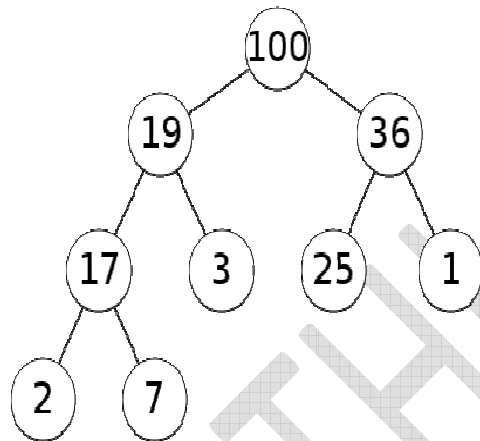


Heap (data structure)

Example of a complete binary max-heap

100
 19 36
 17 3 25 1
 2 7



2
 7
 3
 25
 1
 17
 19
 36
 100

2

" 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Heap Sort

```

1  void swap(int a, int b) {
2     int temp = a;
3     a = b;
4     b = temp;
5 }
6
7  void heapify(int arr[], int n, int i) {
8     int largest = i;
9     int left = 2 * i + 1;
10    int right = 2 * i + 2;
11
12    if (left < n & arr[left] > arr[largest])
13        largest = left;
14
15    if (right < n & arr[right] > arr[largest])
16        largest = right;
17
18    if (largest != i) {
19        swap(arr[i], arr[largest]);
20        heapify(arr, n, largest);
21    }
22 }
23
24 void heapSort(int arr[], int n) {
25     buildHeap(arr, n);
26     for (int i = n - 1; i > 0; i--) {
27         swap(arr[0], arr[i]);
28         heapify(arr, i, 0);
29     }
30 }
31
32 void buildHeap(int arr[], int n) {
33     for (int i = n / 2 - 1; i >= 0; i--)
34         heapify(arr, n, i);
35 }
36
37 int main() {
38     int arr[] = {12, 11, 13, 5, 6, 7};
39     int n = arr.length;
40     heapSort(arr, n);
41     for (int i = 0; i < n; i++)
42         cout << arr[i] << " ";
43     return 0;
44 }
  
```

Heap Operations

- * + arr[0]
- , arr[0]
- , arr[0]
- arr[0]
- , arr[0]
- # arr[0]
- arr[0]
- arr[0]
- arr[0]
- arr[0]

2

"2024 is a year of change and growth. We are excited to see the progress we have made in the past year and look forward to a successful 2024. Our team is committed to delivering high-quality products and services to our customers. We will continue to invest in research and development to stay at the forefront of our industry. Thank you for your support and loyalty. We are confident that 2024 will be a year of great achievements for our company."

Our company is committed to providing the best possible experience for our customers. We will continue to work hard to improve our products and services. We are grateful for the trust and support of our customers and partners. We are looking forward to a bright future for our company and the industry as a whole.

2

2. Recursion and Recursion Tree

& Recursion is a programming technique where a function calls itself. It is used to solve problems that can be broken down into smaller, similar sub-problems. The base case is the condition that stops the recursion. The recursive case is the part of the function that calls itself. Recursion is often used for tasks like traversing data structures, solving puzzles, and calculating mathematical functions.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

$$\leq \left[2^{(\lg n - h) - 1} \right] = \left[\frac{2^{\lg n}}{2^{h+1}} \right] = \left[\frac{n}{2^{h+1}} \right]$$

Recursion tree for calculating Fibonacci(5). The root node is 5, and it has children 4 and 3. Node 4 has children 3 and 2, and node 3 has children 2 and 1. The tree structure is as follows:

$$\sum_{h=0}^{\lceil \lg n \rceil} \frac{n}{2^{h+1}} O(h) = O\left(n \sum_{h=0}^{\lceil \lg n \rceil} \frac{h}{2^{h+1}}\right) \leq O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O(n)$$

2

"Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures."

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

2. Recursion and Recursion Tree

Recursion is a programming technique where a function calls itself. It is used to solve problems that can be broken down into smaller, similar sub-problems. The base case is the condition that stops the recursion. The recursive case is the part of the function that calls itself. Recursion is often used for tasks like traversing data structures, solving puzzles, and calculating mathematical functions.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

2

1. Recursion and Recursion Tree

Recursion is a programming technique where a function calls itself. It is used to solve problems that can be broken down into smaller, similar sub-problems. The base case is the condition that stops the recursion. The recursive case is the part of the function that calls itself. Recursion is often used for tasks like traversing data structures, solving puzzles, and calculating mathematical functions.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

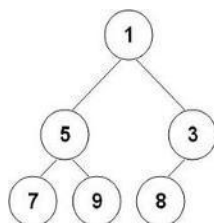
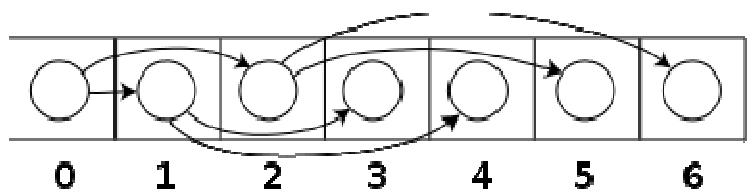
Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

Recursion is a powerful tool for solving complex problems. It allows us to write code that is more concise and easier to understand. However, recursion can be inefficient and may lead to stack overflow errors if not used carefully. It is important to understand the limitations of recursion and to use it appropriately. Recursion is a key concept in computer science and is essential for understanding many algorithms and data structures.

2



Node	1	5	3	7	9	8
Index	0	1	2	3	4	5

2

2 siblings of node i

& 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

2

0 siblings of node i = 2 * (number of nodes in level l - 1) - 1

(number of nodes in level l - 1) * 2

(number of nodes in level l - 1) * 2

2

& 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

(number of nodes in level l - 1) * 2

(number of nodes in level l - 1) * 2

2

" 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

2

+ 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

" 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

2

2 siblings of node i

4 siblings of node i = 2 * (number of nodes in level l - 1) - 1

" 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

. 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

. 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

" 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

. 2 siblings of node i = 2 * (number of nodes in level l - 1) - 1

= Number of nodes including i - Number of nodes through the previous level - One node for i itself

= (i + 1) - (2^l - 1) - 1

= i + 1 - 2^l + 1 - 1

= i - 2^l + 1

' 2 siblings of node i = 2 * (i - 2^l + 1)

2 siblings of node i = 2 * (i - 2^l + 1)

2

So, total siblings to right of is:-

= Total nodes in level l - (Total siblings on left + 1)

= (2^l) - (i - 2^l + 2)

= 2^l + 2^l - i - 2

= 2^{l+1} - i - 2

2

2

So, index of 1st child of node would be:-

$$\begin{aligned}
&= i + \text{Total siblings on right} + 2 * \text{Total siblings on left} + 1 \\
&= i + (2^{l+1} - i - 2) + 2(i - 2^l + 1) + 1 \\
&= i + 2^{l+1} - i - 2 + 2i - 2^{l+1} + 2 + 1 \\
&= i - i + 2i + 2^{l+1} - 2^{l+1} - 2 + 2 + 1 \\
&= 2i + 1
\end{aligned}$$

2

???? -?? ?????????????????

Binary Heap

A Binary Heap is a Binary Tree with following properties.

- 1) It's a complete tree (All levels are completely filled except possibly the last level and the last level has all keys as left as possible). This property of Binary Heap makes them suitable to be stored in an array.
- 2) A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to MinHeap.

How is Binary Heap represented?

A Binary Heap is a Complete Binary Tree. A binary heap is typically represented as an array.

