the exercises provide strong directions as to the nature of the solutions
    - O(1) → accessing a slot in an array, accessing a stored value
    - O(log n) → binary search, balanced search tree, complete binary tree
    - O(n) → allows for traversal of most data structures

HW4

- level of detail in writeup
  - code/pseudocode alone is likely too detailed for the reader to understand the idea
  - but also be concise in your writeup
    - review both your ideas and your writeup for opportunities to simplify – and also to make sure you've provided enough explanation
  - do not need to describe standard operations – find, insert, delete – for standard data structures discussed in class but *do* need to describe other operations or variations on the standard
  - do need to explain how a structure is used to store the values
    - i.e. what is stored in the slots of an array, nodes of a linked list, or nodes of a tree

HW4

- #1 – ADM 3-2
  - make sure you understand the problem – looking for the sum total of sequences of balanced parens, not the longest such



HW4

- #2 – ADM 3-25
  - best-fit – find the bin with the smallest remaining space after the object is added means the bin with the least remaining space *of those with enough space for the object*

HW4

- #4 – ADM 4-41
  - the question is about *expected* performance, not worst case
    - compute the expected number of elements looked at in each scenario
  - the expected value for something with two alternatives is

    p alt1 + (1-p) alt2
  - for binary search, the expected number of elements looked at is $\log_2 n$ for both successful and unsuccessful searches
  - for sequential search, the expected number of elements looked at is $n/2$ for a successful search and n for an unsuccessful search
  - for the two-array scenarios, a regular customer search involves an unsuccessful search of the good customer array followed by a (hopefully) successful search of the regular customer array
  - big-Oh plays no part here – big-Oh addresses the growth rate of the running time but we are interested in comparing for specific values of n
    - it doesn't make sense to say O(log 10000) – that's saying the growth rate grows like log 10000, which is a constant

HW4

- #5 – ADM 3-11
  - hashtables are O(1) *expected* time – any given operation could be O(n) worst cast, though that is unlikely
  - the problem asks for O(1) *worst case* time

HW4

- #6 – ADM 3-18
  - storing additional information can achieve O(1) time the desired operations, but also need to address specifically *how* to update that stored information in other operations with enough detail to understand both the correctness of the update and its running time
  - insert/delete is already O(log n) for a balanced BST, so any additional O(log n) update is sufficient for the problem but it is worth explaining how to do it in O(1) time if possible