**APS PROJECT**

# Study and implement Pairing heap, Fibonacci heap and Binomial heap and compare them

**Team Members**

**Amrit Sahai (2019201009)**
**Anvesh Bakwad (2019201030)**
**Submitted on : November 04, 2019**

## 1. INTRODUTION

### 1.1 BINOMIAL HEAP

Binomial Heap is a collection of Binomial Trees where each Binomial Tree follows min heap property. A Binomial Tree of order k has following properties.
1. It has exactly 2k nodes.
2. It has depth as k.
3. There are exactly kCi nodes at depth i for i = 0, 1, . . . , k.
4. The root has degree k and children of root are themselves Binomial Trees with order k-1,k-2,..0 from left to right.

### 1.2. FIBONACCI HEAP

Like Binomial Heap, Fibonacci Heap is a collection of trees with min-heap or max-heap property. In Fibonacci Heap, trees can can have any shape even all trees can be single nodes (This is unlike Binomial Heap where every tree has to be Binomial Tree).

### 1.3. PAIRING HEAP

**Pairing Heap is like a simplified form fibonacci heap. It also maintains the property of min heap which is parent value is less than its child nodes value. It can be considered as a self-adjusting binomial heap.**
Each node has a pointer towards the left child and left child points towards the next sibling of the child.

## 2. OPERATIONS

### 2.1. Binomial Heap

#### 2.1.1 INSERTION

Inserts a key (k) to Binomial Heap (H). This operation first creates a Binomial Heap with single key (k), then calls union on (H) and the new Binomial heap.

Structure of binomial tree node-

```
struct Node
{
    int data, degree;
    Node *child, *sibling, *parent;
};
```

Algorithm

```
1.    Binomial-Heap-Insert(H,x):
2.      H' := Make-Binomial-Heap()
3.      p[x] := NIL
4.      child[x] := NIL
5.      sibling[x] := NIL
6.      degree[x] := 0
7.      head[H'] := x
8.      H := Binomial-Heap-Union(H,H')
```

**2.1.2 RETRIEVE MINIMUM**

Algorithm

```
Binomial-Heap-Minimum(H)
1.      y := NIL
2.      x := head[H]
3.      min := infinity
4.      while x != NIL
5.              do
6.              if key[x] < min
7.                 then min := key[x]
8.                 y := x
9.                    x := sibling[x]
10.     return y
```

**2.1.3 EXTRACT MINIMUM**

Algorithm

```
function deleteMin(heap)
1.       min = heap.trees().first()
2.         for each current in heap.trees()
3.              if current.root < min.root then min = current
4.         for each tree in min.subTrees()
5.              tmp.addTree(tree)
6.         heap.removeTree(min)
7.         merge(heap, tmp)
```

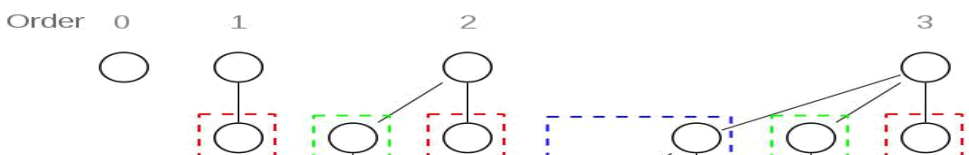


**Figure:** Binomial Heap Structure

**Figure:** Binomial Heap Node Representation

## 2.2. FIBONACCI HEAP

### 2.2.1 INSERTION

Algorithm

Create a new node ‘x’

Check whether Heap H is empty

1. If H is empty then:
   - Make x as the only node in the root list.
   - Set H(min) pointer to x.
2. Else:
   - Insert x into root list and update H(min).

#### 2.2.2 UNION

Algorithm
Join root lists of Fibonacci heaps H1 and H2 and make a single Fibonacci heap H.

If H1(min) < H2(min) then:

H(min) = H1(min)

Else:

H(min) = H2(min)

#### 2.2.3 EXTRACT-MIN

Algorithm
Delete the min node.