- The root element will be at Arr[0].
- Below table shows indexes of other nodes for the ith node, i.e., Arr[i]:

Arr[(i-1)/2]	Returns the parent node
Arr[(2*i)+1]	Returns the left child node
Arr[(2*i)+2]	Returns the right child node

The traversal method use to achieve Array representation is **Level Order**

2

2

2

2

2

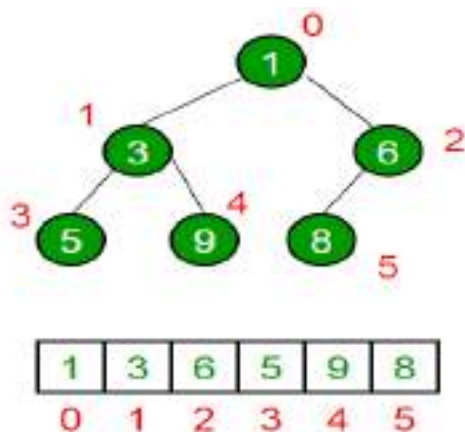
2

2

2

2

2





Applications of Heaps:

- 1) **Heap Sort:** Heap Sort uses Binary Heap to sort an array in $O(n \log n)$ time.
- 2) **Priority Queue:** Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() and extractmax(), decreaseKey() operations in $O(\log n)$ time. Binomial Heap and Fibonacci Heap are variations of Binary Heap. These variations perform union also efficiently.
- 3) **Graph Algorithms:** The priority queues are especially used in Graph Algorithms like **Dijkstra's Shortest Path** and **Prim's Minimum Spanning Tree**.
- 4) Many problems can be efficiently solved using Heaps. See following for example.
 - a) **K'th Largest Element in an array.**
 - b) **Sort an almost sorted array/**
 - c) **Merge K Sorted Arrays.**

Operations on Min Heap:

- 1) **getMini():** It returns the root element of Min Heap. Time Complexity of this operation is $O(1)$.
- 2) **extractMin():** Removes the minimum element from MinHeap. Time Complexity of this Operation is $O(\log n)$ as this operation needs to maintain the heap property (by calling heapify()) after removing root.
- 3) **decreaseKey():** Decreases value of key. The time complexity of this operation is $O(\log n)$. If the decreases key value of a node is greater than the parent of the node, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.
- 4) **insert():** Inserting a new key takes $O(\log n)$ time. We add a new key at the end of the tree. IF new key is greater than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.
- 5) **delete():** Deleting a key also takes $O(\log n)$ time. We replace the key to be deleted with minus infinite by calling decreaseKey(). After decreaseKey(), the minus infinite value must reach root, so we call extractMin() to remove the key.

Below is the implementation of basic heap operations.

Python Code

```

class MinHeap:
    def __init__(self):
        self.heap = []

    def insert(self, key):
        self.heap.append(key)
        self._bubble_up(len(self.heap) - 1)

    def _bubble_up(self, index):
        while index > 0:
            parent = (index - 1) // 2
            if self.heap[index] < self.heap[parent]:
                self.heap[index], self.heap[parent] = self.heap[parent], self.heap[index]
                index = parent
            else:
                break

    def extract_min(self):
        if not self.heap:
            return None
        root = self.heap[0]
        self.heap[0] = self.heap[-1]
        self.heap.pop()
        self._bubble_down(0)
        return root

    def _bubble_down(self, index):
        left = 2 * index + 1
        right = 2 * index + 2
        smallest = index
        if left < len(self.heap) and self.heap[left] < self.heap[smallest]:
            smallest = left
        if right < len(self.heap) and self.heap[right] < self.heap[smallest]:
            smallest = right
        if smallest != index:
            self.heap[index], self.heap[smallest] = self.heap[smallest], self.heap[index]
            self._bubble_down(smallest)
    
```



৩

৯৯

৩

৭৭৩ ৩৩৩৩৩৩৩৩৩৩ ৩! ৩৩ ৩৩ ৩৩ ৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩৩ ৩ ৩৩৩৩৩৩৩

৩৩৩৩ ৩' ৩৩৩)) ৩৩৩৩৩৩: ৩৩৩৩৩

৯৯

৩ ৩৩৩৩৩৩৩ D৩৩৩৩; ৩-৩৩৩

৩ ৩ ৩৩৩৩৩৩৩ " D: &P৩৩৩

৩ ৩৩৩৩৩৩৩ D৩৩৩৩; ৩ ৩৩৩

৩ ৯৯

৩ ৩ ৩৩৩৩ D৩৩৩ ৩৩

৩ ৩ ৩৩৩৩৩৩ ৩৩৩৩৩-৩৩৩

৩ ৯৯

৩

৭৭৩ ৩৩৩৩৩৩৩৩ ৩৩ ৩৩ ৩ ৩৩৩৩৩৩৩৩৩৩ ৩! ৩৩৩৩৩৩৩ ৩৩৩৩৩৩

৩ ৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩-৩৩৩

৩ ৩৩৩৩৩৩-৩৩ ৩৩৩৩৩৩৩৩৩ D৩৩৩৩ 5৩৩৩

৩ ৩৩৩৩ D৩৩৩ ৩৩

৩ : ৩' ৩৩৩৩৩৩-৩৩৩

৩

৩ ৩৩৩৩৩৩ ৩৩৩৩৩

৯৯

৩

৩

৭৭৩ ৩ ৩৩৩৩৩৩৩৩৩৩ ৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩ ৩ ৩৩৩৩৩৩ ৩৩৩৩৩

৭৭৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩

! ৩৩৩৩ ৩' ৩৩৩)) ৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩

৯৯

৩ ৩৩৩৩৩৩৩৩ E ৩৩৩৩৩৩ " D: ৩ ৩৩৩

৩ ৩৩৩৩৩৩৩ ৩৩৩৩৩

৯৯

৩

৭৭৩& ৩৩৩৩৩৩৩ ৩৩ ৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩ ৩৩৩৩ ৩৩৩৩ ৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩

৭৭৩ ৩ ৩৩৩ ৩৩৩৩৩ ৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩

! ৩৩৩৩ ৩' ৩৩৩)): ৩' ৩৩৩৩৩৩৩৩৩৩৩

৯৯

৩ ৩৩৩৩, ৩৩৩৩৩৩৩৩

৩ ৩৩৩৩৩, ৩৩৩৩৩৩৩৩৩৩

৩ ৩৩৩৩৩ ৩৩৩৩৩৩, ৩৩৩৩

৩ ৩৩৩৩৩(৩৩৩৩৩ D৩৩৩৩৩ ৩ ৩৩৩৩৩৩৩(৩৩৩৩৩৩৩৩৩৩)

৩ ৩ ৩৩ ৩৩৩৩৩৩ ৩৩৩৩

৩ ৩৩৩৩৩(৩৩৩৩৩ D৩৩৩৩৩ ৩ ৩৩৩৩৩৩৩(৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩

৩ ৩ ৩৩ ৩৩৩৩৩৩, ৩৩৩৩

৩ ৩৩৩৩৩ ৩৩৩৩৩৩, ৩৩৩৩

৩ ৯৯

৩ ৩ ৩৩ ৩৩৩ ৩৩৩৩৩৩ ৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩

৩ ৩ : ৩' ৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩

৩ ৯৯

৯৯

৩

Problem Solution

5. **Heap Sort** is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

* **Time Complexity** of Heap Sort is $O(n \log n)$ in the worst case.

+ **Space Complexity** of Heap Sort is $O(1)$ as it is an in-place sorting algorithm.

| **Stability** of Heap Sort is not stable as it does not preserve the relative order of equal elements.

Heap Sort is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

Heap Sort is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

Heap Sort is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

Heap Sort is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

Heap Sort is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

5. **Heap Sort** is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

55. **Heap Sort** is a sorting algorithm based on the binary heap data structure. It is a comparison-based sorting algorithm. In each iteration, the root node (the maximum element) is swapped with the last element of the heap, and then the new root node is bubbled down to its proper position. This process is repeated until the entire array is sorted.

Program/Source Code

Here is the source code of a Python program to implement a d-ary heap. The program output is shown below.

```

def d_ary_heap_sort(arr, d):
    """
    Sorts an array using a d-ary heap.
    """
    n = len(arr)
    # Build the d-ary heap
    for i in range((n - 1) // d, -1, -1):
        d_ary_heapify(arr, d, i)
    # Extract elements one by one
    for i in range(n - 1, 0, -1):
        arr[0], arr[i] = arr[i], arr[0]
        d_ary_heapify(arr, d, 0)
    return arr

def d_ary_heapify(arr, d, i):
    """
    Maintains the d-ary heap property for node i.
    """
    n = len(arr)
    # Find the index of the largest child
    largest = i
    for j in range(1, d):
        child_index = i * d + j
        if child_index < n and arr[child_index] > arr[largest]:
            largest = child_index
    # If the largest child is not the current node, swap and recurse
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        d_ary_heapify(arr, d, largest)

```


- ୧ ଘର ଘରର ଘରର ଘରର ଘରର ଘର X 008
- ୧ ଘର ଘରର ଘରର ଘରର ଘରର ଘର ୧-8
- ୧ ଘର ଘରର ଘରର ଘରର ଘରର ଘର

୧୩

୧
 ୭୭୧ ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର
 ଘର ଘର ଘର ଘର ଘର, ' ଘର ଘର ଘର ଘର ଘର ୧୫ ଘର ଘର ଘର ଘର * ଘର
 ଘର

- ୧ ଘର ୧୫ ; ଘର X 008
- ୧ ଘର ଘର ଘର ଘର * 8
- ୧ ଘର * ଘର ; ଘର X 008
- ୧ ଘର ଘର ଘର ଘର ୫8
- ୧

୧ ୭୭୧ ଘର ଘର ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର ଘର ଘର ଘର ଘର X 008

- ୧ ୭୭୧ ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର
- ୧ ୭୭୧ ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର
- ୧ ଘର ୧୫ ଘର ଘର ଘର ଘର ଘର ; ଘର * ଘର ଘର ଘର ଘର ଘର
- ୧ ଘର ଘର ଘର ଘର ୫8
- ୧

