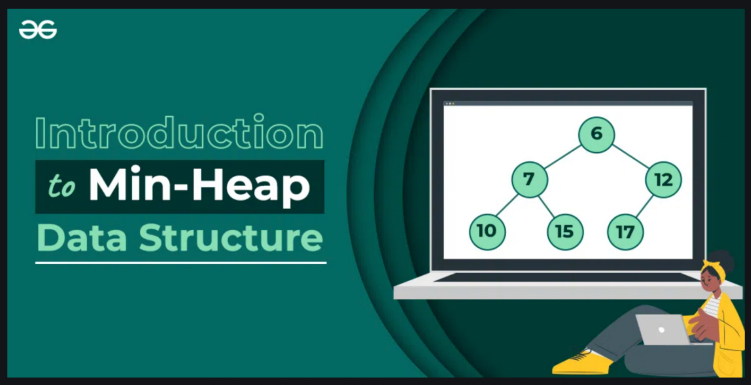
Introduction to Min-Heap – Data Structure and Algorithm Tutorials

Read Discuss Courses Practice

*A **Min-Heap** is defined as a type of Heap Data Structure in which each internal node is smaller than or equal to its children.*

The heap data structure is a type of binary tree that is commonly used in computer science for various purposes, including sorting, searching, and organizing data.



Introduction to Min-Heap – Data Structure and Algorithm Tutorials

Purpose and Use Cases of Min-Heap:

- *Priority Queue: One of the primary uses of the heap data structure is for implementing priority queues.*
- *Dijkstra's Algorithm: Dijkstra's algorithm is a shortest path algorithm that finds the shortest path between two nodes in a graph. A min heap can be used to keep track of the unvisited nodes with the smallest distance from the source node.*
- *Sorting: A min heap can be used as a sorting algorithm to efficiently sort a collection of elements in ascending order.*
- *Median finding: A min heap can be used to efficiently find the median of a stream of numbers. We can use one min heap to store the larger half of the numbers and one max heap to store the smaller half. The median will be the root of the min heap.*

Min-Heap Data structure in Different languages:

1. Min-Heap in C++

A min heap can be implemented using the **priority_queue** container from the Standard Template Library (STL). The **priority_queue** container is a type of container adapter that provides a way to store elements in a queue-like data structure in which each element has a priority associated with it.

```
Syntax: priority_queue<int, vector<int>, greator<int>>minH;
```

2. Min-Heap in Java

In Java, a min heap can be implemented using the **PriorityQueue** class from **java.util package**. The PriorityQueue class is a priority queue that provides a way to store elements in a queue-like data structure in which each element has a priority associated with it.

```
Syntax: PriorityQueue<Integer> minHeap = new PriorityQueue<Integer>
();
```

Recently Published

Top 10 Visual Studio Code Alternatives

Read

Data Stru (DSA) Gu interviev

3. Min-Heap in Python

In Python, a min heap can be implemented using the heapq module, which provides functions for implementing heaps. Specifically, the heapq module provides a way to create and manipulate heap data structures.

```
Syntax: heap = []
        heapify(heap)
```

4. Min-Heap in C#

In C#, a min heap can be implemented using the PriorityQueue<T> class from the System.Collections.Generic namespace. The PriorityQueue<T> class is a priority queue that provides a way to store elements in a queue-like data structure in which each element has a priority associated with it.

```
Syntax: var minHeap = new PriorityQueue<int>();
```

5. Min-heap in JavaScript

A min heap is a binary tree where every node has a value less than or equal to its children. In JavaScript, you can implement a min heap using an array, where the first element represents the root node, and the children of a node at index **i** are located at indices **2i+1** and **2i+2**.

```
Syntax: const minHeap = new MinHeap();
```

Difference between Max Heap and Min Heap

	Min Heap	Max Heap
1.	In a Min-Heap the key present at the root node must be less than or equal to among the keys present at all of its children.	In a Max-Heap the key present at the root node must be greater than or equal to among the keys present at all of its children.
2.	In a Min-Heap the minimum key element is present at the root.	In a Max-Heap the maximum key element is present at the root.
3.	A Min-Heap uses the ascending priority.	A Max-Heap uses the descending priority.
4.	In the construction of a Min-Heap, the smallest element has priority.	In the construction of a Max-Heap, the largest element has priority.
5.	In a Min-Heap, the smallest element is the first to be popped from the heap.	In a Max-Heap, the largest element is the first to be popped from the heap.