1. Set head to the next min node and add all the tree of the deleted node in root list.
2. Create an array of degree pointers of the size of the deleted node.
3. Set degree pointer to current node.
4. Move to the next node.
    - If degrees are different then set degree pointer to next node.
    - If degrees are same then join the Fibonacci trees by union operation.
5. Repeat steps 4 and 5 until the heap is completed.

#### 2.2.4 DELETION

Algorithm
To delete any element in a Fibonacci heap, the following algorithm is followed:

1. Decrease the value of the node to be deleted ‘x’ to minimum by Decrease_key() function.
2. By using min heap property, heapify the heap containing ‘x’, bringing ‘x’ to the root list.
3. Apply Extract_min() algorithm to the Fibonacci heap.

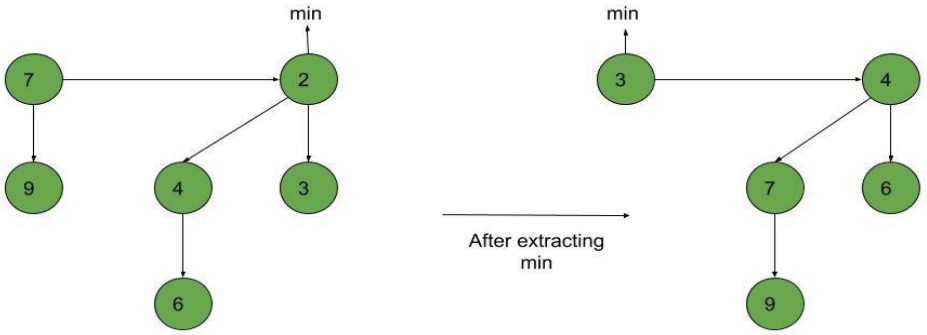



**Figure**: Extract-Min Operation

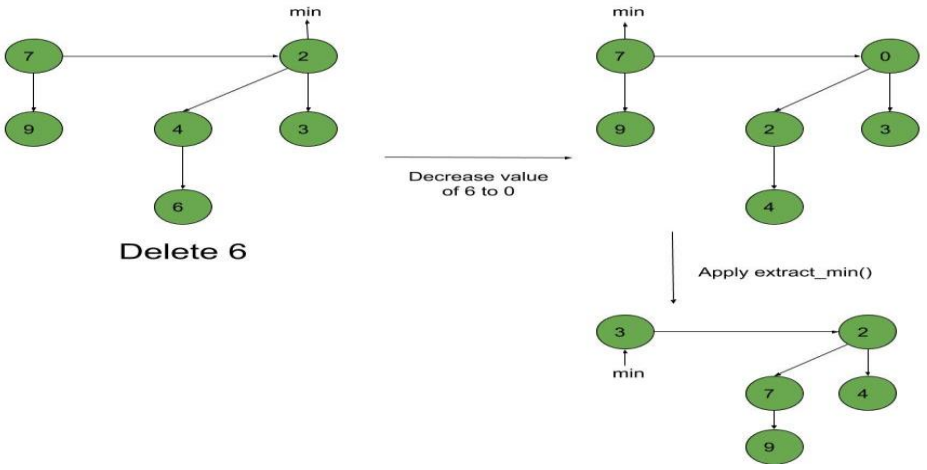



**Figure**: Delete Operation

## 2.3 PAIRING HEAP

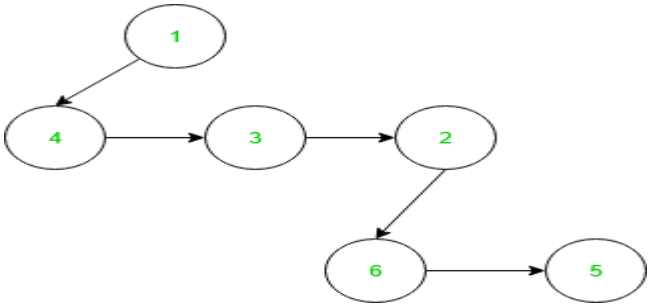



**Figure:** Pairing Heap Structural Representation

### 2.3.1 MERGE OR UNION

Algorithm

To join the two heap, first, we compare the root node of the heap if the root node of the first heap is smaller than the root node of the second heap then root node of the second heap becomes a left child of the root node of the first heap otherwise vice-versa.

