Algorithms and Data Structures for Data Science

Lists and Asymptotic Efficiency Review

CS 277
Brad Solomon

February 8, 2023

Department of Computer Science
Lab_parsing Feedback

Average score: 92%

PL average time: 77 minutes

Class helpful to only half of students (as far as completing lab)

Lab generally improved most but not all confidence; some lost confidence

Many issues with the clarity of questions
Assignments are released on the website

Look on the website for examples and the full details of an assignment

I’ll remove the descriptions in the notebook — at least for now

I’ll put back in working examples in the notebooks
Exam 1

Exams will be proctored by the CBTF: https://cbtf.engr.illinois.edu/

(That link will have a link to Prairietest, where you can sign up for exam 1)

Reservations open on February 2nd @ 9 AM

You must take the exam sometime between 2/14 and 2/16!

See website for expected content:

https://courses.grainger.illinois.edu/cs277/sp2023/exams/
Learning Objectives

Review asymptotic efficiency

Compare list implementations using Big O
Big-O notation

\[ f(n) \text{ is } O(g(n)) \text{ iff } \exists c, k \text{ such that } f(n) \leq cg(n) \quad \forall n > k \]

1) \( cg(n) \) is an upper bound on \( f(n) \)

2) This is true for all input values larger than some arbitrary \( k \)
In-Class Exercise

What is the big O for the following functions?

\[
a(n) = n^4 + 50n + 10
\]

\[
b(n) = 500n \log n + 50n + \log(n)
\]

\[
c(n) = n^3 + 3n! + 12
\]

\[
d(n) = n^2 + n \log n
\]
Big-O Complexity Classes

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n^2)$
- $O(2^n)$
Identifying the Big O of an algorithm

1) Label the key factors that drive algorithm performance

2) Write out the worst-case performance for each step

3) Identify (or reduce to) the largest terms for each factor
def doStuff(inList1, inList2):
    c1 = 0
    for i in inList1:
        c1+=1

    c2 = 0
    for v1 in inList1:
        for v2 in inList2:
            c2+=1

    return c1, c2
def doStuff2(inList):
    ops = 0
    size = len(inList)
    while size > 0:
        size = int(size / 2)
        ops+=1
    return ops

def doStuff3(inList1, inList2):
    ops = 0
    for i in inList1:
        ops+= doStuff2(inList2)
    return ops
def convert_1D_to_2D(inList, rowSize):
    listLen = len(inList)
    numRows = math.ceil(listLen/rowSize)

    outList = []
    count = 0

    ops = 0
    for i in range(numRows):
        tempList = []

        for j in range(rowSize):
            if count >= listLen:
                tempList.append(-1)
            else:
                tempList.append(inList[count])
                ops+=1
                count+=1

        outList.append(tempList)

    print(ops)
    return outList
List Implementations

1. Linked List

head

2. Array List
List Implementation Big O

1. Linked List

2. Array List
List Implementation Big O

1. Linked List

__getItem__( )

head

C → S → 2 → 7 → 7 → None
List Implementation Big O

2. Array List

C S 2 7 7
List Implementation Big O

1. Linked List

head

2. Array List
List Implementation Big O

1. Linked List

head

\[ \textbf{\_find\_}( \ ) \]
List Implementation Big O

2. Array List

\[\text{find}(\ )\]
List Implementation Big O

1. Linked List

```
head
C    S    2    7    7
```

2. Array List

```
C    S    2    7    7
```
List Implementation Big O

1. Linked List

head

[Diagram showing a linked list with nodes containing 'c', 's', 2, 7, 7, and a 'None' at the end.]

**insertFront( )**
List Implementation Big O

2. Array List

insertFront(  )
List Implementation Big O

1. Linked List

head

```
| C | S | 2 | 7 | 7 | None |
```

2. Array List

```
| C | S | 2 | 7 | 7 |
```
List Implementation Big O

1. Linked List

$$\text{insertEnd}(\quad)$$

head

C  S  2  7  7  None
List Implementation Big O

2. Array List
Resize Strategy: x2 elements every resize
Resize Strategy: x2 elements every resize
List Implementation Big O

1. Linked List

head → C → S → 2 → 7 → 7 → None

2. Array List

```
C  S  2  7  7
```
List Implementation Big O

1. Linked List

head

insert(  ,  )
List Implementation Big O

2. Array List

| C | S | 2 | 7 | 7 |   |   |   |   |   |
List Implementation Big O

1. Linked List
   head

   ![Linked List Diagram]

2. Array List
   ![Array List Diagram]
List Implementation Big O

1. Linked List

```plaintext
head

C -> S -> 2 -> 7 -> 7 -> None

remove()
List Implementation Big O

2. Array List

| C | S | 2 | 7 | 7 |   |   |   |   |
## Array Implementation

<table>
<thead>
<tr>
<th></th>
<th>Singly Linked List</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look up given an input <strong>position</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search given an input <strong>value</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert/Remove at <strong>front</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Insert/Remove at <strong>arbitrary</strong> location</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
When would you use LinkedList vs Array?
Questions?