Texas A&M University Kingsville
Department of EECS
CSEN 5303 Foundations of Computer Science
Project 3 Sorting

Mengxiang Jiang
Professor Habib Ammari

October 31, 2022

Contents

Chapter 1

Introduction

The problem given is to sort an array containing the characters 'A', 'M', and 'T', such that all the $T's$ appear first, followed by all the $A's$, and lastly all the $M's$. Special constraints are in place to make this sort harder:

> **-Constraint 1:** Each letter ('A', 'M', or 'T') is evaluated **only once**.
> **-Constraint 2:** The function $SWAP(TAM, i, j)$ is **used only** when it is **necessary**.
> **-Constraint 3: No extra space** can be used by the algorithm $Sort\_TAM$. In other words, **only** the array TAMUK can be used to sort the 'A', 'M', or 'T'.
> **-Constraint 4:** You **cannot count** the number of each letter 'A', 'M', or 'T'.

The main algorithm used is a type of insertion sort, and the programming language used to implement it is Python.

Chapter 2

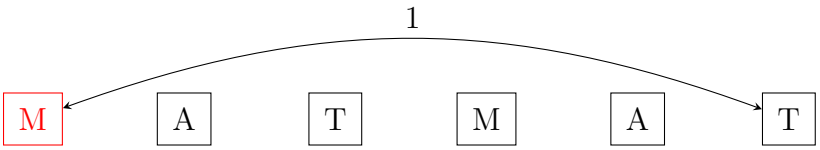
Design

First I will show the algorithm in pseudocode and then give a short example of how it works. Here is the pseudocode:

```
procedure Sort_TAM(TAMUK: arrayChar);
    var
        last_t, first_m, i: integer;
    begin
        last_t := -1;
        first_m := n;
        i := 0;
        while (i < first_m) do
            if TAMUK[i] = 'M' then
                while (TAMUK[first_m - 1] = 'M') do
                    first_m := first_m - 1;
                if i < first_m - 1 then
                    SWAP(TAMUK, i, first_m - 1);
                    first_m := first_m - 1;
            if TAMUK[i] = 'T' then
                if i > last_t + 1 then
                    SWAP(TAMUK, i, last_t + 1);
                last_t := last_t + 1;
            i := i + 1;
    end;
```

Suppose the given character array, $TAMUK$, is given as below:

We initialize our variables: $last\_t = -1$ and $first\_m = 6$



We evaluate $TAMUK[0]$: $last\_t = 0$ and $first\_m = 5$

We evaluate $TAMUK[1]$: $last\_t = 0$ and $first\_m = 5$

We evaluate $TAMUK[2]$: $last\_t = 1$ and $first\_m = 5$

We evaluate $TAMUK[3]$: $last\_t = 1$ and $first\_m = 4$

Since $first\_m = 4$, we don't need to evaluate $TAMUK[4]$ and can stop, so this is the final sorted array.

This algorithm is correct since it is insertion sort with an improvement. Rather than doing a maximum of $O(n)$ swaps in order to place the current element in the intermediate sorted list, it is able to place it in a maximum of 2 swaps. Thus, the worst case run time is $O(n)$ rather than $O(n^2)$ for standard insertion sort. The reason only 2 swaps maximum is needed is that we keep track of the position of the last known letter T and the position of the first known letter M, and only need to swap to these two locations (with their respective offsets) for the current element.

Chapter 3

Code

```
"""
class for sorting an array of T's, A's, and M's
"""

class SortTAM:
    def __init__(self, tam):
        self.tamuk = [c for c in tam]
        # if the last character in the string is '#', remove from array
        if self.tamuk[-1] == '#':
            self.tamuk = self.tamuk[:-1]

    # sorts the tamuk array
    def sort_tam(self):
        last_t = -1
        first_m = len(self.tamuk)
        i = 0
        while(i < first_m):
            if self.tamuk[i] == 'M':
                while(self.tamuk[first_m - 1] == 'M'):
                    first_m -= 1
                if i < first_m - 1:
                    self.swap(i, first_m - 1)
                    first_m -= 1
            if self.tamuk[i] == 'T':
                if i > last_t + 1:
                    self.swap(i, last_t + 1)
                last_t += 1
            i += 1
    # swaps elements in the i-th and j-th position of the array tamuk
    def swap(self, i, j):
        self.tamuk[i], self.tamuk[j] = self.tamuk[j], self.tamuk[i]


def main():
    raw_input = input("Please enter a string of T's, A's, and M's in any
   order (i.e. MMAATT):")
    st = SortTAM(raw_input)
    st.sort_tam()
```

```
38      print(f'The sorted string is {"".join(st.tamuk)}')
39
40  if __name__ == '__main__':
41      main()
```

Listing 3.1: sort_tam.py

Chapter 4

Tests

```
import unittest
from sort_tam import SortTAM

class TestSortTAM(unittest.TestCase):
    # test already sorted
    def test_already_sorted(self):
        tam = 'TTAAAMMMM'
        st = SortTAM(tam)
        st.sort_tam()
        self.assertEqual(''.join(st.tamuk), tam)

    #test a string with all 3 types of letters
    def test_normal(self):
        tam = 'ATMTTAMMAM'
        st = SortTAM(tam)
        st.sort_tam()
        self.assertEqual(''.join(st.tamuk), 'TTTAAAMMMM')

    #test a string with only Ts and Ms
    def test_tm(self):
        tam = 'MTMMMTTTMTM'
        st = SortTAM(tam)
        st.sort_tam()
        self.assertEqual(''.join(st.tamuk), 'TTTTTMMMMMM')

    #test a string with only As and Ms
    def test_am(self):
        tam = 'AMMMAAMAMAM'
        st = SortTAM(tam)
        st.sort_tam()
        self.assertEqual(''.join(st.tamuk), 'AAAAAMMMMMM')

    #test a string with only As and Ts
    def test_at(self):
        tam = 'ATAATATTATA'
        st = SortTAM(tam)
        st.sort_tam()
        self.assertEqual(''.join(st.tamuk), 'TTTTTAAAAAA')
```

```
39
40 if __name__ == '__main__':
41     unittest.main()
```

Listing 4.1: sort_tam_tests.py

Chapter 5

Lessons Learned

I initially had an algorithm that swapped the letter A around, but the resulting number of swaps was much more than the minimum needed. I learned some graph theory in order to merge the swaps in order to get to the minimum, but the solution would have used extra space to store the swaps, which meant it violated constraint 3 and therefore was not good enough. I then went to Professor Ammari's office hours and showed him the algorithm and asked for any suggestions and improvements. With his help, I got to the algorithm currently used. So the big lesson I learned was to not be afraid to ask for help when you are stuck.