



HEAP INSERTION TIME USING PYTHON

Team Members:

Krishna Meghana Chirumamilla – 999903231

Manogna Arla – 999902989


Harsha Vardhan Reddy - 999902545

Priyanka Chekuri - 999903142






ABSTRACT

- The Heap Insertion Time is a Python program that allows users to insert nodes into a binary heap data structure interactively. It determines the average insertion time up to the current step by measuring the time it takes to insert each node. The program has a command-line interface via which the user may insert nodes.
 - Max heapify is an algorithm used to maintain the heap property in a max heap data structure in Python. Max heapify efficiently converts a binary tree into a max heap by comparing parent nodes with children and swapping as needed in a bottom-up manner. This allows efficient retrieval of the maximum element at the root.
- 




INTRODUCTION

- The actual average time per operation, however, may vary from the worst-case bounds for random input sequences. In accordance with some introductory algorithm texts, binary heaps offer $O(1)$ constant time for insertion and deletion on average. On the other hand, the other publications use more conservative $O(\log N)$ worst-case complexity.
 - This program uses max heapify algorithm to build the binary tree structure. This max heapify algorithm will insert the highest element as a root node and calculates the insertion time for every insertion and average insertion time after every node insertion.
 - The program repeatedly asks the user to enter the integer value to insert into the node until the user wants to quit the insertion by entering q.
- 




REQUIREMENTS

- **Python 3.x:** The code is written in Python, so you need to have Python installed on your system
You can download Python from the official website.
 - **Time Library:** By importing this library package this allows us to work with time representations, measure and benchmark code, pause programs, and display formatted timestamps. The time and datetime modules allow date/time manipulations in Python.
 - **IDE or Text Editor:** You need an IDE or a text editor to write and run the Python code Some popular IDEs include Replit, PyCharm, Visual Studio Code and Spyder. You can also use a simple text editor like Sublime Text or Notepad++.
 - **Operating system:** The code is written in Python and should work on any operating system that supports Python.
- 




ALGORITHM

- Max Heapify algorithm is used to maintain the max heap property in a binary max heap data structure. The largest element is at the root. A max heap has the property that each node is greater than or equal to its child nodes.
 - Heapify works based on bottom-up approach. For each parent node, it compares the parent with its left and right children. If the parent is smaller than either child, it is swapped with the larger child to maintain the max heap order.
 - Time complexity is $O(\log N)$ for heapify on a subtree of N nodes.
- 




FEATURES OF MAX HEAPIFY ALGORITHM

- Bottom-up approach - starts heapifying from the lowest parent level and moves upwards to the root. This allows efficient maintenance of the max heap property.
 - Recursive implementation - the heapify method calls itself recursively to heapify each subtree from bottom to top. Swapping nodes - nodes are swapped with a child having greater value to maintain the max heap order. Swaps bubbles the larger child up.
 - Used in heap operations - heapify is used in key operations like insert and extract max on heaps for efficiency. Maintains completeness - preserves the complete binary tree property of the underlying data structure.
- 



DEPENDENCIES

- There are no external dependencies for the Binary Heap Insertion Program.
 - It measures time using the built-in time module.
- 

CODE EXPLANATION

```
2
3 class BinaryHeap:
4     def __init__(self):
5         self.heap = [] # Initialize an empty heap
6         self.total_time = 0 # Initialize total insertion time
7         self.num_insertions = 0 # Initialize number of insertions
8
```

```
8
9 def insert(self, node):
10     start_time = time.time() # Record start time
11     self.heap.append(node) # Append node to the heap
12     self._up_heapify(len(self.heap) - 1) # Perform up-heapify
13     end_time = time.time() # Record end time
14
15     insertion_time = end_time - start_time # Calculate insertion time
16     self.total_time += insertion_time # Update total insertion time
17     self.num_insertions += 1 # Increment number of insertions
18     average_time = self.total_time / self.num_insertions # Calculate average insertion time
19
20     print("Node", node, "inserted.")
21     print("Insertion time:", insertion_time, "seconds")
22     print("Average insertion time:", average_time, "seconds")
23     print("Heap values:", self.heap) # Print all heap values
24     print()
```



CODE EXPLANATION (contd.)

```
25
26 ✓ def _up_heapify(self, index):
27 ✓     while index > 0:
28         parent = (index - 1) // 2
29 ✓         if self.heap[parent] <= self.heap[index]:
30             break
31         self.heap[parent], self.heap[index] = self.heap[index], self.heap[parent]
32         index = parent
33
```

```
34 heap = BinaryHeap() # Create an instance of BinaryHeap
35
36 print("Welcome to the Binary Heap Insertion Program!")
37 print("Please enter the nodes you want to insert into the heap.")
38 print("Enter 'q' to quit the program.")
39
40 ✓ while True:
41     node = input("Enter a node (or 'q' to quit): ")
42 ✓     if node == 'q':
43         break
44
45     # Validate input to accept only numbers
46 ✓     try:
47         node = int(node)
48 ✓     except ValueError:
49         print("Invalid input! Please enter a valid integer number.")
50         continue
51
52     heap.insert(node) # Insert the node into the heap
53
54 print("Program terminated.")
```



ERROR HANDLING

- To validate user input, the program includes error handling. It determines if the input is a valid integer value for insertion.
 - A Value Error is thrown whenever an incorrect input is given, such as a non-numeric character or an empty string.
 - This mistake is detected by the program, which shows an error notice and requests the user for proper input.
- 

OUTPUT

```
C:\WINDOWS\py.exe X + v
Welcome to the Binary Heap Insertion Program!
Please enter the nodes you want to insert into the heap.
Enter 'q' to quit the program.
Enter a node (or 'q' to quit): 45
Node 45 inserted.
Insertion time: 6.699992809444666e-06 seconds
Average insertion time: 6.699992809444666e-06 seconds
Heap values: [45]

Enter a node (or 'q' to quit): 2
Node 2 inserted.
Insertion time: 7.799972881556916e-06 seconds
Average insertion time: 7.249982445500791e-06 seconds
Heap values: [2, 45]


Enter a node (or 'q' to quit): 3
Node 3 inserted.
Insertion time: 9.300012607127428e-06 seconds
Average insertion time: 7.93332583270967e-06 seconds
Heap values: [2, 45, 3]

Enter a node (or 'q' to quit): 9.9
Invalid input! Please enter a valid integer number.
Enter a node (or 'q' to quit): dgwe
Invalid input! Please enter a valid integer number.
Enter a node (or 'q' to quit): 2
Node 2 inserted.
Insertion time: 1.300001167692244e-05 seconds
Average insertion time: 9.199997293762863e-06 seconds
Heap values: [2, 2, 3, 45]

Enter a node (or 'q' to quit):
```



CONCLUSION

- Insertion times are very fast, in the order of microseconds, even as the heap grows. This aligns with the expected $O(\log N)$ time complexity rather than a constant time complexity.
 - The Binary Heap Insertion Program allows you to insert nodes into a binary heap interactively. Users can acquire insights into the heap data structure's performance characteristics by measuring the time required for each insertion and computing the average insertion time.
- 



THANK YOU