୧ ଘର ଘର ଘର ଘର ୧୫ ଘର ଘର ଘର ଘର * ଘର ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର ଘର * 8

- ୧ ୭୭୧ ଘର ! ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର
- ୧ ଘର ଘର ଘର ୧୫ ; ଘର X 008 ଘର * ୧ ; ଘର X 008
- ୧ ଘର

୧ ଘର ଘର ଘର ଘର ଘର ଘର ଘର ୧୫ ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର ଘର ଘର ଘର ଘର * ଘର ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର ୧୫ ଘର ୧୫ ଘର ଘର ଘର ୧୫

- ୧ ଘର ୭୭୧ ଘର ୧୫ ଘର ଘର * ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର ଘର
- ୧ ଘର ଘର ଘର ଘର ୧୫ ଘର ଘର ଘର ; ଘର * ଘର ଘର ଘର ଘର
- ୧ ଘର

୧ ଘର ଘର ଘର ଘର ଘର ଘର ଘର ୧୫ ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର ଘର ୧୫ ଘର ଘର ଘର ଘର ; ଘର * ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର ଘର ୧୫ ଘର ଘର ୧୫

୧୪

୧
 ୭୭୧ ଘର * ଘର ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର
 ୧ ଘର
 ୧ ଘର
 ୧ ଘର ଘର ଘର ୧୫ ଘର * ଘର ଘର ଘର ଘର
 ୧ ଘର ଘର ଘର * ଘର ଘର ଘର ଘର ୫8

77. `int main() { int x = 10; printf("%d", x); }`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `X 008`

77. `int main() { int x = 10; printf("%d", x); }`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `' int main() { int x = 10; printf("%d", x); }`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `' int main() { int x = 10; printf("%d", x); }`

Output:

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output: `10`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `int main() { int x = 10; printf("%d", x); }`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output: `10`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output: `10` ; `int main() { int x = 10; printf("%d", x); }`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output:

77. `int main() { int x = 10; printf("%d", x); }`

Output:

Python Program to Implement Binomial Heap

Problem Solution

5. `int main() { int x = 10; printf("%d", x); }`

* `int main() { int x = 10; printf("%d", x); }`

Program Explanation

1. Create an instance of BinomialHeap.
2. The user is presented with a menu to perform various operations on the heap.
3. The corresponding methods are called to perform each operation.

Runtime Test Cases

```

:?????
2??-?
???????(???)?
??????
????? ????
3-??
5????+!-????!-???,???!??!6??????<?
5????+!-????!-???,???!??!6??????>?
5????+!-????!-???,???!??!6??????=?
5????+!-????!-???,???!??!6??????=?
5????+!-????!-???,???!??!6??????=?
2????-??0??-????
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????<?
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????=?
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????>?
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????!??
5????+!-????!-???,???!??!6?3-??
??
:?????
2??-?
???????(???)?
??????
????? ????
3-??
5????+!-????!-???,???!??!6??????
5????+!-????!-???,???!??!6???????
5????+!-????!-???,???!??!6?????;?
5????+!-????!-???,???!??!6?????C?
5????+!-????!-???,???!??!6??????
2????-??0??-??;?
5????+!-????!-???,???!??!6??????<?
5????+!-????!-???,???!??!6??????
2????-??0??-??<?
5????+!-????!-???,???!??!6??????B?
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????<?
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????;?
5????+!-????!-???,???!??!6??????
5????+!-????!-???,???!??!6????? ????
2????-??0??-????!0????
5????+!-????!-???,???!??!6?3-??

```


?

?

?

?

?

?

(?)?

- ?
- ?

. ? / ?

-) ?

1 ? ? ? ? ?



?

?

?

?

?

?

- ?

(? / ?

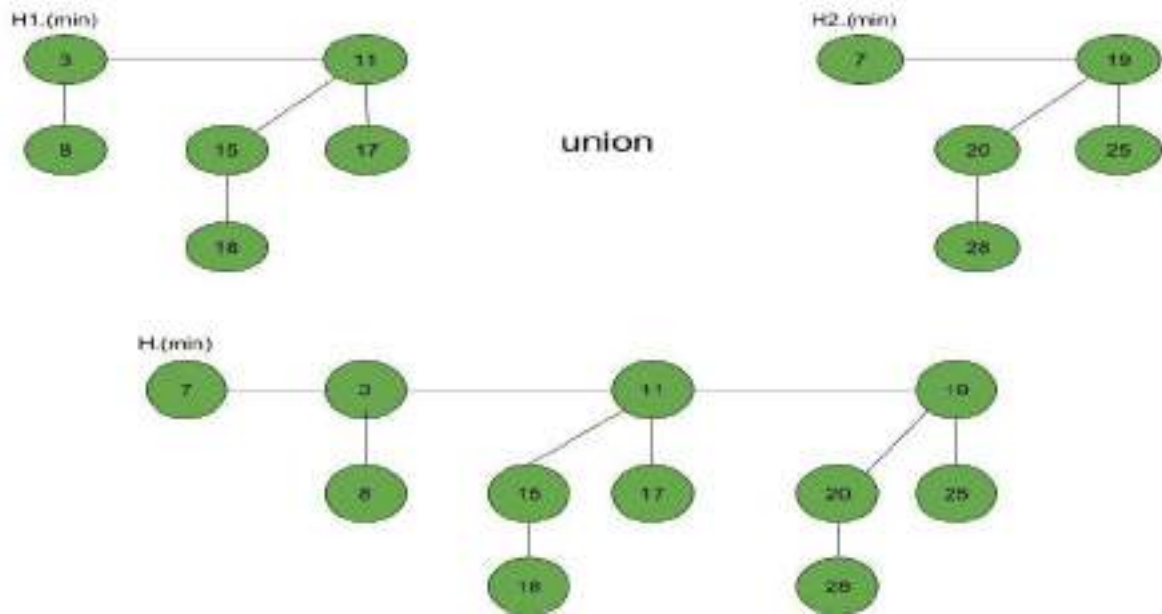
- ?

?

Example:

?

2



2

773->>କାଉଁର ପତ୍ରର କାଉଁର ପତ୍ରର ପତ୍ରର ପତ୍ରର କାଉଁର କାଉଁର

77କାଉଁର କାଉଁର ପତ୍ରର ପତ୍ରର କାଉଁର କାଉଁର ପତ୍ରର ପତ୍ରର କାଉଁର ପତ୍ରର

Cକାଉଁର ପତ୍ରର କାଉଁର କାଉଁର

Cକାଉଁର ପତ୍ରର କାଉଁର କାଉଁର କାଉଁର Aକାଉଁର

Cକାଉଁର ପତ୍ରର କାଉଁର ପତ୍ରର କାଉଁର Aକାଉଁର

ପତ୍ରର କାଉଁର ପତ୍ରର କାଉଁର ପତ୍ରର କାଉଁର

2

ପତ୍ରର ପତ୍ରର ପତ୍ରର ପତ୍ରର

2 ପତ୍ରର 6କାଉଁର ପତ୍ରର 8କାଉଁର

2 ପତ୍ରର 6କାଉଁର କାଉଁର 8କାଉଁର

2 ପତ୍ରର 6କାଉଁର 8କାଉଁର

2 ପତ୍ରର 6କାଉଁର 8କାଉଁର

2 କାଉଁର 8କାଉଁର

Nକାଉଁର

2

773-ପତ୍ରର କାଉଁର କାଉଁର କାଉଁର ପତ୍ରର କାଉଁର

ପତ୍ରର ପତ୍ରର ପତ୍ରର 6କାଉଁର କାଉଁର କାଉଁର X 008କାଉଁର

2

77# ପତ୍ରର କାଉଁର କାଉଁର ପତ୍ରର କାଉଁର ପତ୍ରର କାଉଁର ପତ୍ରର କାଉଁର କାଉଁର କାଉଁର

କାଉଁର Dକାଉଁର Dକାଉଁର ପତ୍ରର 8କାଉଁର

2

774-ପତ୍ରର କାଉଁର କାଉଁର ପତ୍ରର କାଉଁର କାଉଁର କାଉଁର କାଉଁର

! ପତ୍ରର କାଉଁର ପତ୍ରର କାଉଁର ପତ୍ରର କାଉଁର

Lକାଉଁର

2 ପତ୍ରର ପତ୍ରର ପତ୍ରର 6କାଉଁର Dକାଉଁର, କାଉଁର ପତ୍ରର ପତ୍ରର 6କାଉଁର ପତ୍ରର ପତ୍ରର ପତ୍ରର ପତ୍ରର 8କାଉଁର

