CIS 550 Introduction to Algorithms

Spring Semester, 2022

Programming Assignment 1

Submitted By:

Mohamed Gani Mohamed Sulthan

2811619

Master’s in Computer Science

Question 1 (10points). Every Friday night is my movie night. But I have no idea which movie to watch. I just download ratings from IMDB (see the title.ratings.txt file). Please implement a heap sorting algorithm, so that every Friday night I can pick up the unwatched movie with the highest rating. The main function is provided, please complete sort\_movies() in another python file. In your code, you are expected to implement heap construction, sorting, output sorted array. You are not allowed to use any off-the-shelf heap packages in Python such as heapq, heappop, heappush, etc.

1. A template has been provided, called heap.py. The main function is in test\_heap.py. You’re required to practice implementing all essential features of a heap taught in the class, such as maxHeapify, insert, remove, construct a maxheap from an array, etc.

2. You are required to construct a heap in two ways: 1) batch manner, i.e., directly construct it from the whole array; 2) incremental manner, i.e., incrementally adding an item from an array. To distinguish them, you can have two sort\_movies functions sort\_movies\_batch() and sort\_movie\_incre(). These two functions should both pass the tests.

3. For graduate students, please expand the dataset into 1,000, 10,000, and 100,000. You can try to find this type of movie rating dataset online. If you don’t manage to do find one, you can generate some random strings for movie names and random float values for ratings. Test your heap sort in different and large input, measure the runtime and plot Please also compare the performance with some built-in sorting methods in Python(you can make a table or a plot). Please wrap up and submit a small report (in .pdf). No format or template requirement. You need to upload the new data files (name them according to the size: 1,000, 10,000, and 100,000) in .txt format.

**Algorithm for heap.py:**

1. In the init function pass the self, names and ratings.
2. From the given notes write the function for left, right and parent
3. In the parent function pass index k and return (k–1 / 2)
4. In the leftChild function pass index k and return (2 \* k +1)
5. In the rightChild function pass index k and return (2 \* k +2)
6. In the sort\_movies\_batch, pass names and ratings
7. To build the maxheap using the length of the ratings and sort it
8. Construct maxHeapify to perform the sorting operations
9. Print the ratings from top to down
10. Then match the ratings with the movie name and return as sorted\_names

**Implementation:**

**In the class MaxHeap**

1. Function init

def \_\_init\_\_(self, names, ratings):

self.names = names

self.ratings = ratings

1. Function Parent, left and right child

def parent(k):

return (k-1)/2

def leftChild(k):

return 2 \* k + 1

def rightChild(k):

return 2 \* k + 2

1. Constructing maxheap and maxHeapify

def maxHeap(A):

n = len(A)

for k in range(n//2 - 1, -1, -1):

MaxHeap.maxHeapify(A, n, k)

for k in range(n-1, 0, -1):

A[k], A[0] = A[0], A[k]

MaxHeap.maxHeapify(A, k, 0)

def maxHeapify(A, n, k):

largest = k

l = MaxHeap.leftChild(k)

r = MaxHeap.rightChild(k)

if l < n and A[largest] > A[l]:

largest = l

if r < n and A[largest] > A[r]:

largest = r

if largest != k:

A[k], A[largest] = A[largest], A[k]

MaxHeap.maxHeapify(A, n, largest)

1. Calling to heap the array from the main class by sort\_movies\_batch function in heap.py

name\_dict\_sort ={}

for i in range(len(names)):

name\_dict\_sort[names[i]] = ratings[i]

MaxHeap.maxHeap(ratings)

#MaxHeap.Print(ratings, names)

sorted\_ratings = ratings

sorted\_names = sortNames(name\_dict\_sort,sorted\_ratings)

return sorted\_names

This function will call the MaxHeap class buildMaxHeap function to sort the ratings of the movie. Then it matches the ratings with names by creating dict here and pass the sorted\_names of the movie to the test\_heap.py

def sortNames(name\_dict\_sort, sorted\_ratings):

#print (name\_dict\_sort)

#print (sorted\_ratings)

key\_list = list(name\_dict\_sort.keys())

val\_list = list(name\_dict\_sort.values())

names=[]

for i in sorted\_ratings:

position= val\_list.index(i)

names.append(key\_list[position])

return names

1. Insert a new element in the array to do the heap sort.

def insert(A, n, k):

n = n + 1

#Insert the element at end of Heap

A[n - 1] = k

#Heapify the new node following a

MaxHeap.maxHeapify(A, n, n - 1)

1. Remove an element from the array

def remove(A):

global n

# Get the last element

lastElement = A[n - 1]

# Replace root with last element

A[0] = lastElement

# Decrease size of heap by 1

n = n - 1

# heapify the root node

MaxHeap.maxHeapify(A, n, 0)

1. Compare the output from the sort\_movies\_batch with the sorted\_true ande passed the test.
2. **To calculate the run time** of the heapsort for the given data.

