

CSCI803 Assignment

Yao Xiao
SID 2019180015

October 7, 2020

1 Problem 1

$$A = \begin{bmatrix} 2 & 3 & 12 & 14 \\ 4 & 8 & 16 & \infty \\ 5 & 9 & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

2 Problem 2

For the first question, if the first element is ∞ , the rest of the first row need to be ∞ . In this case, all other elements need to be ∞ because they are larger than the first element on their column.

For the second question, if the bottom right element is smaller than ∞ , all the elements on the bottom row need to be smaller than ∞ . So, the other elements, each of them must be smaller than the bottom element on its column.

3 Problem 3

Actually, we can execute a process (similar to MAXHEAPIFY) to restore.

For $A[i, j]$, we can compare it with each of its neighbours and exchange it with the smallest. And this does not destroy and will restore the property of $A[i, j]$, but this will simplify the problem to both $A[i + 1, j]$ and $A[i, j + 1]$.

When $A[i, j]$ is smaller than its neighbours, the process will be terminated.

The relation is:

$$\begin{aligned} T(p) &= T(p-1) + O(1) \\ &= T(p-2) + O(1) + O(1) \\ &= T(p-3) + O(1) + O(1) + O(1) \\ &= \dots \\ &= O(p) \end{aligned} \tag{1}$$

Here is the code:

```
1 def res_tableau( tableau , i=0, j=0):  
2
```

```

3     bottom = tableau[i + 1][j] if (i + 1 < m) else float('inf')
4     right = tableau[i][j + 1] if (j + 1 < n) else float('inf')
5
6     if bottom < right:
7
8         temp = tableau[i][j]
9         tableau[i][j] = tableau[i + 1][j]
10        tableau[i + 1][j] = temp
11
12        res_tableau(tableau, i + 1, j)
13
14    if bottom > right:
15
16        temp = tableau[i][j]
17        tableau[i][j] = tableau[i][j + 1]
18        tableau[i][j + 1] = temp
19
20        res_tableau(tableau, i, j + 1)
21
22
23    def extract_min(tableau):
24
25        min = tableau[0][0]
26
27        tableau[0][0] = float('inf')
28
29        res_tableau(tableau)
30
31        return min
32
33
34    tableau = [[10, 12, 15, 17], [11, 18, 20, 25], [22, 27, 30, 35],
35              [34, 40, 44, 88]]
36
37    (m, n) = (len(tableau), len(tableau[0]))
38
39    for i in range(m * n):
40        print(extract_min(tableau))

```

Output:

```

1 def res_tableau(tableau, i=0, j=0):
2
3     bottom = tableau[i + 1][j] if (i + 1 < m) else float('inf')
4     right = tableau[i][j + 1] if (j + 1 < n) else float('inf')
5
6     if bottom < right:
7
8         temp = tableau[i][j]
9         tableau[i][j] = tableau[i + 1][j]
10        tableau[i + 1][j] = temp
11
12        res_tableau(tableau, i + 1, j)
13
14    if bottom > right:
15
16        temp = tableau[i][j]
17        tableau[i][j] = tableau[i][j + 1]
18        tableau[i][j + 1] = temp
19
20        res_tableau(tableau, i, j + 1)
21
22
23 def extract_min(tableau):
24
25     min = tableau[0][0]
26
27     tableau[0][0] = float('inf')
28
29     res_tableau(tableau)
30
31     return min
32
33
34 tableau = [[10, 12, 15, 17], [11, 18, 20, 25], [22, 27, 30, 35],
35            [34, 40, 44, 88]]
36
37 (m, n) = (len(tableau), len(tableau[0]))
38
39 for i in range(m * n):
40     print(extract_min(tableau))
41

```

4 Problem 4

In fact, the algorithm is similar as the previous. However, the difference is that we need to start with the bottom right element of the table, and then move it upwards and leftwards to the correct position.

For example, assuming that we have inserted the element into the bottom right of the matrix, now we need to move it to a suitable position in the matrix. If it has elements on the top and left, then choose the largest one to exchange with it; if it doesn't have values on both the top and left, then choose to move up or move to the left first. Considering the characteristics of matrix storage, it is better to move up first.

Even if we have an empty array and are inserting new elements, these elements will only be traversed ($i+j$). The maximum values of i and j are m and n , so the running time is $O(m + n)$.