2 ପତ୍ରର Dକାଉଁର Aକାଉଁର 8କାଉଁର

2 ପତ୍ରର Dକାଉଁର Aକାଉଁର କାଉଁର X 008କାଉଁର

2 ପତ୍ରର Dକାଉଁର Aକାଉଁର କାଉଁର X 008କାଉଁର

2 ପତ୍ରର Dକାଉଁର Aକାଉଁର କାଉଁର Dକାଉଁର 8କାଉଁର

2 ପତ୍ରର Dକାଉଁର Aକାଉଁର କାଉଁର Dକାଉଁର 8କାଉଁର

2 କାଉଁର କାଉଁର, କାଉଁର X 008କାଉଁର

2 2 ପତ୍ରର Aକାଉଁର Aକାଉଁର କାଉଁର Dକାଉଁର 8କାଉଁର

2

- Q
- Q [Fibonacci Heap Deletion](#)
- Q [Fibonacci Heap Decrease Key](#)
- Q
- Q [Fibonacci Heap Extract Min](#)
- Q
- Q [Fibonacci Heap Insert](#)
- Q
- Q [Fibonacci Heap Merge](#)
- Q
- Q [Fibonacci Heap Operations](#)
- Q

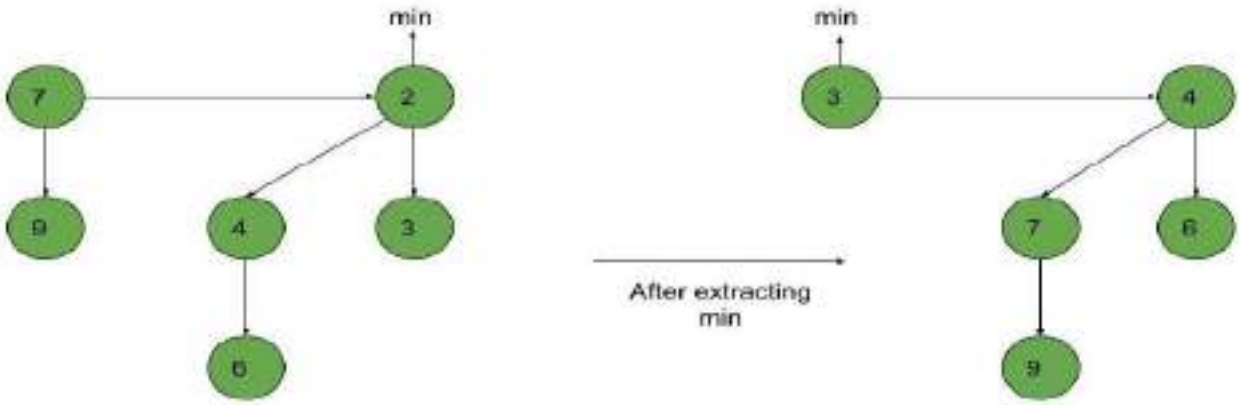
Fibonacci Heap – Deletion, Extract min and Decrease key

1. In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.

- Q # Fibonacci Heap Deletion
- * Q . In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- + Q < In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- | Q . In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- HQ : Q! In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- [Fibonacci Heap Deletion](#)
- [Fibonacci Heap Decrease Key](#)

Q 1. In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.

1.1.1.1 Fibonacci Heap Deletion



2. In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.

Q & A: Fibonacci Heap Deletion

- X In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- Q In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- < In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- : In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.
- & In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.

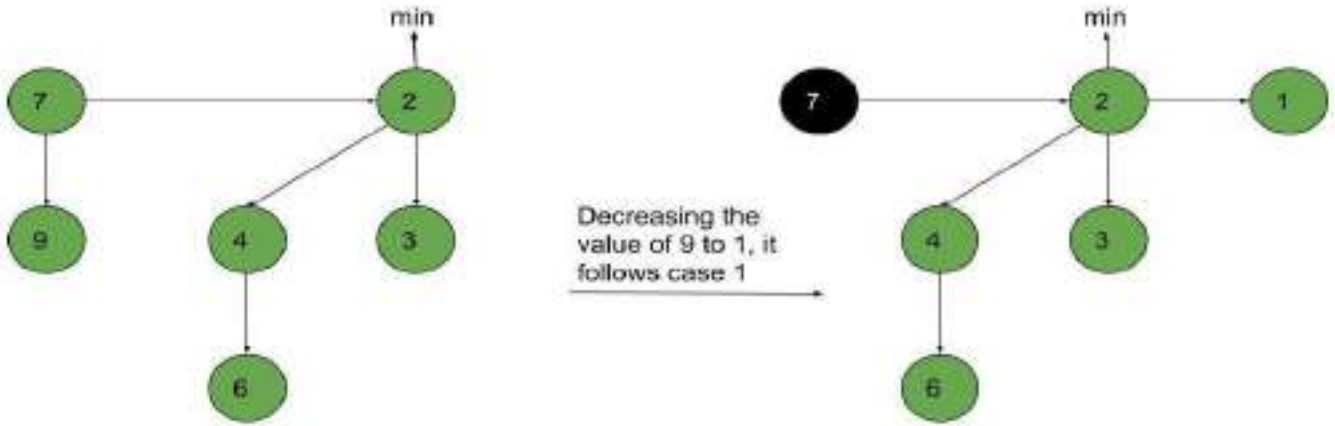
Q & A: Fibonacci Heap Decrease Key

- < In a Fibonacci heap, the root node is the minimum element. When we extract the minimum element, we need to remove the root node and merge its children into a new root node.

?

- <math>7 < 2</math> → swap 7 and 2
- <math>7 < 4</math> → swap 7 and 4
- <math>7 < 3</math> → swap 7 and 3
- 7 is now the root
-]

1.2.2.2 Case-2



?

+ [unclear text]

5. # [unclear text]

* [unclear text]

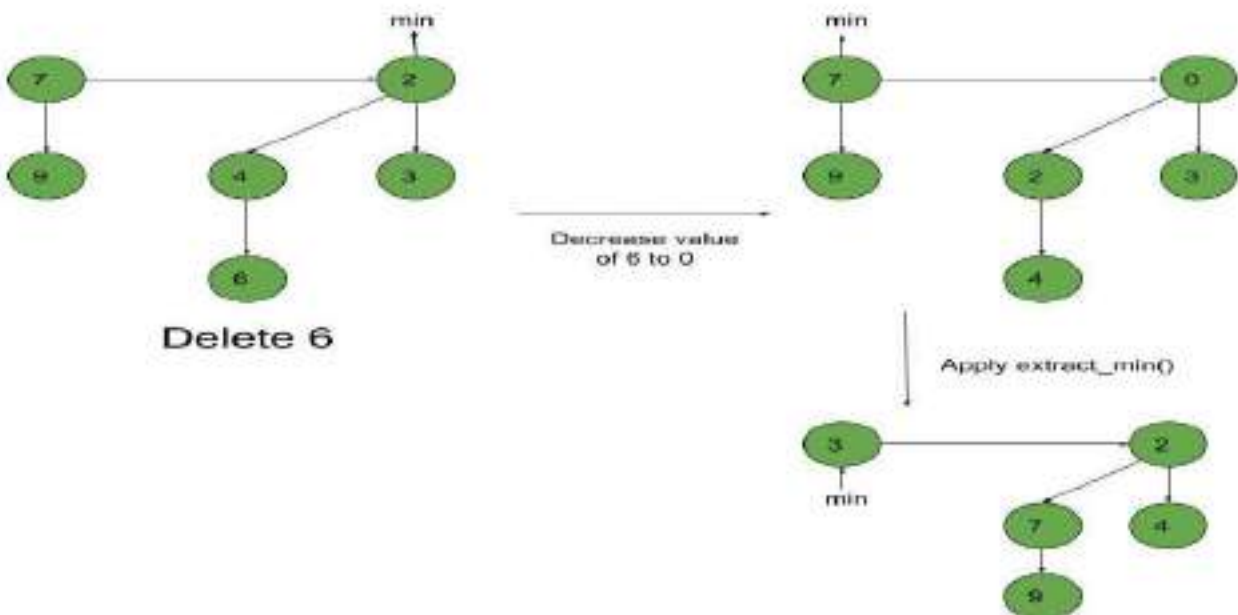
+ [unclear text]

1.2.2.2 Case-2

?

?

?



?

?

[unclear text]

[unclear text]

7. <math>7 < 2</math> → swap 7 and 2

7. [unclear text]

?

