CSC110 Lecture 24: Analyzing Built-In Data Type Operations

Print this handout

You will find the following formula helpful:

$$\forall n \in \mathbb{N}, \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

Exercise 1: Running time of list operations

Each of the following Python functions takes a list as input. Analyse each one's running time in terms of n the size of its input.

1.

```python
def f1(nums: list[int]) -> None:
    list.insert(nums, 0, 10000)
```

Let n be the length of the list input to f1.

The call to list.insert is inserting at the beginning of the list. This causes each of the n elements to shift by one place in memory. $\therefore RT_{f1}(n) = n \in \Theta(n)$

(or use: list.insert(lst, i, ...) is $\Theta(n-i)$ and here "i" is 0.)

2.

```python
def f2(nums: list[int]) -> None:
    for i in range(0, 100):
        list.append(nums, 10000)
```

Let n be the length of the list input to f2.

The for loop iterates 100 times.

Each iteration appends to the list, a $\Theta(1)$ operation that we will count as 1 step.

$$\therefore RT_{f2}(n) = 100 \cdot 1$$
$$= 100$$
$$\in \Theta(1)$$

3.

```python
def f3(nums: list[int]) -> None:
    for i in range(0, 100):
        list.insert(nums, 0, 10000)
```

i goes from 0 to 99

step count increases by 1

Note: the length of nums changes at each iteration, and so the running time of list.insert does a well!

each time

rough analysis:

$$RT_{f3}(n) = \sum_{i=0}^{99} \overbrace{(n+i)}^{\text{the length of the list when next value inserted}}$$

99

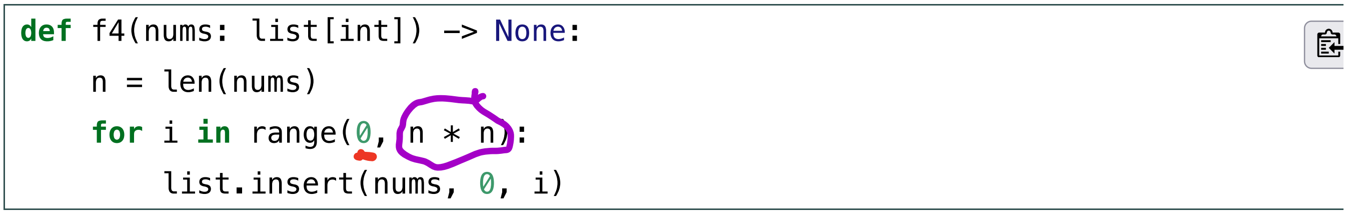
a constant

$$= 100n + \boxed{\sum_{i=0} i}$$

$$= 100n + \frac{99 \cdot 100}{2}$$

$$\in \Theta(n)$$

4.



```
def f4(nums: list[int]) -> None:
    n = len(nums)
    for i in range(0, n * n):
        list.insert(nums, 0, i)
```

rough analysis:

$$RT_{f4}(n) = 1 + \sum_{i=0}^{\boxed{n^2-1}} \boxed{(n+i)}$$

length of list when ith insert made

$$= 1 + n^2 \cdot n + \sum_{i=0}^{n^2-1} i$$

$$= 1 + n^3 + \frac{(n^2-1)(n^2)}{2}$$

$$\in \Theta(n^4)$$

Exercise 2: Running-time analysis with multiple parameters

Each of the following functions takes more than one list as input. Analyse their running time in terms of the size of their inputs; do not make any assumptions about the relationships between their sizes.

5.

```python
def f5(nums1: list[int], nums2: list[int]) -> None:
    for num in nums2:
        list.append(nums1, num)
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

Rough analysis:

$$RT_{f5}(n_1, n_2) = \sum_{i=0}^{n_2-1} 1$$

$$= n_2 \cdot 1$$

$$\in \Theta(n_2)$$

6.