Internal Implementation of Min-Heap Data Structure:

*A **Min heap is typically represented as an array**.*

- *The root element will be at **Arr[0]**.*
- *For any ith node **Arr[i]**:*
  - ***Arr[(i -1) / 2]** returns its parent node.*
  - ***Arr[(2 * i) + 1]** returns its left child node.*
  - ***Arr[(2 * i) + 2]** returns its right child node.*

The Internal Implementation of the Min-Heap requires 3 major steps:

1. **Insertion**: To insert an element into the min heap, we first append the element to the end of the array and then adjust the heap property by repeatedly swapping the element with its parent until it is in the correct position.
2. **Deletion**: To remove the minimum element from the min heap, we first swap the root node with the last element in the array, remove the last element, and then adjust the heap property by repeatedly swapping the element with its smallest child until it is in the correct position.
3. **Heapify**: A heapify operation can be used to create a min heap from an unsorted array.

Operations on Min-heap Data Structure and their Implementation:

Here are some common operations that can be performed on a Heap Data Structure,

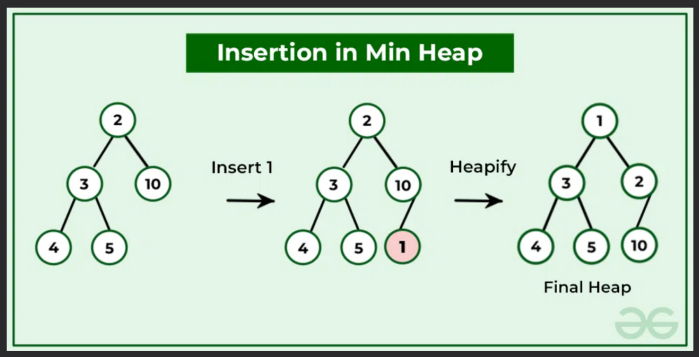
1. Insertion in Min-Heap Data Structure:

Elements can be inserted into the heap following a similar approach as discussed above for deletion. The idea is to:

- The insertion operation in a min-heap involves the following steps:
- Add the new element to the end of the heap, in the next available position in the last level of the tree.
- Compare the new element with its parent. If the parent is greater than the new element, swap them.
- Repeat step 2 until the parent is smaller than or equal to the new element, or until the new element reaches the root of the tree.
- The new element is now in its correct position in the min heap, and the heap property is satisfied.

Illustration:

Suppose the Heap is a Min-Heap as:



Insertion in Min-Heap

Implementation of insertion operation in Min-Heap:

C++ Java Python3 C# Javascript

```python
def insert_min_heap(heap, value):
    # Add the new element to the end of the heap
    heap.append(value)
    # Get the index of the last element
    index = len(heap) - 1
    # Compare the new element with its parent and swap if necessary
    while index > 0 and heap[(index - 1) // 2] > heap[index]:
        heap[index], heap[(index - 1) //
                          2] = heap[(index - 1) // 2], heap[index]
        # Move up the tree to the parent of the current element
        index = (index - 1) // 2


heap = []
values = [10, 7, 11, 5, 4, 13]
for value in values:
    insert_min_heap(heap, value)
    print(f"Inserted {value} into the min-heap: {heap}")
```

Output

```
Inserted 10 into the min-heap: 10
```

```
Inserted 7 into the min-heap: 7 10
Inserted 11 into the min-heap: 7 10 11
Inserted 5 into the min-heap: 5 7 11 10
Inserted 4 into the min-heap: 4 5 11 10 7
Inserted 13 into the min-heap: 4 5 11 10 7 13
```

Time Complexity: **O(log(n))** (where n is no of elements in the heap)
Auxiliary Space: O(n)

