CSC110 Lecture 24: Analyzing Built-In Data Type Operations

Hisbaan Noorani

November 17, 2020

Contents

1 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) # n steps
```

$RT_{f_1} \in \Theta(n)$

2.
```python
def f2(nums: List[int]) -> None:
    for i in range(0, 100):        # 100 steps
        list.append(nums, 10000)   # 1 step
```

$RT_{f_2} \in \Theta(100)$

3.
```python
def f3(nums: List[int]) -> None:
    for i in range(0, 100):            # 100 steps
        list.insert(nums, 0, 10000)    # n^2 steps
```

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

i	shifts
0	$n+0$
1	$n+1$
2	$n+2$
$\dots$	$\dots$
99	$n+99$

$100n + \dfrac{99(99+1)}{2} \in \Theta(n)$

$RT_{f_3} \in \Theta(n)$

4.
```python
def f4(nums: List[int]) -> None:
    n = len(nums)                   # 1 step
    for i in range(0, n * n):       # n^2 steps
        list.insert(nums, 0, i)     # n^3 steps
```

i	shifts
0	$n+0$
1	$n+1$
2	$n+2$
$\dots$	$\dots$
n^2-1	$n+n^2-1$

$$(n^2)(n) + \frac{(n^2-1)(n^2-1+1)}{2} \in \Theta(n^4)$$

$$RT_{f_4} \in \Theta(n^4)$$

2 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.

1.
```python
def f5(nums1: List[int], nums2: List[int]) -> None:
    for num in nums2:               # n_2 steps
        list.append(nums1, num)     # 1 step
```

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

$$RT_{f_5} \in \Theta(n_2)$$

2.
```python
def f6(nums1: List[int], nums2: List[int]) -> None:
    for num in nums2:                   # n_2 steps
        list.insert(nums1, 0, num)      # n_1 * n_2 steps
```

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

$$n_2 \cdot n_1 \cdot \frac{(n_2)(n_2+1)}{2} \in \Theta(n_1 \cdot n_2 + (n_2)^2)$$

$$RT_{f_6} \in \Theta(n_1 \cdot n_2 + (n_2)^2)$$

3.
```python
def f7(nested_nums: List[List[int]]) -> None:
    for nums in nested_nums:            # n steps
        list.insert(nums, 0, 10000)     # m steps
```

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

$$RT_{f_7} \in \Theta(n \cdot m)$$

3 Exercise 3: Sets, dictionaries, and data classes

Analyse the running time of each of the following functions.

11.
```python
def f8(nums: Set[int]) -> bool:
    return 1 in nums or 2 in nums
```

$RT_{f_8} \in \Theta(1)$

12.
```python
def f9(num_map: Dict[int, int]) -> None:
    for num in num_map:
        num_map[num] = num_map[num] + 1
```

Let $n =$ `len(num_map)`

$RT_{f_8} \in \Theta(n)$

13.
```python
def f10(grades: Dict[str, List[int]], new_grades: Dict[str, int]):
    for course in new_grades: # n^2 steps
        if course in grades:  # 1 step (for the entire bock)
            list.append(grades[course], new_grades[course])
        else:
            grades[course] = [new_grades[course]]
```

Let $n_1 =$ `len(grades)`

Let $n_2 =$ `len(new_grades)`

$RT_{f_10} \in \Theta(n_2)$

14.
```python
def f11(people: List[Person]) -> int:
    """Precondition: people != []"""
    max_age_so_far = -math.inf            # 1 step

    for person in people:                 # n steps
        if person.age > max_age_so_far:   # 1 step (for the entire block)
            max_age_so_far = person.age

    return max_age_so_far                 # 1 step
```

(Assume the `math` module has been imported, and we've defined a `Person` data class with four attributes, including `age`.)

Let $n =$ `len(people)`

$RT_{f_11} \in \Theta(n)$

4 Additional exercises

Analyse the running time of each of the following functions.

11.
```python
def extra1(nums: List[int]) -> None:
    for i in range(0, len(nums)):  # n iterations
        nums[i] = 0                # 1 step
```

Let n be the length of `nums`.

$RT_{e_1} \in \Theta(n)$

12.

```python
def extra2(nums: List[int]) -> None:
    for i in range(0, len(nums)):  # n iterations
        list.pop(nums)             # 1 step
```

Let n be the length of `nums`.

$RT_{e_2} \in \Theta(n)$

13.

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

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

Let n be the length of `nums`.

14.

```python
def extra4(nums1: List[int], nums2: List[int]) -> None:
    for i in range(0, len(nums1)):              # n_1 iterations
        for j in range(0, len(nums2)):          # n_2 iterations
            nums1[i] = nums1[i] + nums2[j]      # 2 steps
```

Let n_1 be the size of `nums1` and n_2 be the size of `nums2`

$RT_{e_4} \in \Theta(n_1 \cdot n_2)$

15.

```python
def extra5(nested_nums: List[List[int]]) -> None:
    for i in range(0, len(nested_nums)):                # n steps
        if i == 0:
            for j in range(0, len(nested_nums[0])):     # m steps
                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.

$RT_{e_5} \in \Theta(n + m)$

16.

```python
def extra6(nums: Set[int]) -> List[int]:
    new_nums = []                               # 1 step

    for num in nums:                            # n iterations
        list.insert(new_nums, 0, num ** 2)      # n steps

    return new_nums                             # 1 step
```

Let n be the length of `nums`.

$RT_{e_6} \in \Theta(n^2)$

17.

```python
def extra7(nums: List[int]) -> Dict[int, int]:
    counts_so_far = {}                                      # 1 step

    for num in nums:                                        # n iterations
        if num in counts_so_far:                            # 1 step (in for dict)
            counts_so_far[num] = counts_so_far[num] + 1     # 2 steps
        else:
            counts_so_far[num] = 1                          # 1 step

    return counts_so_far                                    # 1 step
```

Let n be the length of `nums`.

$RT_{e_8} \in \Theta(n)$