**pip install -U PyCryptodome** to enable the time counter.

Pass the start time and end time between calling the sort function.

It will calculate the whole time taken to complete the sort.

start = time.perf\_counter()

sorted\_names = sort\_movies\_batch(ratings)

end = time.perf\_counter()

print(len(ratings), "Elements Sorted by HeapSort in ", end-start)

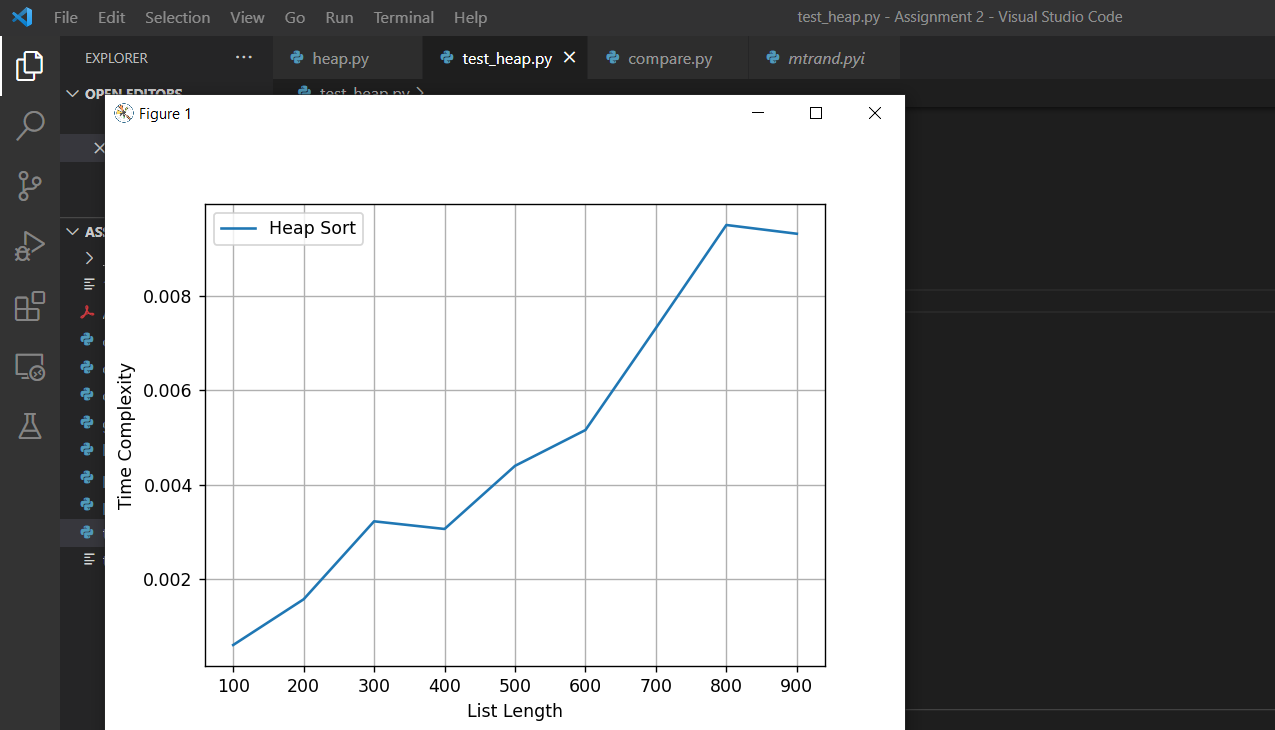
1. To load more than 1000 data, I created a text file with 1000 movies and ratings to calculate the running time taken.