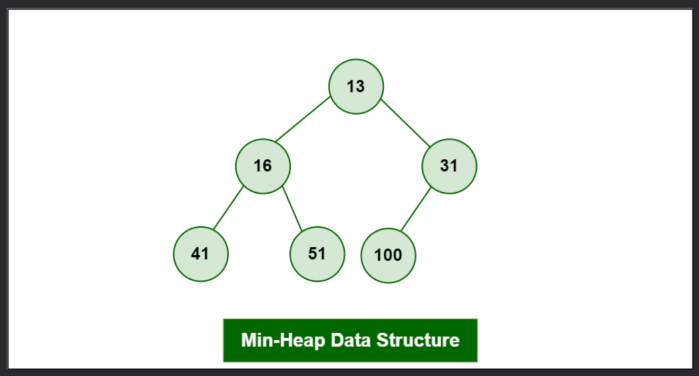
2. Deletion in Min-Heap Data Structure:

Removing the smallest element (the root) from the min heap. The root is replaced by the last element in the heap, and then the heap property is restored by swapping the new root with its smallest child until the parent is smaller than both children or until the new root reaches a leaf node.

- Replace the root or element to be deleted with the last element.
- Delete the last element from the Heap.
- Since the last element is now placed at the position of the root node. So, it may not follow the heap property. Therefore, heapify the last node placed at the position of the root.

Illustration:

Suppose the Heap is a Min-Heap as:



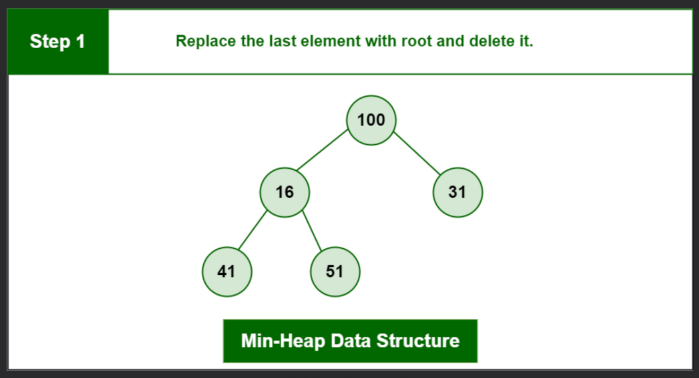
Min-Heap Data Structure

The element to be deleted is root, i.e. 13.

Process:

The last element is 100.

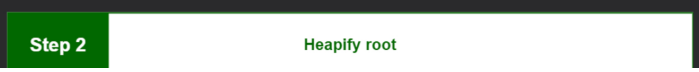
Step 1: Replace the last element with root, and delete it.

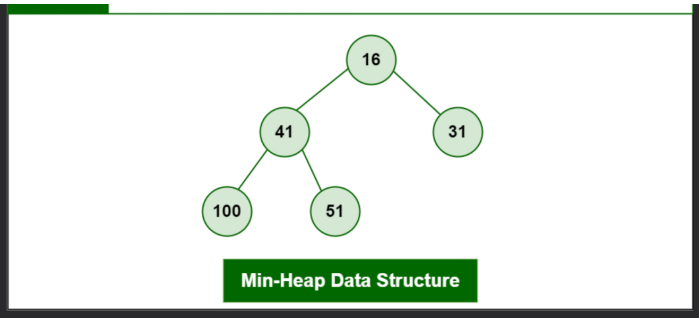


Min-Heap Data Structure

Step 2: Heapify root.

Final Heap:





Min-Heap Data Structure

Implementation of Deletion operation in Min-Heap:

