University of Virginia Computer Science
CS216: Program and Data Representation, Spring 2006 27 February 2006

[PDF version for printing]

Out: 22 February 2006

Exam 1

Due: Monday, 27 February, 11:01AM

Name: _____

Directions

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between receiving this exam and class Monday.

Open resrouces. You may use any books you want, lecture notes and slides, your notes, and problem sets. If you use anything other than the course books and notes, cite what you used. You may not use other people.

No Python. You may not run a Python interpreter between now and when you turn in this exam. If you need to use a Python interpreter for some other purpose before then, request permission first.

Answer well. Write your answers on this exam. You should not need more space than is provided to write good answers, but if you want more space you may attach extra sheets. If you do, make sure the answers are clearly marked.

This exam has 11 questions, and two optional non-credit questions. The questions are not necessarily in order of increasing difficulty, so if you get stuck on one question you should continue on to the next question. There is no time limit on this exam, but it should not take a well-prepared student more than an hour or two to complete.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler.

Order Notation

For each of the questions below, fill in the missing symbol with one of these choices:

- $=$ — the sets are equal
- $\subset$ — the left set is a strict subset (cannot be equal) of the right set
- $\supset$ — the left set is a strict superset (cannot be equal) of the right set
- $\subseteq$ — the left set is a subset (can be equal) of the right set
- $\supseteq$ — the left set is a superset (can be equal) of the right set
- $\neq$ — the sets are not equal, but there is no subset or superset relationship

You should select the strongest possible choice (for example, if two sets are equal and you select subset, that is a correct statement, but not worth full credit).

For each answer, provide a short justification of your answer. Your justification should follow from the definitions of the order notations.

1. (5 points) $O(n)$ _____ $\Theta(n^2)$
Justification:

2. (5) $\Theta(n)$ _____ $O(2n)$
Justification:

3. (5) $\varnothing$ _____ $O(n!) \cap \Omega(n^n)$
Justification:

4. (5) $O(1)$ _____ $\Theta(1)$
Justification:

Lists

5. (10) Complete the definition of the `ListNode` method `reverse`, that is called by the `LinkedList` method `reverse` to produce a reversed self list as its output (the same elements as in self, but in reverse order). For example,

```
l = LinkedList.LinkedList ().append(1).append(2).append(3)
r = l.reverse ()
```

should make `r` the list `[3, 2, 1]` and leave `l` as the list `[1, 2, 3]`.

The rest of the code is taken from the `LinkedList.py` implementation of an immutable list datatype from Problem Set 2.

Remeber that you are not allowed to use a Python interpreter for this exam. We are concerned with the correctness and clarity of your algorithm, not the details of Python syntax. For full credit, your code must be simple and straightforward. You should not need more than 6 lines.

```
class LinkedList:
    def __init__(self):
        self.__node = None

    def access (self, index):
        return self.__node.access (index)

    def append (self, e):
        res = LinkedList ()
        if self.__node == None:
            res.__node = ListNode(e)
        else:
            res.__node = self.__node.append (e)
        return res

    def reverse (self):
        res = LinkedList ()
        if self.__node == None:
            res.__node = None
        else:
            res.__node = self.__node.reverse ()
        return res

class ListNode:
    def __init__(self,info):
        self.__info = info
        self.__next = None

    def getNext (self):
        return self.__next

    def access (self, index):
        if index == 0:
            return self.__info
        else:
            return self.__next.access (index - 1)
```

(Continues on next page)

```
def append (self, value):
    res = ListNode (self.__info)
    last = res
    current = self
    while not current.__next == None:
        current = current.__next
        last.__next = ListNode (current.__info)
        last = last.__next

    last.__next = ListNode (value)
    return res

def reverse (self):
    head = ListNode (self.__info)
    if self.__next == None:
        return head
    else:
       # Answer question 5 by writing your code here:
```

6. (10) What is the asymptotic running time of your `reverse` implementation? Explain your answer convincingly, and be sure to define any variables you use and state any assumptions you make clearly.

