Lab 4 – Lists

Instructors: Lorenzo De Carli & Maan Khedr

Slides by Lorenzo De Carli

What we are going to focus on today

- Understanding implementation and performance of arrays and linked lists
- Refresh our complexity analysis skills

Math notation in markdown

- Several exercises ask you to discuss complexity analysis and provide the answer in markdown (.md) files
- It may be useful to know that markdown supports mathematical notation (based on LaTeX math syntax)
- Basic examples:

examples.md

```
1  Superscript: $O(n^2)$
2
3  Subscript: $a_1,a_2$
4
5  Sum: $\sum_{i=1}^{100} i = 5050$
```

Superscript: $O(n^2)$

Subscript: a_1, a_2

Sum: $\sum_{i=1}^{100} i = 5050$

Exercise #1: binary search on linked lists [0.5 pt]

- Did you know that it is possible to implement binary search on a list?
- This is somewhat counter-intuitive (why?)
- In this exercise we'll look into how to do it

Exercise #1 /2

- Look at discussions of this operation, for example at:
  - https://www.geeksforgeeks.org/binary-search-on-singly-linked-list/
  - https://www.prepbytes.com/blog/linked-list/binary-search-on-linked-list/

1. Implement a linked list (storing integers) as a Python class [0.1 pts]
2. Add a binary_search(num) method implementing the methods above [0.1 pts]
3. Implement an array class (storing integers) also implementing binary search (you can just use a Python array internally) [0.1 pts]

Exercise #1 /3

4. You already know that complexity of binary search for array is *O(logn)*. What is the complexity of binary search for linked lists? (explain in your own words, do not just copy/paste from the sites above) [0.1 pts]
5. Measure average-case performance of binary search in linked list and array on inputs of size [1000, 2000, 4000, 8000]; plot performance for both in the same figure, and interpolate with appropriate functions [0.1 pts]

Exercise 1 - What to deliver

- Submit a file named ex1.py with your implementation of both algorithms and your timing code
- The file should also contain the answer to question 4 as comment in the code

Exercise 2 – dynamic arrays [0.5 pts]

- In class we discussed the inner functioning of dynamic array

1. Explain the difference between an array size and capacity [0.1 pts]
2. What happen when an array needs to grow beyond its current capacity? Explain and produce a diagram showing the memory layout before and after expansion
    1. First, consider the case where there is space in memory after the end of the array [0.1 pts]
    2. Then, consider the case where the memory after the end of the array is occupied by another variable. What happens in that case? [0.2 pts]
3. Discuss one or more techniques real-world array implementations use to amortize the cost of array expansion [0.1 pts]

Exercise 2 - What to deliver

- Submit a file named ex2.pdf with answers to questions 1, 2 and 3
- You can use any editor of your choice – Word, Google Docs, Overleaf/LaTex, etc. as long as the output is PDF

Exercise 3 – more dynamic arrays [0.5 pts]

- In this exercise you will build understanding of how Python concretely manages and grows dynamic arrays
- First, look at the actual C implementation of Python lists (they are called “lists”, but really are dynamic arrays) (file `lists.c` on D2L)

Exercise 3 /2

1. Identify and explain the strategy used to grow arrays when full, with references to specific lines of code in the file above. What is the growth factor? [0.1 pts]
2. Produce test code that checks the correctness of your inference for lists up to 64 elements. Specifically, write a Python loop that grows a list from 0 to 63 integers, printing out a message when the underlying capacity of the list changes. [0.1 pts]
   - Recall that the capacity of the list is the number of elements that can be stored (`sys.getsizeof` is your friend here, but remember that it returns sizes expressed in bytes!)

Exercise 3 /3

- Consider the largest array size S identified in question 2 which causes the array to be expanded when an element is added.

3. Measure the time it takes to grow the array from size S to $S+1$ (i.e., start from an array of size S and append one element). Repeat the measure 1,000 times [0.1 pts]
4. Measure the time it takes to grow the array from size $S-1$ to S. Repeat the measure 1,000 times [0.1 pts]
5. Plot the distribution of both measurements (you can use hist or similar). Do you see any difference? Why? [0.1 pts]

Exercise 3 - What to deliver

- Submit a file named ex3.py with answers to questions 2, 3, 4, 5
- The file should also contain the answer to question 1 and discussion requested by question 5 as comments in the code

Exercise 4 – Complexity analysis refresh [0.5 pts]

- Let's practice complexity analysis
- First, Consider the following code:

```
def processdata(li):
  for i in range(len(li)):
    if li[i] > 5:
      for j in range(len(li)):
        li[i] *= 2
```

Exercise 4 /2

1. State and justify what is the best, worst and average case complexity for the code in the previous slide [0.1 pts]
2. Are average, best, and worst case complexity the same? If not, produce a modified version of the code above for which average, best, and worst case complexity are equivalent. [0.1 pts]

