Project Report

Course Name: Algorithm Design and Analysis Lab
Course Code: CSE 132

Submitted to:
Nasif Istiak Remon
Lecturer
Department of Computer Science & Engineering,
Metropolitan University, Sylhet.

Submitted by:
Abid Shahriar
232-115-022
CSE-59 (A)

Department of Computer Science & Engineering
Metropolitan University, Sylhet.

Date of submission: 13.04.2025

1. Introduction

This report presents an interactive visualizer for the Heap Sort algorithm using a Min Heap, developed in C++. The program vividly demonstrates the step-by-step construction and usage of a Min Heap for sorting a list of user-defined size. With integrated console-based visualization, the project highlights operations such as insertion, heapify, and extraction while providing real-time feedback and comparison/swap counts. This tool aims to assist students in understanding the internal workings of heap-based sorting techniques through visual feedback and structured output.

2.Problem Description

Sorting is a fundamental concept in computer science, used in countless applications such as searching, optimization, and data organization. Heap Sort is an efficient, comparison-based sorting algorithm that uses a binary heap data structure. While Max Heap is typically used for descending order, this visualizer focuses exclusively on Min Heap, where the smallest element is always at the root. The algorithm repeatedly extracts the minimum element and rebuilds the heap until the array is sorted. The goal of this project is to demonstrate each of these steps interactively, enabling users to visualize the heap as a tree and observe how data gets sorted.

3. Solution Approach

The solution uses a binary Min Heap, implemented using a dynamic array.
Heap Construction:

- User enters the number of elements and the elements themselves.
- Each element is inserted into the heap with a bubble-up process to maintain the Min Heap property.

Sorting Phase (Heap Sort):

- The root (minimum) is extracted and swapped with the last element.
- A bubble-down (heapify) operation is performed to restore the heap.
- This process is repeated until all elements are sorted.

Visualization & Highlighting:

- After each insertion or heapify operation, the heap is displayed as a tree.
- The current node being inserted or heapified is highlighted.
- One-second fixed delays allow users to follow each update clearly.
- Comparisons and swaps are tracked and displayed at the end.

4. Code Structure and Design

4.1 Input Handling

- The user is prompted to enter the number of elements followed by the elements themselves.
- No limit is placed on input size.

4.2 Heap Operations and Update

- A dynamic array represents the Min Heap.
- Insertions maintain the heap property using bubble-up.
- The sorting phase uses bubble-down to restore the heap after extracting the minimum.

4.3 Display Functions

- **Tree Visualization:** The heap is printed as a structured tree using indentation based on depth.
- **Highlighting:** The current node under operation is wrapped in square brackets ([x]).
- **Delay:** A fixed 1-second delay (std::this_thread::sleep_for) is applied after each operation to help users follow along.

4.4 Decision Messages

- During operations, the program prints decisions such as:
    - "Swapping with parent..." during insertion
    - "Heapifying..." during extraction
- Each decision includes index and value details for clarity.

4.5 Comparison & Swap Counter

- Every comparison and swap is tracked.
- Final totals are shown after the heap sort completes.

5. Time Complexity

- **Heap Insertion:** O(log n) per insertion
- **Heapify (Extraction):** O(log n) per deletion

Heap Sort (Overall):

- **Time Complexity:** O(n log n)
- **Space Complexity:** O(n) (heap array and output vector)

6. Additional Considerations

Platform Compatibility:

- The program uses system("cls") for Windows and system("clear") for macOS/Linux, determined by preprocessor macros.
- This ensures cross-platform terminal clearing.

Educational Value:

- This tool provides a clear demonstration of how Min Heaps work in sorting.
- The combination of delays, tree visualization, and decision tracing enhances understanding.

Potential Enhancements:

- Adjustable delay time
- Color-coded node display
- GUI-based tree rendering for enhanced clarity

7. Conclusion

This C++ program offers an interactive and educational visualization of the Heap Sort algorithm using a Min Heap. It emphasizes the internal process behind heap operations with clear, step-by-step feedback and visual explanations. The approach not only reinforces theoretical understanding but also serves as a practical tool for learning heap-based sorting methods.

8. Future Scope

- **Customizable Delay:** Allow users to adjust delay duration for better pacing control.
- **Color Output:** Use color to highlight comparisons, swaps, and active nodes.
- **Graphical Interface:** Upgrade to a GUI (e.g., with Qt or SFML) for better heap visualization.
- **Step Navigation:** Add options to pause, resume, or step backward through operations.
- **Support for Max Heap:** Let users toggle between Min Heap and Max Heap visualizations.
- **Web Version:** Adapt for web or mobile to increase accessibility.

9. Limitations

- **Fixed Delay:** The current one-second delay might feel slow or fast depending on user preference.
- **Terminal-Only UI:** Visualization may become cluttered for large arrays in a console environment.
- **No Error Handling:** Input validation is minimal; invalid or non-numeric input could crash the program.
- **Limited Interactivity:** Users cannot control the flow once sorting starts (e.g., skip or reverse steps).
- **Platform Dependency:** Uses system("cls"/"clear") which may not behave consistently across all terminal types.

10. GitHub Repository

The complete source code for this project :

🔗 **GitHub Link:** https://github.com/CodewithShahriar/HeapSort-Visualizer