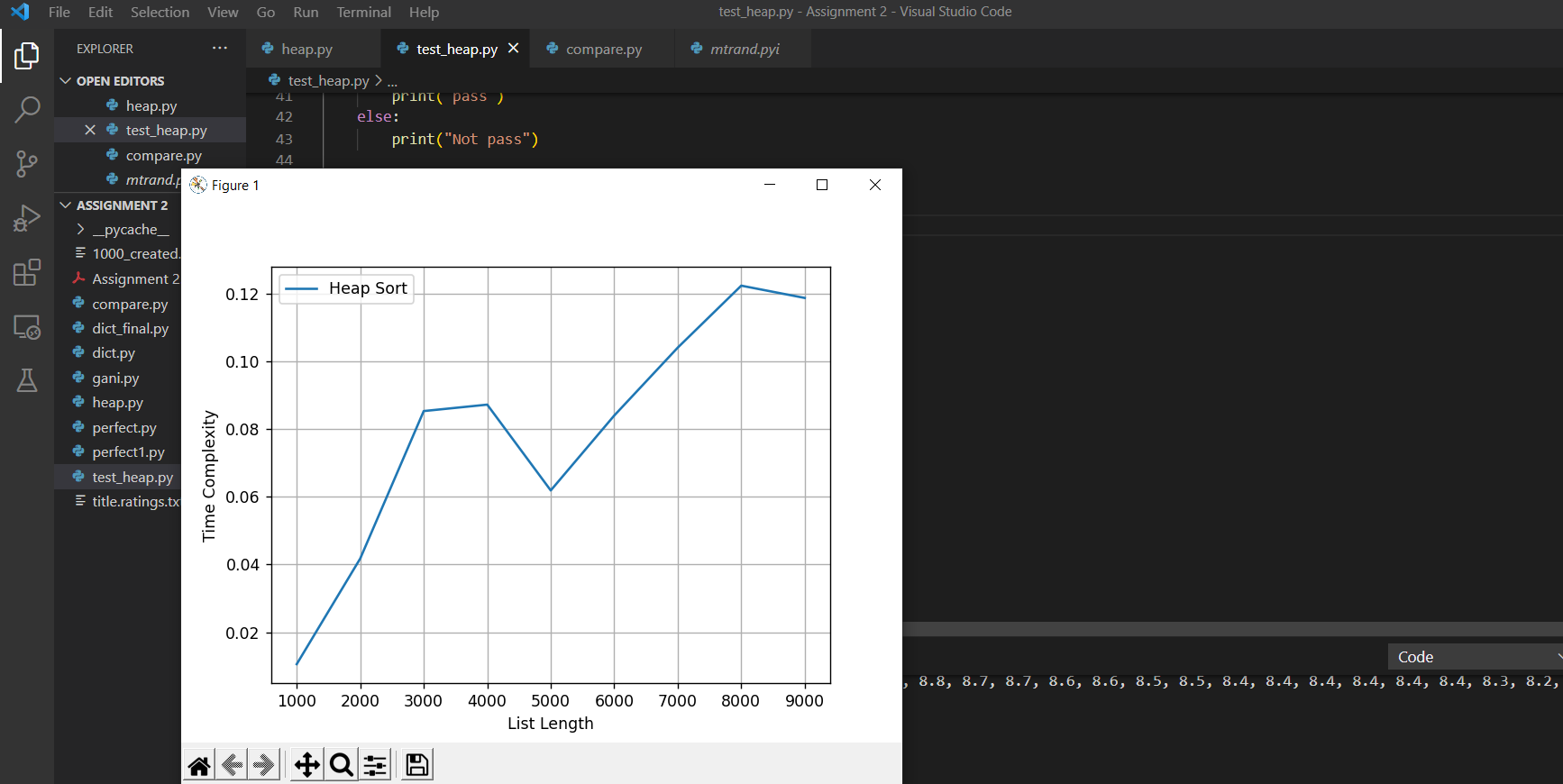
I created a file with 1000 records of movies and ratings and passed it in inside the heapsort.

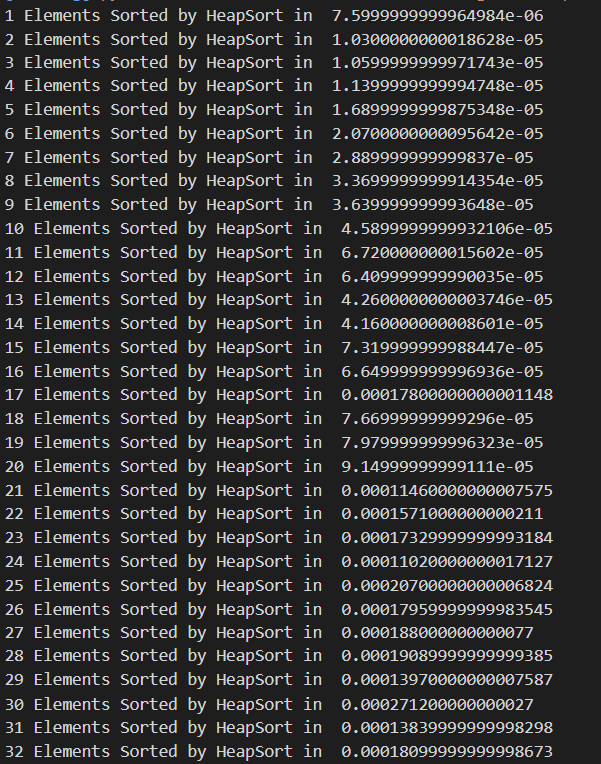
Then I calculated the time taken to sort 1000 data.