୧
୨
୩
୪
୫
୬
୭
୮
୯
୧୦
୧୧
୧୨
୧୩
୧୪
୧୫
୧୬
୧୭
୧୮
୧୯
୨୦
୨୧
୨୨
୨୩
୨୪
୨୫
୨୬
୨୭
୨୮
୨୯
୩୦
୩୧
୩୨
୩୩
୩୪
୩୫
୩୬
୩୭
୩୮
୩୯
୪୦
୪୧
୪୨
୪୩
୪୪
୪୫
୪୬
୪୭
୪୮
୪୯
୫୦

ଉପରୋକ୍ତ

୧ ଟଙ୍କା କରତ D୧୧୧୧୧୧୧୧୧

୩୩

୧୧ D୧୧D୧୧୧୧୧୧>୧୧

୩୩

୭୭୦୩୩୩

! ୩୩୩

୩୩

୩୩୩୩* A୩୩୩୩ A୩୩୩୩୩୩୩୩ ୩୩୩୩* A୩୩୩୩୩୩୩୩

୩୩୩୩* A୩୩୩୩୩୩୩ A୩୩୩୩୩୩ ୩୩୩୩* A୩୩୩୩୩୩୩୩

୩୩୩୩୩୩୩୩ A୩୩୩୩୩୩୩ ; ୩୩୩୩୩୩୩

୧ ୩୩୩୩ ୩୩୩୩୩୩୩୩

୩୩୩୩* A୩୩୩୩୩୩୩ ୩୩୩୩* ୩୩୩୩

୩୩୩୩* A୩୩୩୩୩୩୩ ୩୩୩୩* ୩୩୩୩

୩୩୩୩* A୩୩୩୩୩୩୩୩ ୩୩୩୩୩୩୩୩

୩୩୩୩୩୩୩୩ A୩୩୩୩୩୩ ; ୩ X 00୩୩୩

୧ ୩୩୩୩ A୩୩୩୩୩୩ ୩୩୩୩* ୩୩୩୩

୩୩୩୩* A୩୩୩୩୩୩୩ ୩୩୩୩୩ A୩୩୩୩୩୩୩

୩୩୩୩* A୩୩୩୩୩୩୩ ୩୩୩୩୩୩ A୩୩୩୩୩୩୩ A୩୩୩୩୩୩୩

୩୩୩୩୩୩୩ A୩୩୩୩୩ A୩୩୩୩୩୩ A୩୩୩୩୩୩୩ ୩୩୩୩* ୩୩୩୩

୩୩୩୩୩ A୩୩୩୩୩ A୩୩୩୩୩୩ ୩୩୩୩* ୩୩୩୩

୩୩୩୩୩୩୩* A୩୩୩୩୩୩୩୩୩୩ A୩୩୩୩୩ A୩୩୩୩୩୩୩

୧ ୩୩୩୩ A୩୩୩୩୩୩ ୩୩୩୩* ୩୩୩୩

୩୩୩୩ A୩୩୩୩୩୩>>୩୩

୩୩

୭୭୩-୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩

! ୩୩୩୩-୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩

୩୩

୩୩୩୩୩୩୩୩ ୩୩୩୩୩୩

୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩ ୩* ୩୩୩୩୩୩୩୩ D୩୩D୩୩୩୩୩୩୩୩୩୩୩୩୩୩* ୩୩୩୩୩୩୩

୩୩୩୩୩୩୩୩ ୩+୩୩ ୩୩୩୩୩ ୩* ୩୩୩୩

୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩ ୩+୩୩୩୩୩

୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩ ୩-୩୩୩୩୩୩୩୩; ୩୩୩୩୩ ୩+୩୩୩୩>୩୩୩

୧ ୩୩୩୩୩୩୩୩୩ ୩ X 00୩୩୩୩

୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩

୩୩୩୩୩୩୩୩୩୩* ୩୩୩୩୩

୩୩୩୩୩୩୩୩୩୩+୩୩୩୩୩

୩୩୩୩୩୩୩୩୩୩୩୩୩ ୩୩୩୩୩୩୩୩

୩୩୩୩୩୩୩

୩୩୩୩ ୩୩୩୩ ୩୩୩୩ A୩୩୩୩୩୩୩୩୩

୩୩୩୩ ୩୩୩୩୩ ୩୩୩୩୩ A୩୩୩୩୩୩୩୩୩୩୩୩

୩୩ ୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩ ୩ X 00୩୩୩୩୩୩

୩୩ ୩୩୩ ୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩ ୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩

୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩୩ A୩୩୩୩୩୩* A୩୩୩୩୩୩୩୩୩

୧

୧

୧ ଉପର ପ୍ର A ପ୍ରପରପରପ୍ର କାପର ପ୍ର A ପ୍ରପରପରପରପ୍ର ୫୫୫

୧ ଉପରପ୍ର A କାପରପ୍ର କାପରପ୍ର ୫୫୫

୧ ଉପରପ୍ର A କାପରପ୍ର କାପରପ୍ର ୫୫୫

୧ ଉପ କାପ A କାପରପ୍ର A କାପରପ୍ର କାପରପ୍ର ୫୫୫

୧ ଉପରପ୍ର A କାପରପ୍ର କା କା ୫୫୫

୧ ଉପରପ୍ର A କାପରପ୍ର କା କାପ A କାପରପ୍ର ୫୫୫

୧ ପ୍ର କାପ A କାପରପ୍ର କାପରପ୍ର ୫୫୫

୧ ଉପରପ୍ର A ପ୍ରପରପରପ୍ର କା X 00୫୫

୧ ଉପରପ୍ର A ପ୍ର ପ୍ରପ୍ରପ୍ର କା ୫୫୫

N/A

୧

77୧. ପ୍ରପ୍ରପ୍ରପ୍ର ପ୍ର କାପରପ୍ରପ୍ର ପ୍ର କାପରପ୍ରପ୍ର କାପରପ୍ରପ୍ରପ୍ର

! ପ୍ର କାପରପ୍ରପ୍ର D ପ୍ରପ୍ରପ୍ରପ୍ରପ୍ର ପ୍ରପ୍ରପ୍ର 6 କାପରପ୍ର ପ୍ରପ୍ର

L/A

୧ ପ୍ରପ୍ରପ୍ର 6 କାପର H; କାପର ପ୍ର A ପ୍ରପ୍ରପ୍ର ୫୫୫

୧ କାପରପ୍ର H; କା X 00୫୫

୧ ପ୍ର କାପରପ୍ର ପ୍ର A ପ୍ର ପ୍ରପ୍ର ; କା କାପର

୧ ପ୍ର ପ୍ର ପ୍ର ପ୍ର ପ୍ର A ପ୍ର ପ୍ରପ୍ର କା ୫୫୫

୧ ପ୍ର N/A

୧ ପ୍ର ପ୍ରପ୍ରପ୍ର

୧ ପ୍ର ପ୍ର <ପ୍ରପ୍ରପ୍ରପ୍ର ପ୍ରପ୍ରପ୍ର-୫୫୫

୧ ପ୍ର ପ୍ର <ପ୍ରପ୍ରପ୍ରପ୍ର D ପ୍ରପ୍ରପ୍ରପ୍ର-୫୫୫

୧ ପ୍ର N/A

୧ N/A

N/A

୧

77୨. ପ୍ରପ୍ରପ୍ରପ୍ର କାପର କାପରପ୍ରପ୍ର କାପରପ୍ର ପ୍ର କାପର କାପର କାପରପ୍ର କାପର କାପର

! ପ୍ର କାପ୍ର ପ୍ରପ୍ରପ୍ରପ୍ର D ପ୍ରପ୍ରପ୍ରପ୍ରପ୍ର କାପରପ୍ର 6 କାପରପ୍ର ପ୍ରପ୍ର ପ୍ର

L/A

୧ କାପର କା କା ; କା X 00୫୫

୧ ପ୍ର ପ୍ରପ୍ରପ୍ର K କା' ପ୍ରପ୍ର ପ୍ରପ୍ର କାପ୍ର ପ୍ର ପ୍ରପ୍ର 9 କା କା କା କା

୧

୧ କାପରପ୍ରପ୍ରପ୍ର ; କା X 00୫୫

୧ ପ୍ର ପ୍ରପ୍ରପ୍ର K କା ପ୍ରପ୍ର କାପରପ୍ରପ୍ର କା କାପ୍ର ପ୍ର ପ୍ରପ୍ର 9 କା କା କା କା

୧

୧ ଉପରପ୍ର A ପ୍ରପ୍ରପ୍ର ପ୍ର ୫୫୫

୧

୧ ପ୍ରପ୍ରପ୍ର କାପ୍ରପ୍ର 6 କାପ୍ର ପ୍ର କାପ୍ରପ୍ର A ପ୍ରପ୍ରପ୍ର ୫୫୫

୧ କାପରପ୍ର ପ୍ର; କା X 00୫୫୦ କାପ୍ରପ୍ର A ପ୍ରପ୍ରପ୍ର କାପ୍ର ପ୍ର A ପ୍ରପ୍ରପ୍ର

୧ ପ୍ର <ପ୍ରପ୍ରପ୍ରପ୍ର କାପ୍ର ପ୍ର ୫୫୫

୧ ପ୍ର <ପ୍ରପ୍ରପ୍ରପ୍ର D ପ୍ରପ୍ରପ୍ରପ୍ର ପ୍ର ୫୫୫

୧ N/A

୧ କାପରପ୍ରପ୍ର A ପ୍ରପ୍ରପ୍ର କା କା A ପ୍ରପ୍ରପ୍ର

୧ ପ୍ର କା କା କା କା କା କା

୧

৩

১৫৫

৩ ৩৩৩৩৬৫৫৫৫৫ ৫৫ ৫৫৫৫

৩ ৫৫৫৫৫৫৫৫ ; ৫৫ X ০০৫৫

৩ ৩ ৩৩৩৩৩৩৩৩' ৩৩৩ ৩৩৩৩৩৩ ৩ ৩৩৩৩৩৩৩৩৩৩৩

৩

৩ ৩৩৩৩৩৩

৩ ৩ ৩৩৩৩৩৩৩৩' ৩৩৩৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩ (৩৩৩৩)৩৩৩৩৩৩৩

