FIT1045/FIT1053 Algorithmic Problem Solving – Tutorial 4.

Objectives

After this tutorial, you should be able to:

- Sort a list using selection sort.
- Sort a list using insertion sort.
- Explain the important components of selection and insertion sort.
- Describe how functions interact.

Prepared Question

Your phone (likely) knows you who your most-used contacts are so they can be displayed on the speed-dial page. Here are two different ways we could implement this functionality in Python.

```
def speed_dial_v1(contact_list):
    sd_name, sd_number, sd_freq = contact_list.pop()
    while len(contact_list) > 0:
        name, number, freq = contact_list.pop()
        if freq > sd_freq:
            sd_name, sd_number, sd_freq = name, number, freq
    return (sd_name, sd_number)

def speed_dial_v2(contact_list):
    sd_name, sd_number, sd_freq = contact_list[0]
    for contact in contact_list:
        name, number, freq = contact
        if freq > sd_freq:
            sd_name, sd_number, sd_freq = name, number, freq
    return (sd_name, sd_number)
```

1. Complete a code trace on these function calls, and record the values of `x` and `y`:

```
list1 = [('Buffy', '04#', 5),('Blade', '04#', 2),('Abe', '03#', 4)]
x = speed_dial_v1(list1)

list2 = [('Tiffany', '04#', 5),('Hermione', '07#', 2),('Morgan', '00#', 4)]
y = speed_dial_v2(list2)
```

2. Which implementation do you think is better? Justify your answer.

You must attempt both parts to receive the mark.

Workbook

§5 Code traces and §§12.1, 2 Sorting.

While your tutors mark the prepared question, attempt the above Workbook sections. Your tutors will go through some of these questions with the class.

Warm-up

Do not solve these questions by running in Python.

Multiple assignment: What does `x` yield after executing the code block?

- `x, y, z = 'abc'`
- `w, x = ['horse', 'donkey']`

Mutability: What has changed after executing each function call?

- ```
  def remove_letters(lst):
      count = 0
      for i in range(len(lst)):
          if lst[i].isalpha():
              count += 1
              lst[i] = None
      return count

  my_list = ['3', 'a', '5', '2', 'b', 'd']
  num_letters = remove_letters(my_list)
  ```
- ```
  def shuffle_min(lst):
      min = lst[0]
      i = 1
      while i < len(lst):
          if lst[i] < min:
              lst[i], lst[0] = lst[0], lst[i]
              min = lst[0]
          i += 1
      return min

  my_list = [3, 6, 2, 9, 1, 4]
  minimum = shuffle_min(my_list)
  ```

Lists: What does `x` yield at the end of the code block?

- ```
  my_list = ['purple', 'blue', 'grey']
  x = my_list
  my_list.append('red')
  ```
- ```
  x = ['breakfast', 'lunch', 'dinner']
  hobbit_meals = x
  hobbit_meals.insert(1, 'elevenses')
  ```

Deep Copying: Draw a pointer/reference diagram of the *Frames* and *Objects* when the following code is executed, and answer the following:

- Is `'t2'` a deep copy of `'t1'` after execution?
- Is `'t2'` a shallow copy of `'t1'` after execution?

```
from copy import deepcopy

t1 = [['a', 'b'],
              ['c', 'd']]

t2 = deepcopy(t1)
t2[1] = t1[1]
```

Selection sort: What is the state of the list after the *first* swap that occurs when applying selection sort? (If you feel confident, practice sorting the list using selection sort.)

- `[3,6,1,8,2]`
- `[2,6,3,1,8]`

Insertion sort: What is the state of the list after the *first* swap that occurs when applying insertion sort? (If you feel confident, practice sorting the list using insertion sort.)

- `[3,6,1,8,2]`
- `[2,6,3,1,8]`

Introduction to Sorting

Selection Sort

Selection Sort has the following steps:

1. Set a pointer, k, at the first item in the list.
2. Find the location, m, of the smallest item in the list from k onwards.
3. Swap the item at k with the item at m.
4. Move k up by one.
5. Repeat steps 2 to 4 while k is within the list.