Exercise 4 /3

- Now, consider the tasks of **searching in a sorted array**

3. Provide the code for an inefficient implementation and an efficient implementation. [0.1 pts]
4. State the worst-case complexity of each. [0.1 pts]
5. Provide the code for an experiment that demonstrates the difference. [0.1 pts] The experiment should:
    1. Time the execution of both implementations on realistic, large inputs (1000 elements or above)
    2. Plot the distribution of measured values across multiple measurements (>= 100 measurements per task)

Exercise 4 - What to deliver

- Submit a file named ex4.1.py with answers to questions 1, 2 (provide answers as comments, and code as necessary)
- Submit a file named ex4.2.py with answers to questions 3, 4, and 5

Exercise 5 – Reflecting on measurements [1 pt]

- In class, we have seen two different ways to perform multiple measures with timeit:
  - The first uses the number parameters, as in:
    ```
    elapsed_time = timeit.timeit(lambda: fibonacci(1000), number=100)
    ```
  - The second uses the repeat function, as in:
    ```
    times = timeit.repeat(lambda: fibonacci(1000), repeat =5, number=10)
    ```

Exercise 5 /2

- These approaches are designed to deal with different types of measurement noise. Think about what happens when we try to time a program, and which types of issues may result in an incorrect measurement. Reflect on how the two approaches (`timeit` and `repeat`) attempt to address these issues. Discuss when it is appropriate to use one or the other. [0.5 pts]
- Typically, the output of `timeit` is post-processed to compute some sort of aggregate statistics. Let's only consider three: *average*, *min*, and *max*. Which one is the appropriate statistic to apply to the output of `timeit.timeit()`? What is the appropriate statistics to apply to the output of `timeit.repeat()`? Discuss why. [0.5 pts]

Exercise 5 - What to deliver

- Submit a file named ex5.md with answers to questions 1 and 2

Exercise 6 - Back to arrays and lists [1 pt]

- In the lecture recordings, we discussed some of the main differences between arrays (or lists in python) and Linked Lists.

1. Compare advantages and disadvantages of arrays vs linked list (complexity of task completion) [0.1 pts]
2. For arrays, we are interested in implementing a replace function that acts as a deletion followed by insertion. How can this function be implemented to minimize the impact of each of the standalone tasks? [0.1 pts]

Exercise 6 /2

3. Assuming you are tasked to implement a **doubly linked list** with a sort function, given the list of sort functions below, state the feasibility of using each one of them and elaborate why is it possible or not to use them. [0.4 pts]
    1. Insertion sort
    2. Merge sort
4. Also show the expected complexity for each and how it differs from applying it to a regular array [0.4 pts]

Exercise 6 - What to deliver

- Submit a file named ex6.md with answers to questions 1, 2, 3 and 4

Exercise 7 – Reversing list [1 pt]

- Consider a list implementation based on a **singly-linked list** with **no tail pointer**
- The list offers the following methods:
  - `insert_head(node)`: insert node at head
  - `insert_tail(node)`: insert node at tail
  - `get_size()`: return number of elements in list
  - `get_element_at_position(pos)`: return element at position pos

Exercise 7 /2

- Consider the following implementation of a function to reverse the list:

```
def reverse(self):
  newhead = None
  prevNode = None
  for i in range(self.get_size()-1, -1, -1):
    currNode = self.get_element_at_pos(i)
    currNewNode = Node(currNode.data)
    if newhead is None:
      newhead = currNewNode
    else:
      prevNode.next = currNewNode
    prevNode = currNewNode
  self.head = newhead
```

Exercise 7 /3

1. Give an expression for the time complexity of the `reverse()` implementation. Explain how you reached the conclusion describing your step-by-step reasoning. [0.3 pts]
2. Design an optimized implementation of the same function with better performance. Discuss which changes you made and how they should be expected to result in a better function [0.3 pts]
3. Time both methods (the given one and yours) on list, of 1000, 2000, 3000, 4000 elements, for 100 times. [0.2 pts]
4. Plot the measurement results in a X-Y plots for both methods. [0.2 pts]

Exercise 7 - What to deliver

- Submit a file named ex7.md with answers to questions 1, and 2
- Submit a file named ex7.py with answers to question 2 (coding part), 3 and 4

How to submit

- Upload a zip file to the “Lab 4” dropbox on D2L, containing the required content for every exercise

Grading rubric

- You get **3 pts** for uploading a **partial solution** by **end of lab**
  - **Must not be an empty file or irrelevant material**
- Then, you'll have until **11:59PM of the day before the next lab** to upload the **complete solution**. That will be graded as follows:
  - Exercises 1-4: 0.5 pts each
  - Exercises 5-7: 1 pt each
  - **Video: 2 pts**
  - **Can upload the complete solution to the same dropbox**

That’s all folks!