৩ ৩ ৩৩৩৩

৩ ৩ ৩ ৩৩৩৩৩৩৩৩ A৩৩৩৩৩

৩ ৩ ৩ ৩৩৩৩ ৩৩৩ A৩৩৩৩৩

৩ ৩ ৩ ৩৩৩৩৩৩৩৩; ৩ ৩ ৩৩৩৩

৩ ৩ ৩ ৩ ৩৩৩৩৩৩৩ A9৩৩

৩ ৩ ৩ N৩

৩ ৩ N৩ ৩৩৩৩৩৩৩৩; ৩ ৩ ৩৩O৩৩৩ A৩৩৩৩৩; ৩ X ০০৩৩

৩ ৩ ৩৩৩৩৩৩৩৩৩

৩ ৩ ৩ KK৩' ৩৩৩৩৩৩ ৩৩৩৩৩৩৩ D৩D৩৩৩৩৩KK৩৩৩৩৩৩৩৩৩৩৩

৩ ৩ ৩ KK৩৩৩৩

৩ N৩

N৩

৩

77## ৩৩ ৩৩৩৩৩৩৩

৩৩৩৩ ৩৩৩৩৩

১৫৫

৩ 77/ ৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩

৩ ৩৩৩৩৩৩৩৩< ৩৩৩৩৩৩৩৩৩ ৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩

৩ ৩৩৩৩৩৩৩৩-৩৩৩

৩ ৩৩৩৩৩৩৩৩* ৩৩৩

৩ ৩৩৩৩৩৩৩৩R৩৩৩

৩

৩ 77৩৩ ৩৩ ৩৩ ৩৩৩৩ ৩৩৩ ৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩

৩ ৩৩৩৩৩৩৩৩৩

৩

৩ 77৩৩ ৩৩ ৩৩ ৩৩৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩ ৩৩ ৩ ৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩

৩ ৩৩৩৩৩৩৩৩] ৩৩৩৩৩৩৩৩ ৩৩ ৩৩৩৩৩৩৩৩

৩] ৩৩৩৩৩D৩ ৩৩৩৩

৩ ৩৩৩৩৩৩৩৩৩

৩

৩ 77৩৩ ৩৩ ৩৩ ৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩ ৩৩R৩৩৩৩ ৩৩ ৩৩

৩ ৩৩৩৩৩৩৩# ৩৩৩৩৩৩৩৩ ৩৩৩৩৩৩R৩৩৩ ৩৩ ৩৩৩৩৩৩৩৩

৩ 4৩৩৩৩ ৩৩ ৩৩R৩৩ ৩৩৩

৩ ৩৩৩৩৩৩৩৩৩

৩

৩ 77৩৩ ৩৩ ৩৩ ৩৩৩৩ ৩৩৩৩ ৩৩৩৩৩৩৩৩৩৩

৩ ৩৩৩৩৩৩৩# ৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩৩

৩

0

00

000000000000000000000000, 000000
000000000000+0000000000007! 000000400000, 00000
0000000000000000000000000000000000+00000000
00000000000000000000000000000000! 0000000, 0000 (00000000000000, 000000
00000000000000000000000000000000+0000000
000000000000000000! -000000000000! -00000000

00

000000000000000000000000
0000000000000000000000000000! 00000
000000000000000000000000! 0000
0000000000000000000000000000, 0000

00

0000000000 00000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000! 00000
00
00
00000000000000000000000000000! 000000000000000
0000000000000000000000000000000000
00000000000000000000000000000! 0000
0000000000000000000000000000000000
000000000000000000000000000000, 0000

00

0000000000! 00! 0000000000000000
0000000000- 0000000! ! 0000! 0? 00000000! -00000000000! 00000

00

0000000000000000000000000000*0000000
00000000000000 0000000000000000000
000000000000! 00000000 0! 000000
0000000000000000000000000! 0000 00
000000000000000000- 0! 000000000000000! 00000
0000000000000000000000- 0! 0000000
0000000000000000000000 0, 0000) 0000, 00000
000000000000000000000000 0000000000 0
000000000000000000000000000000000000
00000000000000000000- 0! 0000000000! 0000
000000000000000000000! 00000000! 0000000000
0000000000000000- 0! 00000000 0

00

000000000000000000000000! 0000
0000000000000, 000000- 00
0000000000000000, 0000000000! 00000
000000000000000000000000000000000000, 00
00000000000000000000000000000000! 0000
0000000000000000000000000000, 0, 0000 (00000000000000, 000000
0000000000000000000000000000000000, 0

00

00000000! 0000! 0? 0 0000
000000000000000000000000 00 0000000000

00

00

0000000000007! 00000000000000

00

0000000, 200- .00
0000000, 00000000(00000) .00
0000000, 00000000 .00
0000000, 000000 000000 .00
0000000, 3-00 .00

0

2

4. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

5. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

* $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

2. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

5. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

* $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

+ $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

3. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

5. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

* $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

+ $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

| $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

$\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

$\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

$\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

+ $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

5. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

* $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

+ $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

| $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

H $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

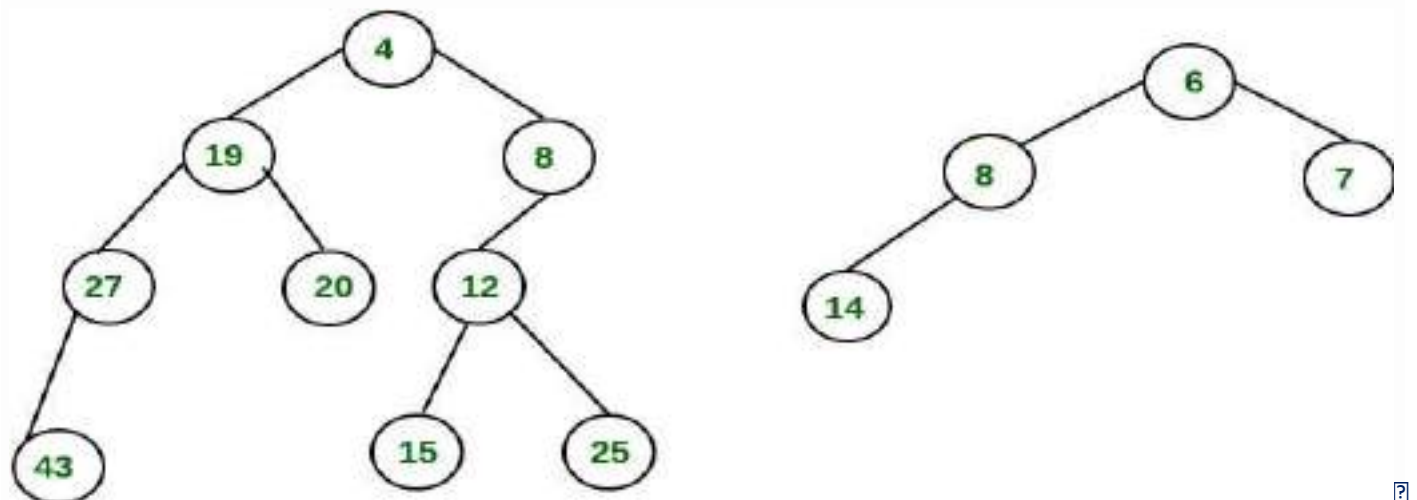
$\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

1. $\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

$\frac{1}{2} \times \frac{2}{3} = \frac{1}{3}$

2

2

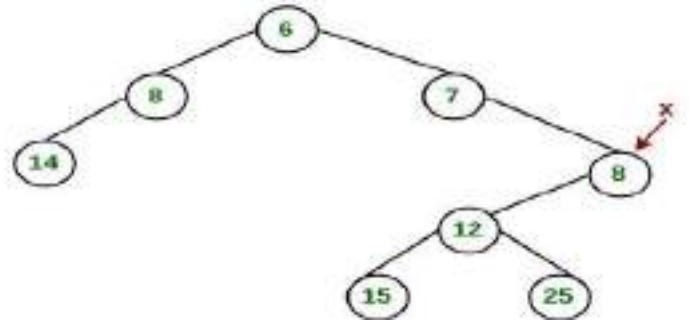
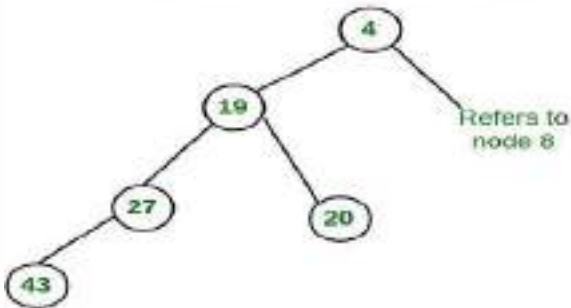