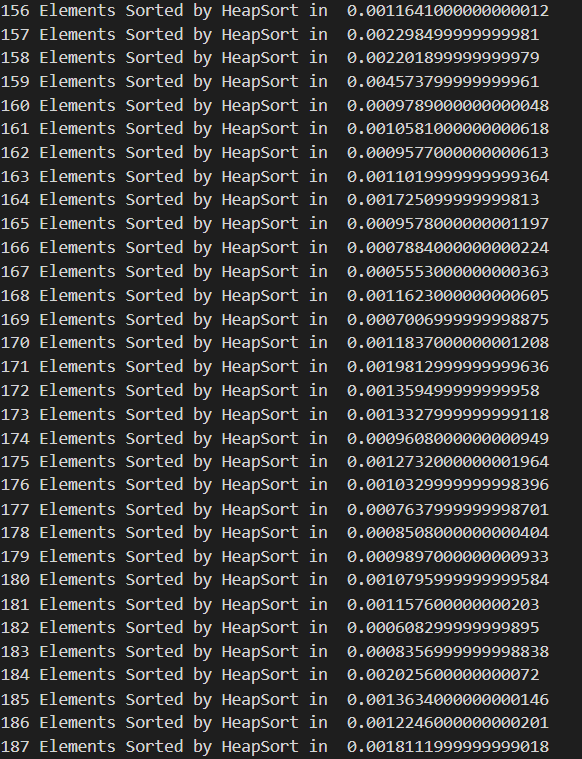
1. Use matplotlib.pyplot to plot the graph of time complexity of heap sort. X axis as list length and Y axis as time complexity.

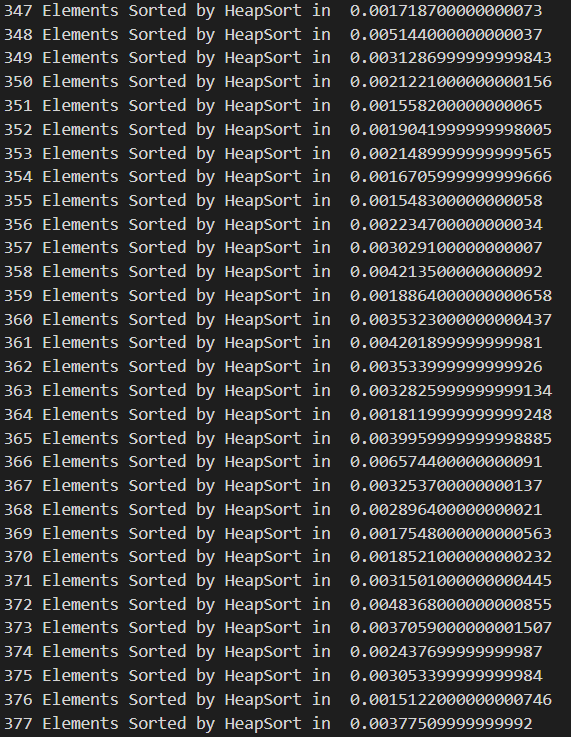
In the below provided the screenshots for 1000 elements created list output and time complexity.