Here is the code:

```

1 def insert(tableau, i, j):
2     if i == 0 and j == 0:
3         return
4
5     if i == 0:
6         if tableau[i][j] < tableau[i][j - 1]:
7             temp = tableau[i][j]
8             tableau[i][j] = tableau[i][j - 1]
9             tableau[i][j - 1] = temp
10
11            insert(tableau, i, j - 1)
12        return
13

```

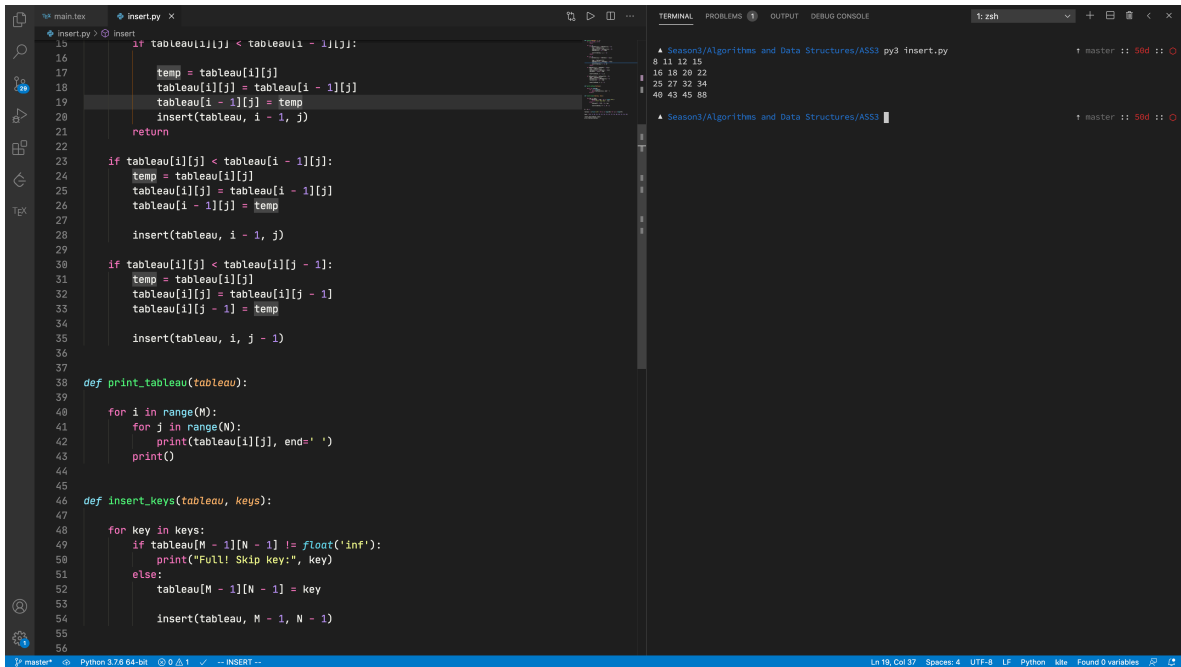
```

14     if j == 0:
15         if tableau[i][j] < tableau[i - 1][j]:
16
17             temp = tableau[i][j]
18             tableau[i][j] = tableau[i - 1][j]
19             tableau[i - 1][j] = temp
20             insert(tableau, i - 1, j)
21     return
22
23     if tableau[i][j] < tableau[i - 1][j]:
24         temp = tableau[i][j]
25         tableau[i][j] = tableau[i - 1][j]
26         tableau[i - 1][j] = temp
27
28         insert(tableau, i - 1, j)
29
30     if tableau[i][j] < tableau[i][j - 1]:
31         temp = tableau[i][j]
32         tableau[i][j] = tableau[i][j - 1]
33         tableau[i][j - 1] = temp
34
35         insert(tableau, i, j - 1)
36
37
38 def print_tableau(tableau):
39
40     for i in range(M):
41         for j in range(N):
42             print(tableau[i][j], end=' ')
43         print()
44
45
46 def insert_keys(tableau, keys):
47
48     for key in keys:
49         if tableau[M - 1][N - 1] != float('inf'):
50             print("Full! Skip key:", key)
51         else:
52             tableau[M - 1][N - 1] = key
53
54             insert(tableau, M - 1, N - 1)
55
56
57 M = N = 4
58
59 tableau = [[float('inf') for x in range(N)] for y in range(M)]
60
61 keys = [12, 8, 20, 22, 25, 32, 34, 11, 43, 27, 16, 40, 88, 15, 18,
62         45]
63
64 insert_keys(tableau, keys)

```

```
64 print_tableau(tableau)
```

Output:



