CSC111 Lecture 17: Iterative Sorting Algorithms, Part 1

Hisbaan Noorani

March 15, 2021

Contents

1 Exercise 1: Implementing selection sort

Here is the skeleton of a selection sort algorithm we developed in lecture:

```python
def selection_sort(lst: list) -> None:
    """Sort the given list using the selection sort algorithm.

    Note that this is a *mutating* function.

    >>> lst = [3, 7, 2, 5]
    >>> selection_sort(lst)
    >>> lst
    [2, 3, 5, 7]
    """
    for i in range(0, len(lst)):
        # Loop invariants
        #   - lst[:i] is sorted
        #   - if i > 0, lst[i - 1] is less than all items in lst[i:]

        # Find the index of the smallest item in lst[i:] and swap that
        # item with the item at index i.
        index_of_smallest = _min_index(lst, i)
        lst[index_of_smallest], lst[i] = lst[i], lst[index_of_smallest]


def _min_index(lst: list, i: int) -> int:
    """Return the index of the smallest item in lst[i:].

    In the case of ties, return the smaller index (i.e., the index that appears first).
```

```

    Preconditions:
        - 0 <= i <= len(lst) - 1

    >>> _min_index([2, 7, 3, 5], 1)
    2
    """
    min_index_so_far = i
    for x in range(i + 1, len(lst)):
        if lst[x] < lst[min_index_so_far]:
            min_index_so_far = x

    return min_index_so_far
```

Complete this implementation by implementing the helper function `_min_index`. Hint: this is similar to one of the functions you implemented on this week's prep!

2 Exercise 2: Running-time analysis

1. Analyse the running time of the helper function `_min_index` in terms of n, the length of the input `lst`, and/or i, the second argument.

   We will assume that the smallest item is at the end of the list for a worst case running time analysis.

```
def _min_index(lst: list, i: int) -> int:
    min_index_so_far = i                      # 1 step
    for x in range(i + 1, len(lst)):          # iterates n - i - 1 times
        if lst[x] < lst[min_index_so_far]:    # 1 step for whole if blcok
            min_index_so_far = x

    return min_index_so_far                   # 1 step
```

   Therefore an upper bound for the running time is $2 + n - i - 1 = 1 + n - i \in \mathcal{O}(n - i)$.

   Next, we will can find an input family that proves a lower bound, but we will ommit it in this example.

   Therefore, since we have found both an upper and lower bound that match, $RT_{\texttt{\_min\_index}} \in \Theta(n - i)$.

2. Analyse the running time of `selection_sort`.

   We will assume that the list to be sorted is in reverse order, thus slection sort will take as long as possible.

```
def selection_sort(lst: list) -> None:
    for i in range(0, len(lst)):                 # n steps
        index_of_smallest = _min_index(lst, i)   # n - i steps
        lst[index_of_smallest], lst[i] =\        # 1 step
            lst[i], lst[index_of_smallest]
```

The loop runs n times for $i = 0, 1, \ldots, n-1$

The iteration i takes $n - i$ steps (because of `_min_index(lst, i)`) plus one step for constant time operations.

Therefore $RT_{\texttt{selection\_sort}} = \sum_{i=0}^{n-1} (n - i + 1) \in \Theta(n^2)$.

3 Additional exercises

1. Translate the two loop invariants in `selection_sort` into Python assert statements. You can use `is_sorted=/=is_sorted_sublist` from this week's prep.

   (One version is included in the Course Notes, but it's a good exercise for you to try it yourself without looking there first!)