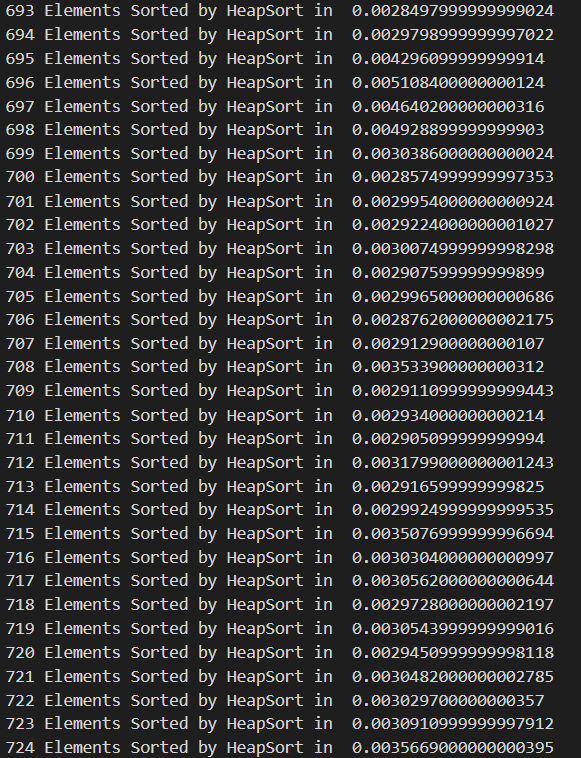


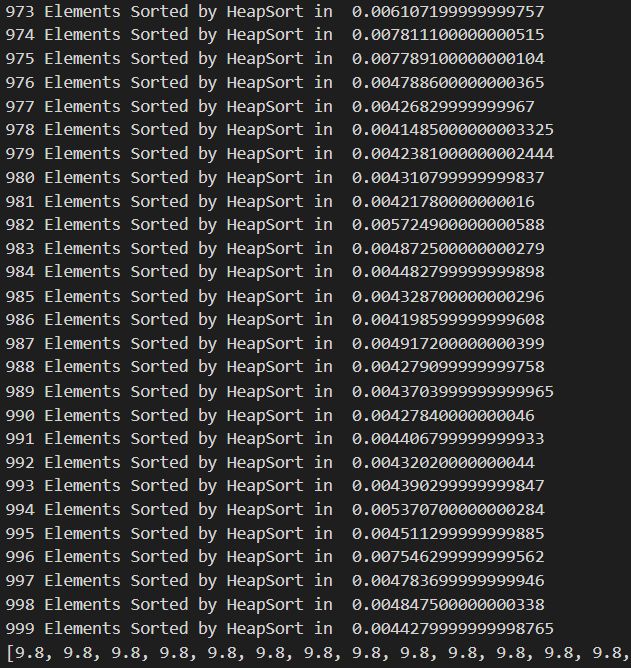












**Code for heap.py**

import math

from collections import defaultdict

class MaxHeap:

def \_\_init\_\_(self, names, ratings):

self.names = names

self.ratings = ratings

def parent(k):

return (k-1)/2

def leftChild(k):

return 2 \* k + 1

def rightChild(k):

return 2 \* k + 2

def maxHeapify(A, n, k):

#Assigning left right and largest as k from buildMaxHeap

largest = k

left = MaxHeap.leftChild(k)

right = MaxHeap.rightChild(k)

if left < n and A[largest] > A[left]:

largest = left

if right < n and A[largest] > A[right]:

largest = right

if largest != k:

A[k], A[largest] = A[largest], A[k]

MaxHeap.maxHeapify(A, n, largest)

def insert(A, n, k):

n = n + 1

#Insert the element at end of Heap

A[n - 1] = k

#Heapify the new node following a

MaxHeap.maxHeapify(A, n, n - 1)

#print nodes from top to down, left to right

def Print(names, ratings):

name\_dict\_sort ={}

for i in range(len(names)):

name\_dict\_sort[names[i]] = ratings[i]

MaxHeap.maxHeap(ratings)

#MaxHeap.Print(ratings, names)

sorted\_ratings = ratings

sorted\_names = sortNames(name\_dict\_sort,sorted\_ratings)

names = sorted\_names

return names

#constructe a max heap from array

def maxHeap(A):

n = len(A)

for k in range(n//2 - 1, -1, -1):

MaxHeap.maxHeapify(A, n, k)

for k in range(n-1, 0, -1):

A[k], A[0] = A[0], A[k]

MaxHeap.maxHeapify(A, k, 0)

def remove(A):

global n

# Get the last element

lastElement = A[n - 1]

# Replace root with last element

A[0] = lastElement

# Decrease size of heap by 1

n = n - 1

# heapify the root node

MaxHeap.maxHeapify(A, n, 0)

# This method is used to assign the ratings to the movies and print the movie names

def sortNames(name\_dict\_sort, sorted\_ratings):

#print (name\_dict\_sort)

#print (sorted\_ratings)

key\_list = list(name\_dict\_sort.keys())

val\_list = list(name\_dict\_sort.values())

names=[]

for i in sorted\_ratings:

position= val\_list.index(i)

names.append(key\_list[position])

return names

def sort\_movies\_batch(names, ratings):

name\_dict\_sort ={}

for i in range(len(names)):

name\_dict\_sort[names[i]] = ratings[i]