C++ Java Python3 C# Javascript

```cpp
#include <iostream>
#include <vector>

using namespace std;

// Function to insert a new element into the min-heap
void insert_min_heap(vector<int>& heap, int value)
{
    // Add the new element to the end of the heap
    heap.push_back(value);
    // Get the index of the last element
    int index = heap.size() - 1;
    // Compare the new element with its parent and swap if
    // necessary
    while (index > 0
           && heap[(index - 1) / 2] > heap[index]) {
        swap(heap[index], heap[(index - 1) / 2]);
        // Move up the tree to the parent of the current
        // element
        index = (index - 1) / 2;
    }
}

// Function to delete a node from the min-heap
void delete_min_heap(vector<int>& heap, int value)
{
    // Find the index of the element to be deleted
    int index = -1;
    for (int i = 0; i < heap.size(); i++) {
        if (heap[i] == value) {
            index = i;
            break;
        }
    }
    // If the element is not found, return
    if (index == -1) {
        return;
    }
    // Replace the element to be deleted with the last
    // element
    heap[index] = heap[heap.size() - 1];
    // Remove the last element
    heap.pop_back();
    // Heapify the tree starting from the element at the
    // deleted index
    while (true) {
        int left_child = 2 * index + 1;
        int right_child = 2 * index + 2;
        int smallest = index;
        if (left_child < heap.size()
            && heap[left_child] < heap[smallest]) {
            smallest = left_child;
        }
        if (right_child < heap.size()
            && heap[right_child] < heap[smallest]) {
            smallest = right_child;
        }
        if (smallest != index) {
            swap(heap[index], heap[smallest]);
            index = smallest;
        }
        else {
            break;
        }
    }
}

// Main function to test the insert_min_heap and
// delete_min_heap functions
int main()
```

```
{
    vector<int> heap;
    int values[] = { 13, 16, 31, 41, 51, 100 };
    int n = sizeof(values) / sizeof(values[0]);
    for (int i = 0; i < n; i++) {
        insert_min_heap(heap, values[i]);
    }
    cout << "Initial heap: ";
    for (int j = 0; j < heap.size(); j++) {
        cout << heap[j] << " ";
    }
    cout << endl;

    delete_min_heap(heap, 13);
    cout << "Heap after deleting 13: ";
    for (int j = 0; j < heap.size(); j++) {
        cout << heap[j] << " ";
    }
    cout << endl;

    return 0;
}
```

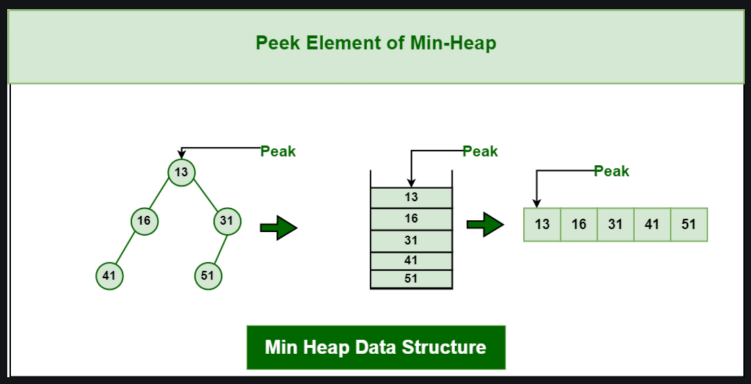
Output

```
Initial heap: 13 16 31 41 51 100
Heap after deleting 13: 16 41 31 100 51
```

Time complexity: O(log n) where n is no of elements in the heap
Auxiliary Space: O(n)

3. Peek operation on Min-Heap Data Structure:

To access the minimum element (i.e., the root of the heap), the value of the root node is returned. The time complexity of peek in a min-heap is O(1).



Min Heap Data Structure

Implementation of Peek operation in Min-Heap:

C++ Java Python3 C# Javascript

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

int main()
{
    // Create a max heap with some elements using a
    // priority_queue
    priority_queue<int, vector<int>, greater<int> > minHeap;
    minHeap.push(9);
    minHeap.push(8);
    minHeap.push(7);
    minHeap.push(6);
    minHeap.push(5);
    minHeap.push(4);
    minHeap.push(3);
    minHeap.push(2);
    minHeap.push(1);

    // Get the peek element (i.e., the largest element)
```

```
    int peakElement = minHeap.top();

    // Print the peak element
    cout << "Peak element: " << peakElement << std::endl;