2

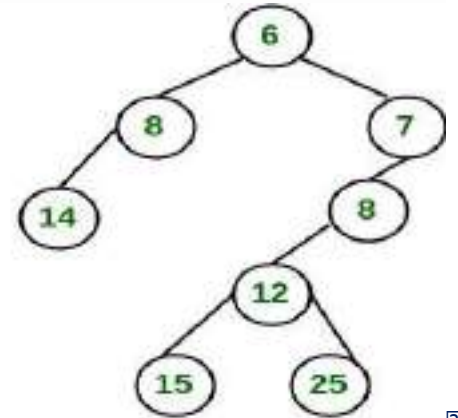
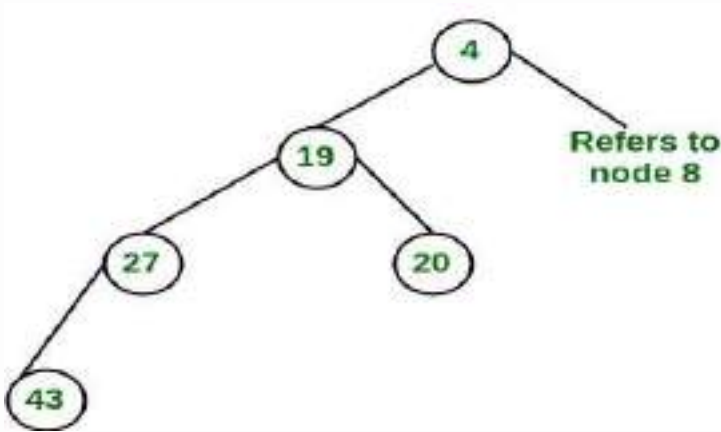
2

2

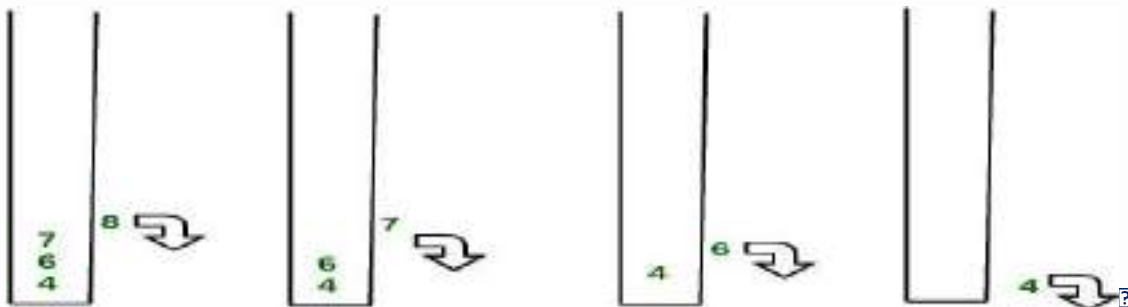
Compare(4,6)
 Push 4
 Compare(8,6)
 Push 6
 Compare(8,7)
 Push 7
 Compare(8,null)
 As null is encountered, we make node 8 as right sub-tree of stack top, i.e. 7

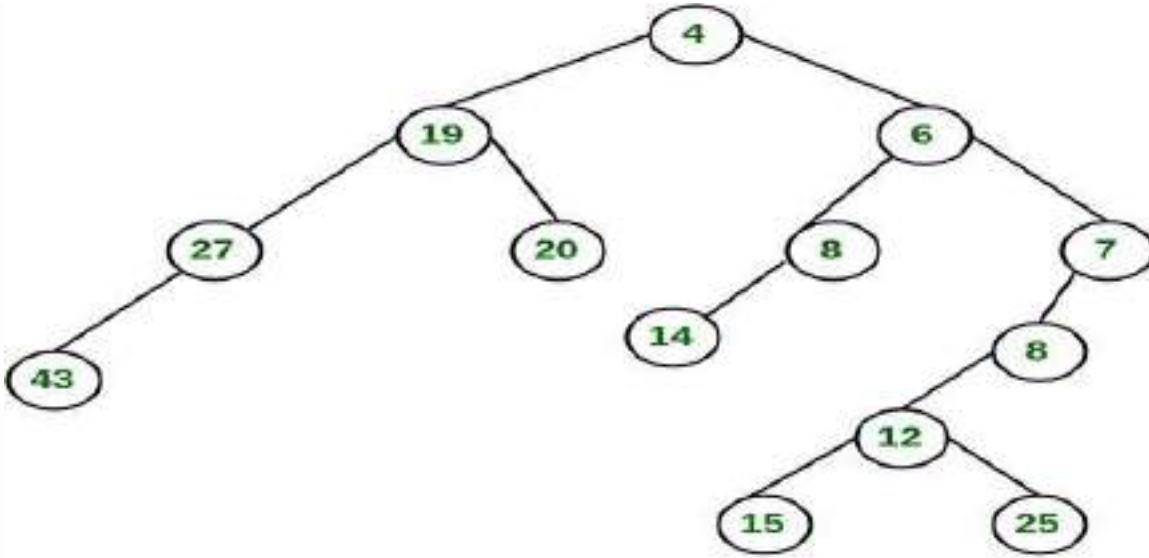


" [Placeholder text]



< [Placeholder text]





Final leftist heap

")

PARTHA SAINI

2

2 02000000 0200000000000000000000000000000000 X 00000

2 2 2 2 02000000 0200000000000000000000000000000000 X 00000000000000000000

2 1??

2 2 2000 A20000 2000000000000000000000000000000000

2 2 2000000000000000000000000000000000

2 2 2000000000000000000000000000000000

2 2 2000000000000000000000000000000000

2 1??

~~1??~~

2

77<0000000000000000000000000000000000

0000000000000000000000000000000000

1??

00000000)??

2 02000000 2000000000000000000000000000000000

2 02000000 2000000000000000000000000000000000 2000000000000000000000000000000000

2 _02000000 2000000000000000000000000000000000

2 00000000) 2 2000000000000000000000000000000000

2 0000000040000000000000000000000000

2 2000000000000000000000000000000000

2 ! 2000000000000000000000000000000000

2 ! 2000000000000000000000000000000000

2 ! 2000000000000000000000000000000000 2000000000000000000000000000000000

2 ! 2000000000000000000000000000000000

2 ! 2000 2000000000000000000000000000000000 2000000000000000000000000000000000

2 02000000 2000000000000000000000000000000000 2000000000000000000000000000000000

2000 2000)??

2 02000000 2000000000000000000000000000000000

2 02000000 20000000: 2000000000000000000000000000000000 2000000000000000000000000000000000

2 2 2 2 2 02000000 2000000000000000000000000000000000 * 2000

2 02000000 20000000: 2000000000000000000000000000000000 2000000000000000000000000000000000

2 2 2 2 2 2 02000000 2000000000000000000000000000000000 * 2000

2 ! 20000000 200<2000000000000000000000000000000000 2000000000000000000000000000000000

2 ! 2000000000000000000000000000000000 : 200 2000000000000000000000000000000000

2 02000000 2000000000000000000000000000000000 2000000000000000000000000000000000

~~1??~~

2

77<0000000000000000000000000000000000

02000000 2000)02000000 2000000000000000000000000000000000

1??