In your group write some lists of numbers. For each list, apply selection sort, ensuring that you track the state of the list after each occurence of step 4. Is it possible to change step 5 so that the algorithm can always end earlier? What happens if step 2 found the minimum item in the whole list? Why is a return statement unnecessary?

Choose *one* step from selection sort, and translate the instructions into Python. If this is too easy, then do more steps.[1]

Insertion Sort

Insertion Sort has the following steps:

1. Set a pointer, k, at the first item in the list.
2. Set a pointer, j equal to k.
3. If the item to the left of j is bigger than the item at j, swap the items and move j down by one.
4. Repeat step 3 until step 3 results in no swap.
5. Move k up by one.
6. Repeat steps 2 to 5 while k is within the list.

In your group write some lists of numbers. For each list, apply insertion sort, ensuring that you track the state of the list after each occurence of step 5. Is it possible to change step 1 to remove an unneccesary check? What kind of list would mean that the swap in step 3 never occurs? What kind of list would mean that the swap in step 3 occurs the maximum possible number of times.

Choose *one* step from insertion sort, and translate the instructions into Python. If this is too easy, then do more steps.

You are the Function

You have the following set of functions:

```
def sum(lst):
    total_sum = 0
    for elem in lst:
        total_sum += elem

def list_names(lst):
    lst = []
    for person in group: #this is NOT Python code
        x = person.middle_name()[:8]
        if person.middle_name() == None:
            x = person.favourite_animal()[:8]
        lst += [x]
```

[1] Keep in mind that the Python function `min` does not return the location of the minimum item.

```
def get_letter(string):
    count = 0
    for letter in string:
        count += 1
    count //= 2
    i = 0
    while count >= 0:
        i += 1
        count -= 1
    return get_letter_position(string[i])

def get_letter_position(letter):
    count = 1
    alphabet = '!abcdefghijklmnopqrstuvwxyz'
    while count <= 26:
        if alphabet[count] == letter:
            return count
        count += 1
    return 0

def main():
    num_list = []
    names = list_names()
    for name in list_names():
        num_list += [get_letter(name)]
    return sum(num_list)
```

Each member of your group should take a function. Your tutor will call the `main` function. If your tutor is unavailable, whoever takes `main` should decide when to begin executing. While you are playing at being a function, you may only interact with the rest of your group by taking input and giving output. (This includes `list_names` who must ask each person in the group for the necessary information once they are called.)

Minimalist Sorting

The University of Michigan and IBM are locked in a battle to see who can create the smallest computer. In 2018, both independently created a computer smaller than a grain of salt.[2]

Small computers present their own unique challenges when it comes to implementing algorithms. Both of the sorting algorithms so far seen not only require enough memory for the given list, but also require that there be enough memory to store additional variables.

In your group, design a sorting algorithm that requires exactly one variable in addition to the input list. You may assume that you have access to the input list's length, and that you can swap two items in the list. Once you have designed an algorithm, try implementing it in Python.

Shellsort (FIT1053)

Shellsort[3] (or Shell's sort) is a sorting algorithm that works by applying insertion sort to sublists, where the sublists are created by taking elements at a fixed gap in the list, and progressively reducing the gap. The gaps are predetermined.

For instance sorting a list with 12 elements, with gaps 5, 3, and 1 would result in the following steps:

- insertion sort would be run on lists $[a_1, a_6, a_{11}]$, $[a_2, a_7, a_{12}]$, $[a_3, a_8]$, $[a_4, a_9]$, and $[a_5, a_{10}]$
- insertion sort would be run on lists $[a_1, a_4, a_7, a_{10}]$, $[a_2, a_5, a_8, a_{11}]$, and $[a_3, a_6, a_9, a_{12}]$
- insertion sort would be run on the list as a whole.

Do you think Shellsort is an improvement on insertion sort? In your group discuss the pros and cons of Shellsort, and try implementing Shellsort in Python.

[2] Read about it here: `https://www.digitaltrends.com/computing/worlds-smallest-computer-dwarfed-by-rice-grain/`.
[3] Not examinable.