Written Assignment #2

Content Covered: PA2 Reflection, Asymptotic Runtime vs Constant Factors

---

Submission Process, Late Policy and Grading

Due Date: 4/2/23 @ 11:59PM
Total points: 30

Your written solution may be either handwritten and scanned, or typeset. Either way, you must produce a PDF that is legible and displays reasonably on a typical PDF reader. This PDF should be submitted via autolab as WA2. You should view your submission after you upload it to make sure that it is not corrupted or malformed. Submissions that are rotated, upside down, or that do not load will not receive credit. Illegible submissions may also lose credit depending on what can be read. Ensure that your final submission contains all pages.

You are responsible for making sure your submission went through successfully.

Written submissions may be turned in up to one day late for a 50% penalty.
No grace day usage is allowed.

Prelude

This written assignment is a reflection on PA2 and will measure your understanding of the runtime complexity of various operations on Linked Lists with and without the variations we introduced in PA2.

For all of the following problems, assume that the lists have n elements, and unless specified otherwise, your asymptotic runtimes should only be given in terms of n (ie no constants or variables other than n).

Problem 1 - Sorted Lists

[6/30 points]

For PA2, your list was required to store elements in sorted order.

a. **[2 points]** What is the unqualified worst-case complexity of finding an element in a regular linked list (one in which the items aren't stored in sorted order)?

b. **[2 points]** What is the unqualified worst-case complexity of finding an element in a linked list in which the elements are stored in sorted order?

c. **[2 points]** Depending on your answers to questions 1 and 2 explain why storing elements in sorted order ***does*** or ***does not*** affect the asymptotic complexity of finding an element?

Problem 2 - Hinted Search

[12/30 points]

For PA2, your list included hinted versions of the search functions. This gave these functions a different place to start their search from rather than the `headNode`.

Imagine we are using the `SortedLinkedList` from PA2 to store a list of users sorted by first name. We've devised a system for getting hints for the hinted version of insert and search: we create a separate `Array[Option[SortedListNode]]` of size 26 named `hints`. `hints[0]` stores a reference to the ***first*** node in our linked list whose value starts with an `"A"`, or `None` if no nodes in our list start with `"A"`. `hints[1]` does the same for nodes starting with `"B"`, `hints[2]` for `"C"`, etc.

Whenever we want to search for, or insert a new name into our list, we first check `hints` to see if a node starting with the same letter as our target exists. If it does, we use that element as the hint to our search/insert. Otherwise we use the unhinted search/insert. If the operation was an insert, we also update `hints` as needed.

For this problem, assume that there are roughly the same number of names starting with each letter of the alphabet.

a. **[2 points]** How does the sorted nature of the list affect our ability to use hinted searches?

b. **[2 points]** What is the asymptotic complexity of updating and retreiving hints from the `hints` array?

c. **[4 points]** What is the unqualified worst-case complexity of finding an element in our list when using a hint retreived from the `hints`? In this case you may assume the hint in `hints` was not `None`, so we will use the hinted version of search.

d. **[4 points]** If the answer is different from the answer you gave in Problem 1b, explain how the use of hints changes the complexity class of our search. If the answer is the same as the answer you gave in Problem 1b, then you claim the runtime of the two search methods only differs by a constant factor. What is this constant factor and why?

Problem 3 - Duplicate Elements

[12/30 points]

For PA2, your linked list handled duplicate elements by storing them all within a single node with a `count` member to keep track of how many elements with that value have been added to the list. This is in contrast to creating a new node for every element, even if multiple elements have the same value.

a. **[2 points]** How does the sorted nature of the list affect our ability to treat duplicate elements specially?

b. **[5 points]** If we use the `SortedLinkedList` from PA2 to store a list of student grades, where each grade is an integer ranging from 0 to 100, what is the unqualified worst-case runtime to find the node for a particular grade? Is this runtime aymptotically different if we create a new node for every element? Explain why or why not.

c. **[5 points]** If we use the `SortedLinkedList` from PA2 to store a list of student first names, where we can assume there are an infinte number of possible first names, what is the unqualified worst-case runtime to find the node for a particular name? Is this runtime aymptotically different if we create a new node for every element? Explain why or why not.