The screenshot shows a code editor with a Python file named 'insert.py'. The code implements an insertion sort algorithm. It includes functions for inserting an element into a tableau, printing the tableau, and inserting keys. The terminal output shows the execution of the program, displaying the sorted tableau and the keys inserted.

```
15 def insert(tableau, i, j):
16     if tableau[i][j] < tableau[i - 1][j]:
17         temp = tableau[i][j]
18         tableau[i][j] = tableau[i - 1][j]
19         tableau[i - 1][j] = temp
20         insert(tableau, i - 1, j)
21     return
22
23 if tableau[i][j] < tableau[i - 1][j]:
24     temp = tableau[i][j]
25     tableau[i][j] = tableau[i - 1][j]
26     tableau[i - 1][j] = temp
27     insert(tableau, i - 1, j)
28
29 if tableau[i][j] < tableau[i][j - 1]:
30     temp = tableau[i][j]
31     tableau[i][j] = tableau[i][j - 1]
32     tableau[i][j - 1] = temp
33     insert(tableau, i, j - 1)
34
35 def print_tableau(tableau):
36     for i in range(M):
37         for j in range(N):
38             print(tableau[i][j], end=' ')
39         print()
40
41 def insert_keys(tableau, keys):
42     for key in keys:
43         if tableau[M - 1][N - 1] != float('inf'):
44             print("Full! Skip key:", key)
45         else:
46             tableau[M - 1][N - 1] = key
47             insert(tableau, M - 1, N - 1)
48
49 if __name__ == '__main__':
50     M = 10
51     N = 10
52     tableau = [[float('inf')] * N for _ in range(M)]
53     keys = [15, 16, 18, 20, 22, 25, 27, 32, 34, 40, 43, 45, 50]
54     insert_keys(tableau, keys)
55     print_tableau(tableau)
56
```

Terminal Output:

```
Season3/Algorithms and Data Structures/AS53 py3 insert.py
8 11 12 15
16 18 20 22
25 27 32 34
40 43 45 50
Season3/Algorithms and Data Structures/AS53
```

5 Problem 5

Here is the pseudocode **sort(A)**:

```
1 for i = 1 to m
2   insert(Y, n, n, A[i])
3 for i = 1 to m
4   A[i] = EXTRACT-MIN(Y)
```

We can sort by starting with an empty tableau and inserting all the n^2 elements in it, line 1 to 2 is inserting n^2 in to tableau which takes $O(n+n)$ and lines 3 to 4 extract n^2 and EXTRACT-MIN runs $O(n+n)$, so this gives the running time of $n^2O(n) = O(n^3)$.

6 Problem 6

```
1 def search(tableau, key):
2
3     i = 0
4     j = len(tableau[0]) - 1
5
6     while i < len(tableau) and j >= 0:
7
```

```

8         if tableau[i][j] < key:
9             i = i + 1
10
11        elif tableau[i][j] > key:
12            j = j - 1
13
14        else:
15            return True
16
17    return False
18
19
20    tableau = [[10, 12, 15, 17], [11, 18, 20, 25], [22, 27, 30, 35],
21              [34, 40, 44, 88]]
22
23
24    keys = [8, 17, 26, 88]
25
26    for key in keys:
27        if search(tableau, key):
28            print("Key", key, "found in table!")
29        else:
30            print("Error: No element found!")

```

Output:

The screenshot shows a code editor with a file named `search.py` open. The code in the editor is as follows:

```

1 def search(tableau, key):
2
3     i = 0
4     j = len(tableau[0]) - 1
5
6     while i < len(tableau) and j >= 0:
7
8         if tableau[i][j] < key:
9             i = i + 1
10
11        elif tableau[i][j] > key:
12            j = j - 1
13
14        else:
15            return True
16
17    return False
18
19
20    tableau = [[10, 12, 15, 17], [11, 18, 20, 25], [22, 27, 30, 35],
21              [34, 40, 44, 88]]
22
23
24    keys = [8, 17, 26, 88]
25
26    for key in keys:
27        if search(tableau, key):
28            print("Key", key, "found in table!")
29        else:
30            print("Error: No element found!")
31

```

The terminal output on the right shows the execution results:

```

Season3/Algorithms and Data Structures/ASS3 py3 search.py
Error: No element found!
Key 17 found in table!
Error: No element found!
Key 88 found in table!

```

The status bar at the bottom indicates the file is `search.py`, the Python version is `3.7.6 64-bit`, and the encoding is `UTF-8`.