Name: Adham Ali

ID: 90223243

***Assignment 5 (Analysis and Complexity):***

In the code submitted, there were 8 functions implemented:

🡪int main

🡪heapify\_max

🡪heapify\_min

🡪build\_max\_heap

🡪build\_min\_heap

🡪maximum\_subsequence\_sum

🡪remove

🡪tokens

**heapify\_max:**

**A screenshot of a computer program

Description automatically generated**

Complexity: O(log n)/The recursive function performs constant work for each heap level. Since the heap is a binary heap, the height of the heap is logarithmic concerning the number of nodes (n), resulting in a logarithmic time complexity. Note: the SWAP function in the heapify\_max will have a complexity of O(1).

**heapify\_min:**

**A screenshot of a computer program

Description automatically generated**

Complexity: O(log(n))/ same as heapify\_max

**build\_max\_heap:**

**A screen shot of a computer code

Description automatically generated**

Complexity: O (n log(n))/ The function calls heapify\_max on each non-leaf node of the heap. Since there are n/2 non-leaf nodes in a heap with n nodes, O(((n/2)-1) \*log(n)) = O(n log(n)).

**build\_min\_heap:**

**A screenshot of a computer program

Description automatically generated**

Complexity: O(n log(n))/ The function calls heapify\_min on each non-leaf node of the heap. Since there are n/2 non-leaf nodes in a heap with n nodes, O(((n/2)-1) \*log(n)) = O (n log(n)).

**maximum\_subsequence\_sum:**

**A screenshot of a computer program

Description automatically generated**

Complexity: O(n)/ The function uses a single loop to iterate over the input vector (differences). Each iteration involves constant time operations, so the time complexity is linear concerning the size of the input vector.

**remove:**

**A screenshot of a computer program

Description automatically generated**

Complexity: O(n log(n))/ In the worst case, when removing the root, the function involves erasing the root element, which has a complexity of O(n) and rebuilding the heap. The build\_max\_heap and build\_min\_heap functions each have an O(n log(n)) time complexity, and there is nothing else in the function. Therefore, the complexity is O(n log(n))+O(n) = O(n log(n)).

**tokens:**

**A screen shot of a computer program

Description automatically generated**

Complexity: O(m)/ Where m is the length of the input string.

Explanation: The function iterates over the characters of the input string once, and each iteration involves constant time operations. Therefore, the time complexity is linear with respect to the length of the input string.

**(GOING TO THE INT MAIN):**

A screenshot of a computer program

Description automatically generated

1. Here, we have an if statement, which is O(1), and a while loop over the lines in the open file so that it will have the complexity of O(n). Additionally, there is a tokens function with an O(m) complexity. Therefore, the total complexity of the code in this picture will be O(n\*m)+O(1) = O(n\*m).

A screen shot of a computer program

Description automatically generated

1. Here, we have some assignments that have a complexity of O(1), and we have the build\_ max\_heap function, which has a complexity of O(n log(n)). Moreover, we have a for loop with a complexity of O(10) = O(1), and there is the remove function inside it, which has a complexity of O(n log(n)). Therefore, the complexity here will be O(10 \*(n log(n))) = O(n log(n)). In addition, we have the same operations. but instead of using the build\_heap\_max, we used the buil\_min\_heap function, which has the same complexity as build\_heap\_max. As a result, the total complexity of the code in this picture will be

O(n log(n))+ O(n log(n)) )+O(1) = O(n log(n)).

A screen shot of a computer program

Description automatically generated

1. This picture has a for loop with an O(n) complexity. Moreover, we have some declaration statements and assignments with a complexity O(1). The last thing is the maximum\_subsequence\_sum function, which has a complexity of O(n). Therefore, the total complexity of the code in this picture will be O(n)+O(n)+O(1) = O(n).
2. The total complexity of the program: O(n\*m)+O(n log(n))+O(n)=**O(n log(n)).**
3. Output of the code:

A screenshot of a computer

Description automatically generated