    return 0;
}
```

Output

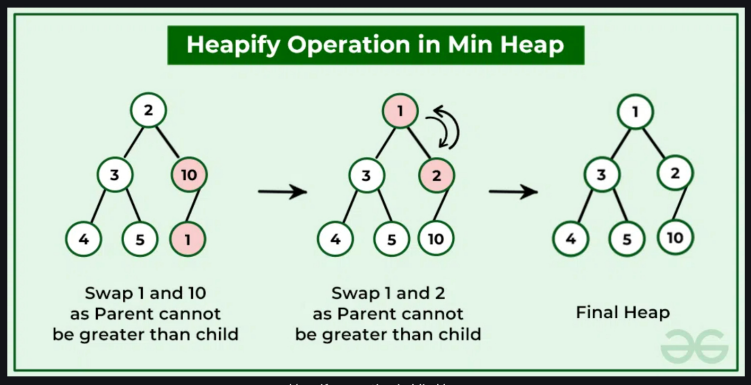
```
Peak element: 1
```

Time complexity: In a min heap implemented using an array or a list, the peak element can be accessed in constant time, O(1), as it is always located at the root of the heap.
In a min heap implemented using a binary tree, the peak element can also be accessed in O(1) time, as it is always located at the root of the tree.

Auxiliary Space: O(n)

4. Heapify operation on Min-Heap Data Structure:

A heapify operation can be used to create a min heap from an unsorted array. This is done by starting at the last non-leaf node and repeatedly performing the "bubble down" operation until all nodes satisfy the heap property.



Heapify operation in Min Heap

Implementation of Heapify operation in Min-Heap:

C++ Java Python C#

```
#include <iostream>
#include <vector>
using namespace std;

void minHeapify(vector<int> &arr, int i, int n) {
    int smallest = i;
    int l = 2*i + 1;
    int r = 2*i + 2;

    if (l < n && arr[l] < arr[smallest])
        smallest = l;

    if (r < n && arr[r] < arr[smallest])
        smallest = r;

    if (smallest != i) {
        swap(arr[i], arr[smallest]);
        minHeapify(arr, smallest, n);
    }
}

int main() {
    vector<int> arr = {10, 5, 15, 2, 20, 30};

    cout << "Original array: ";
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";

    // Perform heapify operation on min-heap
    for (int i = arr.size()/2 - 1; i >= 0; i--)
        minHeapify(arr, i, arr.size());
```

```
    cout << "\nMin-Heap after heapify operation: ";
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";

    return 0;
}
```

Output

```
Original array: 10 5 15 2 20 30
Min-Heap after heapify operation: 2 5 15 10 20 30
```

The time complexity of heapify in a min-heap is O(n).

5. Search operation on Min-Heap Data Structure:

To search for an element in the min heap, a linear search can be performed over the array that represents the heap. However, the time complexity of a linear search is O(n), which is not efficient. Therefore, searching is not a commonly used operation in a min heap.

Here's an example code that shows how to search for an element in a min heap using **std::find()**:

C++ Java Python3 C# Javascript

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    priority_queue<int, vector<int>, greater<int> >
        min_heap;
    // example max heap

    min_heap.push(10);
    min_heap.push(9);
    min_heap.push(8);
    min_heap.push(6);
    min_heap.push(4);

    int element = 6; // element to search for
    bool found = false;

    // Copy the min heap to a temporary queue and search for
    // the element
    std::priority_queue<int, vector<int>, greater<int> >
        temp = min_heap;
    while (!temp.empty()) {
        if (temp.top() == element) {
            found = true;
            break;
        }
        temp.pop();
    }

    if (found) {
        std::cout << "Element found in the min heap."
                  << std::endl;
    }
    else {
        std::cout << "Element not found in the min heap."
                  << std::endl;
    }

    return 0;
}
```

Output

```
Element found in the min heap.
```

complexity analysis:

The **time complexity** of this program is **O(n log n)**, where **n** is the number of elements in the priority queue.

The insertion operation has a time complexity of **O(log n)** in the worst case because the heap property needs to be maintained. The search operation involves copying the priority queue to a temporary queue and then traversing the temporary queue, which takes **O(n log n)** time in the worst case because each element needs to be copied and popped from the queue, and the priority queue needs to be rebuilt for each operation.

The **space complexity** of the program is **O(n)** because it stores **n** elements in the priority queue and creates a temporary queue with **n** elements.

Applications of Min-Heap Data Structure:

- **Heap sort:** Min heap is used as a key component in heap sort algorithm which is an efficient sorting algorithm with a time complexity of O(nlogn).
- **Priority Queue:** A priority queue can be implemented using a min heap data structure where the element with the minimum value is always at the root.
- **Dijkstra's algorithm:** In Dijkstra's algorithm, a min heap is used to store the vertices of the graph with the minimum distance from the starting vertex. The vertex with the minimum distance is always at the root of the heap.
- **Huffman coding:** In Huffman coding, a min heap is used to implement a priority queue to build an optimal prefix code for a given set of characters.
- **Merge K sorted arrays:** Given K sorted arrays, we can merge them into a single sorted array efficiently using a min heap data structure.