2 20000000 X 00000

~~1??~~

2

77<0000000000000000000000000000000000

2

୧

୦୧୨୩୪୫୬ ୭୮୯))୦୧୨୩୪୫୬ ୭୮୯୦୧୨୩୪୫୬ ୭୮୯୦୧୨୩୪୫୬

ୱ

୧ ୩୪୫୬୭ ୮୯ X ୦୦୧୨

୧ ୬୭୮୯୦୧ ୨୩୪୫୬୭

ୱ

୧

୭୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୦୧୨୩୪୫୬ ୭୮୯))_୦୧୨୩୪୫୬ ୭୮୯୦୧୨୩୪୫୬

ୱ

୧ ୨ ୩୪୫] ୬ ୭୮୯୦୧୨୩୪୫୬

ୱ

୧

୭୬୭ ୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୩୪୫୬୭୮୯୦୧୨୩୪୫୬ ୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୧୨୩୪୫ ୬୭୮୯୦୧୨

! ୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

ୱ

୧ ୩୪୫୬୭୮୯୦ ; ୧୦୧୨୩୪୫୬

୧ ୨ ୩୪୫୬୭୮୯୦

୧ ୩୪୫୬୭୮୯ ୧୦ ୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୧ ୩୪୫୬୭୮୯୦୧୨ ୩ X ୦୦୧୨

ୱ

୧

୭୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୧୨୩୪୫୬ ୭୮୯୦! ୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦ ୧୨ ୩୪୫୬୭୮୯୦୧୨

୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୧ ୨ ୩ ୪ ୫ ୬ ୭ ୮ ୯ ୦ ୧୨୩୪୫୬୭୮୯୦

୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬*୭୮୯୦

ୱ

୧ ୩୪୫୬୭୮୯୦ ; ୧୧ X ୦୦୧୨

୧ ୨ ୩୪୫୬୭୮୯୦*୧୨

୧ ୩୪୫୬୭*୮୯ ; ୧୨ X ୦୦୧୨

୧ ୨ ୩୪୫୬୭୮୯୦୧୨*୩୪

୧ ୩୪୫୬୭୮୯୦ A୧୨୩୪୫୬୭୮୯* A୧୨୩୪୫୬୭୮୯

୧ ୨ ୩୪୫୬୭୮୯୦ ୧୨୩୪୫୬୭୮୯*୧୨୩୪

୧ ୧୨୩୪୫

୧ ୨ ୩୪୫୬୭୮୯୦ ୧୨୩୪୫୬୭୮୯*୧୨୩୪୫୬

ୱ

୧

୭୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

&୧୨୩୪୫ ୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୧୨୩ ୪୫୬୭୮୯୦୧୨ ୩୪୫୬

୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬୭୮୯୦

୧ ୨ ୩ ୪ ୫ ୬ ୭ ୮ ୯ ୦ ୧୨୩୪୫୬୭୮୯୦

୦୧୨୩୪୫୬୭୮୯୦୧୨୩୪୫୬*୭୮୯୦

ୱ

୧

?
 ?
 ? ? ? ? ? ? ? ? ? ?
 ? * ? ? ? ?
 ?
 ? * ? ? ? ? ? ? : ? ? ? ? ? ?
 ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
 ?
 ? ? ? ? ? ? ? ? ? ?
 N
 ?

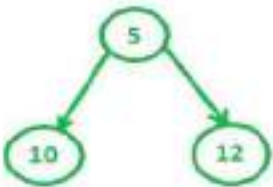
???? ? ????

A **skew heap** (or self – adjusting heap) is a heap data structure implemented as a **binary tree**. Skew heaps are advantageous because of their ability to **merge more quickly** than binary heaps. In contrast with **binary heaps**, there are no structural constraints, so there is no guarantee that the height of the tree is logarithmic. Only two conditions must be satisfied :

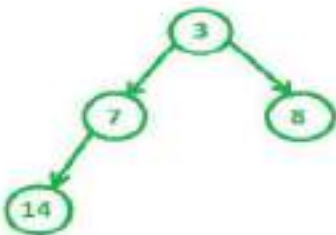
1. The general heap order must be there (root is minimum and same is recursively true for subtrees), but balanced property (all levels must be full except the last) is not required.
2. Main operation in Skew Heaps is Merge. We can implement other operations like insert, extractMin(), etc using Merge only.

Example :

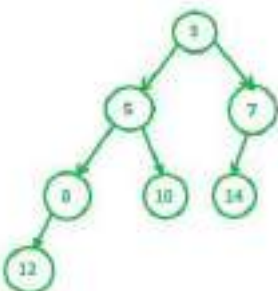
1. Consider the skew heap 1 to be



2. The second heap to be considered



4. And we obtain the final merged tree as



□

Recursive Merge Process :

merge(h1, h2)

1. Let h1 and h2 be the two min skew heaps to be merged. Let h1's root be smaller than h2's root (If not smaller, we can swap to get the same).
2. We swap h1->left and h1->right.
3. h1->left = merge(h2, h1->left)

Examples :

```

XXXXXXXXXX7XX
XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX
XXX?XXXXXXXX<XX
XXXXXXXXXXXXXXXX
=XXXXXXXXXX;XX
□
XXXXX?X7XX
XXXXXXXXXX;X
XXXXXXXXXXXXXXXX
XXX?;XXXXXXXX<;X
XXXXXXXXXX
=;XXXX;;X
□
DXXXXXXXX+XXXXXXXXXXXX)XXXXXXXXXXXX)XXXXXXXXXX+XXXXXXXX
XXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXX
XXX<XXXXXXXX?XX
XXXXXXXXXXXXXXXX
;XXXXXXXXXX=XX
□
X!+X+XXXXX-XXX0XXXX2XXXXX
XXX<XX
XXXXXXXXXXXXDXXXXXX
XX=XX
□
XXXXXXXXXX;X
XXXXXXXXXXXXXXXX
XXX?;XXXXXXXX<;X
XXXXXXXXXX
=;XXXX;;X

```

□

2

D22222222-222022222222+22222222N2222222!22222

-2222222222222222222222

22222222;2

222222222222

222<2222222?;2

2222222222222222

<;2222=2222222=;2

2

52222,222222222222222222222222222222222222!22!22222222

22222222+22222222!22!+22222222-22222

2222222222222222

22222222222222222222

22222222;22222222?22

222222222222222222222222

222<22222?;2222=22222

2222222222222222

<;2222=2222222=;2

?

77&- - 77&77& 77&77& 77&77& 77&77& 77&77& 77&77&

77&77&77&77&77&77&

C77&77&77&K77&77&77&>77A77

77&77&77&77& 77&77&77&77& 77&

2

77&77&77& 77&77&77& ' 77&77&

|77&

2 77&77&77&877&

2 . 77&77&77& ' 77&77&677&77&877&

2 . 77&77&77& ' 77&77&677&77&877&

2

2 77&77&77&77&77&77&77&77& 77&77&77&77& 77&

2 77&77&77&77&77&77&77&77&

2 . 77&77&77& ' 77&77&77&77&

2 |77&

2 2 77&77&77& 77&77&77&

2

2

2 2 [DIPLOMA] X 00000

2 2 [DIPLOMA] X 00000

2 NA

2

7 [DIPLOMA] [DIPLOMA] [DIPLOMA]

7 [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA]

7 [DIPLOMA]

2 . [DIPLOMA] ' [DIPLOMA] [DIPLOMA] [DIPLOMA] ' [DIPLOMA] [DIPLOMA] [DIPLOMA] ' [DIPLOMA] * [DIPLOMA]

2 [DIPLOMA]

2 2 [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA]

2 2 [DIPLOMA] 5 ; [DIPLOMA] X 00000

2 2 2 [DIPLOMA] * [DIPLOMA]

2 2 [DIPLOMA] * ; [DIPLOMA] X 00000

2 2 2 [DIPLOMA] [DIPLOMA] 5

2

2 2 7 [DIPLOMA] [DIPLOMA] [DIPLOMA] 5 [DIPLOMA] [DIPLOMA]

2 2 7 [DIPLOMA]

2 2 [DIPLOMA] 5 A [DIPLOMA] * A [DIPLOMA]

2 2 [DIPLOMA] [DIPLOMA] 5 [DIPLOMA] * [DIPLOMA]

2

2 2 7 [DIPLOMA] [DIPLOMA] 5 A [DIPLOMA] [DIPLOMA] 5 A [DIPLOMA]

2 2 [DIPLOMA] [DIPLOMA] 5 A [DIPLOMA] [DIPLOMA] 5 A [DIPLOMA]

2

2 2 7 [DIPLOMA] [DIPLOMA] * [DIPLOMA] [DIPLOMA] 5 A [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA]

2 2 7 [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] 5 [DIPLOMA]

2 2 [DIPLOMA] 5 A [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] * [DIPLOMA] 5 A [DIPLOMA]

2

2 2 [DIPLOMA] [DIPLOMA] 5

2 NA

2

7 [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA] [DIPLOMA]

2

2

1.??

773<222222222222 2??2222??

. 222' 222??222??222 252, ?? X 0022222 2*2 ?? X 008??

76??

2 H??

????????/??

??????/??????

??2 = ??????? * ????? 72

??2??22225?? 222 * ??-??2 = ??

76??

2 +??

????????/??

??????/??????

????? ? ???????

??/?

7??

?????? | ??????? 72

??2??2222*??2 222-??2??2??2 | ??

??2??252 ??2??2??2??252??2??2??2??2??252=??2??

??2??2*2 ??2??2??2??2*2??/??2??2??2??2*2=??2??

??2 252 ??2??2??2??2??2??2 25??2??25??25??2??

??2 2*2 ??2??2??2??2??2??2 2*??2??2*??*??2??

772 2222??2 2??2222??

??2 252 ??2??2?? 2??2??2?? 25??2?? 2*??2??

76??

2 2 +??

2 ?????????/??

2 ???????/??????

2 ?????-? ???????

2 ??/?N??????/?

2 ?????????R????2 = ?? | ??

2 ???????/?

2 ????? * ????? 72

222222K2: 222222 222(??)222K??222??

22222 ??2??2??2??2 25??2??

2

2

PARTHA SARATHI