This homework covers applications of ADTs, elementary data structures (arrays and linked lists), and binary search trees. It is due in class Wednesday, February 11.

See the Policies page on the course website for information about revise-and-resubmit, late work, and academic integrity as it applies to homework.

Write your solutions carefully — your work should be neat, readable, and organized.

For "design a data structure" problems, describe what is stored and how and explain how each operation is carried out. For both data structure and algorithm design problems, if a target running time is given, also explain how your solution achieves it.

Hints for design problems:

- *Use the highest level of abstraction you can — in this case, the heading of the section containing the problem gives a big hint as to whether to be starting with ADTs or concrete data structures in thinking about the solution. Don't jump straight to arrays or linked lists when you could use a stack or queue, for example.*
- *Use target runtimes (when given) to help steer you towards a solution — keep in mind what you know about the running time of standard ADT, array, and linked list operations as well as the tactics discussed in class for improving runtimes (e.g. store instead of search).*

When writing up your solution, keep in mind that the goal is to express the algorithm/data structure idea and operations with sufficient detail to be understood and to be able to assess correctness and running time, but not to overwhelm the ideas with details that obscure understanding.

- *Give a brief overview of the idea before the details. This provides context and is especially important if the process is complex or requires a detailed algorithm.*
- *Avoid unnecessary detail — use the highest level of abstraction you can, don't explain known things, and don't provide implementation specifics if they don't affect the running time. For example, the Queue ADT is well-known so you don't need to explain how to implement enqueue or dequeue (just how those operations are used in your solution), and it is well-known that there is an implementation of Queue where the operations are $\Theta(1)$ so you don't need to identify whether the queue uses an array or a linked list. What is known depends on your audience, of course — write for your classmates, so standard things that have been discussed in class or in the reading do not need to be explained in your writeup.*

- *There is something of an art to good writeups and it takes practice — review (and revise) your writeup with an eye towards achieving the right level of detail.*

---

1. ADM 3-3, page 103.
2. ADM 3-4, page 103.
3. ADM 3-5, page 103. *Amortized cost* refers to averaging the cost of a series of operations — this is the kind of analysis done in the "grow an array by doubling its length" example in the "big-Oh from code" reading posted on the schedule page (readings from Wednesday 1/28). For (a), describe a sequence of n operations and do the analysis to show that the "bad amortized cost" is worse than constant time. For (b), describe a better underflow strategy and, do the analysis to show that the constant amortized time is achieved for the sequence of operations from (a) and for a sequence of n deletions, and explain why you can thus conclude that *any* sequence of n insert/delete operations will result in constant amortized time.
4. ADM 3-7, page 103. Write a body for the `remove` method below which runs in $\Theta(1)$ time using the strategy referenced in the problem and discussed in class. (Write actual Java code, not pseudocode or a description of the implementation.) Be sure to handle any special cases, such as removing the first element. Use the `ListNode` class given in figure 1.

   Note: this question is only asking you to write out the code — you don't have to implement and run it, though if you want to do so (e.g. for testing purposes), you can use the `ListNode` class posted on the schedule page (materials from Monday 2/2).

   ```
   /**
    * Remove the element todel.getElt() from the list.
    *
    * @param head head of the list
    * @param todel node containing the element to remove
    * @return the head of the list after the removal
    */
   public ListNode remove ( ListNode head, ListNode todel ) { ... }
   ```

5. ADM 3-8, page 103.
6. Do the homework #2 drill problems on Canvas. (Look for `hw2 drill` in the Quizzes section.)

```
public class ListNode {
  public ListNode ( int elt ) { ... }
  public ListNode ( int elt, ListNode next ) { ... }

  public int getElt () { ... }
  public void setElt ( int elt ) { ... }

  public ListNode getNext() { ... }
  public void setNext ( ListNode next ) { ... }
}
```

Figure 1: `ListNode` class for #4.