```python
def f6(nums1: list[int], nums2: list[int]) -> None:
    for num in nums2:
        list.insert(nums1, 0, num)
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

Rough analysis:

$$RT_{f6}(n_1, n_2) = \sum_{i=0}^{n_2-1} (n_1 + i)$$

$$= n_1 \cdot n_2 + \sum_{i=0}^{n_2-1} i$$

$$= n_1 n_2 + \frac{(n_2-1)(n_2)}{2}$$

$$= \frac{n_2}{2}(2n_1 + n_2 - 1)$$

$$\in \Theta(n_1 n_2 + n_2^2)$$

Can simplify

$$\Theta(n_2(n_1+n_2))$$

or $\Theta(n_2 \max(n_1, n_2))$

Exercise 3: Sets, dictionaries, and data classes

Analyse the running time of each of the following functions.

7.

```python
def f7(nums: set[int]) -> bool:
    return 1 in nums or 2 in nums
```

Let n be the length of the set input to f7.

Since searching in a set is $\Theta(1)$,

$RT_{f7}(n) \in \Theta(1)$.

8.

```
def f8(num_map: dict[int, int]) -> None:
    for num in num_map:
        num_map[num] = num_map[num] + 1
```

Let n be the length of the dict input to f8.

There are n iterations of the for loop body. Since dict indexing and assignment are $\Theta(1)$ operations

$$RT_{f8}(n) = n \cdot 1$$
$$= n$$
$$\in \Theta(n)$$

9.

```
def f9(grades: dict[str, list[int]], new_grades: dict[str, int]):
    for course in new_grades:
        if course in grades:
            list.append(grades[course], new_grades[course])
        else:
            grades[course] = [new_grades[course]]
```

Let n be the length of the input new_grades to f9.

The for loop iterates n times.

Since key search in a dict, list append and assignment to a dict are all $\Theta(1)$ operations,

$$
\begin{aligned}
RT_{f9}(n) &= n \cdot 1 \\
&= n \\
&\in \Theta(n)
\end{aligned}
$$

10.

```python
from dataclasses import dataclass
import math


@dataclass
class Person:
    """Docstring omitted"""
    name: str
    age: int

def f10(people: list[Person]) -> int:
    """Precondition: people != []"""
    max_age_so_far = -math.inf

    for person in people:
        if person.age > max_age_so_far:
            max_age_so_far = person.age

    return max_age_so_far
```

Let n be the length of the input list to f10.

Since dataclass attribute lookup and assignment are $\Theta(1)$ operations,

$$\begin{aligned} RT_{f10}(n) &= 1 + n \cdot 1 + 1 \\ &= n + 2 \\ &\in \Theta(n). \end{aligned}$$

Additional exercises

Analyse the running time of each of the following functions.

1.

```python
def extra1(nums: list[int]) -> None:
    for i in range(0, len(nums)):
        nums[i] = 0
```

2.

```python
def extra2(nums: list[int]) -> None:
    for i in range(0, len(nums)):
        list.pop(nums)
```

3.

```python
def extra3(nums: list[int]) -> None:
    for i in range(0, len(nums)):
        list.pop(nums, 0)
```

Note: the length of `nums` changes at each iteration, and so the running time of `list.pop` does as well!

4.

```python
def extra4(nums1: list[int], nums2: list[int]) -> None:
    for i in range(0, len(nums1)):
        for j in range(0, len(nums2)):
            nums1[i] = nums1[i] + nums2[j]
```

(Let n_1 be the size of `nums1` and n_2 be the size of `nums2`.)

5.

```python
def extra5(nested_nums: list[list[int]]) -> None:
    for i in range(0, len(nested_nums)):
        if i == 0:
            for j in range(0, len(nested_nums[0])):
                nested_nums[i][j] = 0
        else:
            nested_nums[i][0] = 0
```

(Let n be the length of `nested_nums`, and assume every inner list has length m.)

6.

```python
def extra6(nums: set[int]) -> list[int]:
    new_nums = []

    for num in nums:
        list.insert(new_nums, 0, num ** 2)

    return new_nums
```

7.

```python
def extra7(nums: list[int]) -> dict[int, int]:
    counts_so_far = {}

    for num in nums:
        if num in counts_so_far:
            counts_so_far[num] = counts_so_far[num] + 1
        else:
            counts_so_far[num] = 1

    return counts_so_far
```