MaxHeap.maxHeap(ratings)

#MaxHeap.Print(ratings, names)

sorted\_ratings = ratings

sorted\_names = sortNames(name\_dict\_sort,sorted\_ratings)

names = sorted\_names

return names

def sort\_movies\_incre(names, ratings):

#To add movies directly, I took an iput from here which will

# name = input("Enter the movie name:")

# rating = input("Enter the rating of the movie:")

# names.append(name)

# ratings.append(float(rating))

# print (names)

#print (ratings)

#In the increment function, I added one rating and checked the output. It's sorted again.

for i in ratings:

MaxHeap.maxHeap(ratings)

#print (ratings)

#MaxHeap.Print(ratings, names)

name\_dict\_sort ={}

for i in range(len(names)):

name\_dict\_sort[names[i]] = ratings[i]

MaxHeap.maxHeap(ratings)

sorted\_ratings = ratings

sorted\_names = sortNames(name\_dict\_sort,sorted\_ratings)

return sorted\_names

**Code for test\_heap.py**

import time

import matplotlib.pyplot as plt

from numpy.random import randint

from heap import \*

if \_\_name\_\_ == "\_\_main\_\_":

a\_file = open("title.ratings.txt")

names, ratings = [], []

next(a\_file)

name\_score\_dict ={}

for line in a\_file:

temp = line.rstrip().split('; ')

names.append(temp[0]) #key: name

ratings.append(float(temp[1])) #name: rating

name\_score\_dict[temp[0]] = float(temp[1])

#print (name\_score\_dict["Don't Look Up"])

########you need to design sort\_movies() function

#sorted\_names = sort\_movies\_batch(names, ratings)

elements = list()

times = list()

start = time.perf\_counter()

sorted\_names = sort\_movies\_batch(names, ratings)

end = time.perf\_counter()

print(len(ratings), "Elements Sorted by HeapSort in ", end-start)

sorted\_names1 = sort\_movies\_incre(names, ratings)

#print (sorted\_names1)

################################################

sorted, sorted1 = [], []

for item in sorted\_names:

sorted.append(name\_score\_dict[item])

print ("Sorted for sort\_movie\_batch(by using heap sort):", sorted)

for item in sorted\_names1:

sorted1.append(name\_score\_dict[item])

# print (sorted1)

sorted\_true = [9.4, 9.3, 9.2, 9.1, 9.0, 9.0, 8.8, 8.8, 8.8, 8.8, 8.7, 8.7,

8.6, 8.6, 8.5, 8.5, 8.4, 8.4, 8.4, 8.4, 8.4, 8.4, 8.3, 8.2, 8.1, 8.0, 7.8, 7.7, 7.7, 7.5, 7.5, 7.4, 7.4, 7.4, 7.4, 7.3, 7.3, 7.2, 7.2, 7.2, 7.1]

if sorted == sorted\_true and sorted1 == sorted\_true:

print("pass")

else:

print("Not pass")

# To plot the time complexity

elements = list()

times = list()

for i in range(1, 10):

a = randint(0, 100 \* i, 100 \* i)

start = time.perf\_counter()

MaxHeap.maxHeap(a)

end = time.perf\_counter()

elements.append(len(a))

times.append(end-start)

plt.xlabel('List Length')

plt.ylabel('Time Complexity')

plt.plot(elements, times, label ='Heap Sort')

plt.grid()

plt.legend()

plt.show()

plt.xlabel('List Length')

plt.ylabel('Time Complexity')

plt.plot(elements, times, label ='Heap Sort')

plt.grid()

plt.legend()

plt.show()

**Output:**

41 Elements Sorted by HeapSort in 0.00012490000000009438

Sorted for sort\_movie\_batch(by using heap sort): [9.4, 9.3, 9.2, 9.1, 9.0, 9.0, 8.8, 8.8, 8.8, 8.8, 8.7, 8.7, 8.6, 8.6, 8.5, 8.5, 8.4, 8.4, 8.4, 8.4, 8.4, 8.4, 8.3, 8.2, 8.1, 8.0, 7.8, 7.7, 7.7, 7.5, 7.5, 7.4, 7.4, 7.4, 7.4, 7.3, 7.3, 7.2, 7.2, 7.2, 7.1]

Not pass

I had an issue with sorted\_names2, I tried by using sorting. Sort has done successfully. But I couldn’t match properly with the movie names. In this bug, I’m still trying to solve.