Advantages of Min-heap Data Structure:

- **Efficient insertion and deletion**: Min heap allows fast insertion and deletion of elements with a time complexity of O(log n), where n is the number of elements in the heap.
- **Efficient retrieval of minimum element:** The minimum element in a min heap is always at the root of the heap, which can be retrieved in O(1) time.
- **Space efficient:** Min heap is a compact data structure that can be implemented using an array or a binary tree, which makes it space efficient.
- **Sorting:** Min heap can be used to implement an efficient sorting algorithm such as heap sort with a time complexity of O(n log n).
- **Priority Queue:** Min heap can be used to implement a priority queue, where the element with the minimum priority can be retrieved efficiently in O(1) time.
- **Versatility:** Min heap has several applications in computer science, including graph algorithms, data compression, and database systems.

Overall, min heap is a useful and versatile data structure that offers efficient operations, space efficiency, and has several applications in computer science.

Last Updated : 21 Sep, 2023

Previous
K-ary Heap

Next
Finding Minimum travel time to Nearest city from Current location

Difference between Binary Heap, Binomial Heap and Fibonacci Heap

Difference between Min Heap and Max Heap

Convert Min Heap to Max Heap

Heap Sort for decreasing order using min heap

When building a Heap, is the structure of Heap unique?

Applications of Heap Data Structure

Top 50 Problems on Heap Data Structure asked in SDE Interviews

Stack Vs Heap Data Structure

Heap data structure implementation in swift

Leaf starting point in a Binary Heap data structure

Complete Tutorials

DSA Crash Course | Revision Checklist with Interview Guide

Learn Data Structures and Algorithms | DSA Tutorial

Mathematical and Geometric Algorithms - Data Structure and Algorithm Tutorials

Learn Data Structures with Javascript | DSA Tutorial

Introduction to Max-Heap – Data Structure and Algorithm Tutorials

Article Contributed By :

itsadityash

itsadityash

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : RishabhPrabhu, itsadityash, thebeginner01, vikas2gqb5, mishraaabcf9, pratyusujhr

Article Tags : min-heap, Data Structures, DSA, Heap

Practice Tags : Data Structures, Heap

Improve Article | Report Issue

Grokking Modern System Design

NCERT Solutions

- Class 12
- Class 11
- Class 10
- Class 9
- Class 8
- Complete Study Material

School Subjects

- Mathematics
- Physics
- Chemistry
- Biology
- Social Science
- English Grammar

Commerce

- Accountancy
- Business Studies
- Indian Economics
- Macroeconomics
- Microeconimics
- Statistics for Economics

Management & Finance

- Management
- HR Managament
- Income Tax
- Finance
- Economics

UPSC Study Material

- Polity Notes
- Geography Notes
- History Notes
- Science and Technology Notes
- Economy Notes
- Ethics Notes
- Previous Year Papers

SSC/ BANKING

- SSC CGL Syllabus
- SBI PO Syllabus
- SBI Clerk Syllabus
- IBPS PO Syllabus
- IBPS Clerk Syllabus
- SSC CGL Practice Papers

Colleges

- Indian Colleges Admission & Campus Experiences
- Top Engineering Colleges
- Top BCA Colleges
- Top MBA Colleges
- Top Architecture College
- Choose College For Graduation

Companies

- IT Companies
- Software Development Companies
- Artificial Intelligence(AI) Companies
- CyberSecurity Companies
- Service Based Companies
- Product Based Companies
- PSUs for CS Engineers

Preparation Corner

- Company Wise Preparation
- Preparation for SDE
- Experienced Interviews
- Internship Interviews
- Competitive Programming
- Aptitude Preparation
- Puzzles

Exams

- JEE Mains
- JEE Advanced
- GATE CS
- NEET
- UGC NET
- CAT

More Tutorials

- Software Development
- Software Testing
- Product Management
- SAP
- SEO
- Linux
- Excel

Write & Earn

- Write an Article
- Improve an Article
- Pick Topics to Write
- Share your Experiences
- Internships