Matching

For Problem Sets 1 and 2 (and possibly some future problem sets), I assigned students partners. The goal of the assignments is to maximize the total goodness value of all the assignments, where each pair has a goodness score calculated according to some function. For example, the goodness score used for assigning Problem Set 1 partners was:

```
def goodnessScore (s, t):
    if s == None or t == None: return -1

    # matching in same section preferred
    if records[s]['section'] == records[t]['section']:
        score += 100

    # better to match with different major
    if not records[s]['major'] == records[t]['major']:
        score += 20

    # better to match students in different years
    if not records[s]['year'] == records[t]['year']:
        score += 10

    # very bad to match with someone either partner listed in question 6
    # on the registration survey
    if records[s]['notpartners'].find (t) != -1:
        score -= 1000
    if records[t]['notpartners'].find (s) != -1:
        score -= 1000

    return score
```

One way to determine the partnerships is to use a greedy algorithm — it goes through the students in order, finding the best possible match for each student considering only the students who have not yet been matched:

```
def assignPartners(students):
   partners = { }
   for student in students:
      if partners.has_key (student):
         break # already matched with a partner

      partner = None
      for ppartner in students:
          # can't partner with yourself or already partnered student
          if not ppartner == student and not partners.has_key (ppartner):
              if goodnessScore (student, ppartner) \
                    > goodnessScore (student, partner):
                 partner = ppartner
      if partner == None:
          pass # No partner for student
      else:
          partners[student] = partner
          partners[partner] = student
   return partners
```

This is a simple algorithm, but it is not optimal. (In this context, an optimal algorithm would produce a set of partner pairings that maximizes the total goodness score. We count each partnership twice, from the perspective of both students. Whether or not those partnerships are ideal according to the goals of the course depends on having the correct `goodnessScore` function, which of course, is much tougher.)

7. (10) Prove the greedy partnering algorithm shown is not optimal by showing an input for which it would not produce the correct result.

8. (5) What is the asymptotic running time of `assignPartners`? Be sure to define any variables you use in your answer and state your assumptions about Python operations clearly.

Zulma Zyppy is upset with her problem set partners (actually, she is upset that she didn't get assigned any partner at all for both problem sets). She suggests replacing the greedy partner assignment algorithm with an optimal algorithm that trys all possible partner assignments to find the one with the best total goodness score.

She has implemented part of the code for an optimal algorithm below:

```
def assignPartners(students):
    best = None
    bestscore = 0

    for partners in allPossiblePartnerAssignments(students):
        score = 0
        for student in students:
            score += goodnessScore (student, partners[student])
        if score > bestscore:
            best = partners
            bestscore = score

    return best
```

Because of her problems with PS1 and PS2 partners, however, Zulma has asked you for help implementing `allPossiblePartnerAssignments`.

9. (10) Define the `allPossiblePartnerAssignments` procedure Zulma needs.

10. (10) Explain why Zuma's partner assignment algorithm would not run fast enough to be used to assign partners for PS4. (Note: a good answer would include an explanation of the running time of `assignPartners`. Assume you have a correct and optimally efficient `allPossiblePartnerAssignments` implementation regardless of your answer to question 9. You should be able to answer question 10 well, even if you could not answer question 9.)

11. (no expected points, bonus points possible) Suggest a better algorithm to use to assign partners for future problem sets. (Bonus points will be awarded both for better `assignPartners` algorithms for finding a goodness-maximizing partner assignment, and for good (even non-technical) suggestions about the `goodnessScore` function.)

These two questions are optional and worth no credit, but we appreciate your answers.

12. (no credit) Do you feel your performance on this exam will fairly reflect your understanding of the course material so far? If not, explain why.

13. (no credit) Do you have any comments about how the course is going so far or suggests for improving the remainder of the course? (If you prefer to answer this question anonymously, you may turn in a separate page with your answer on it and no name.)