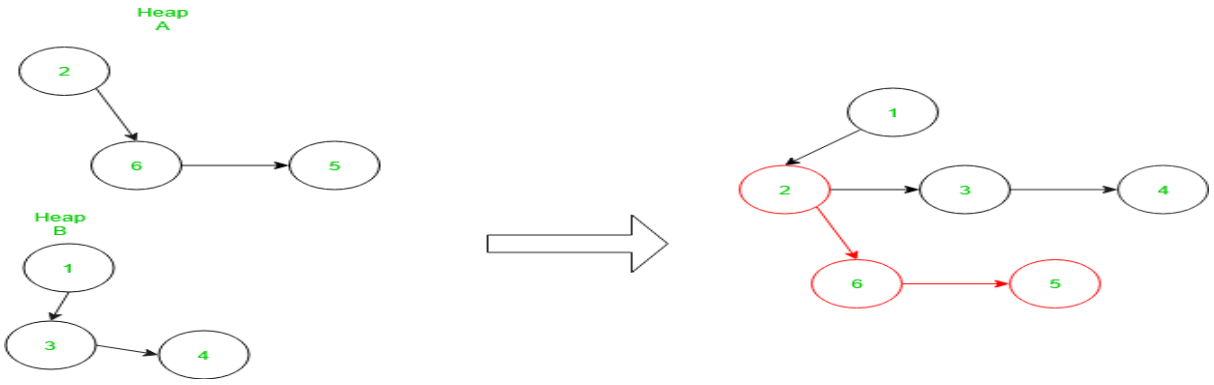



**Figure:** Merge Operation in Pairing Heap

### 2.3.2 INSERTION

Algorithm

To insert a new node in heap, create a new node and Merge it with existing heap as explained above. Therefore, the time complexity of this function is O(1).

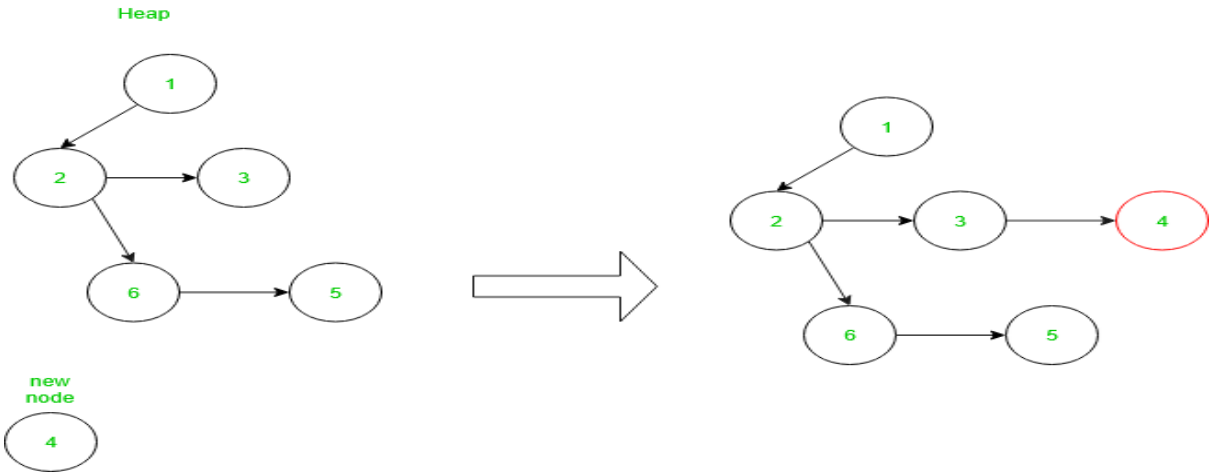



**Figure:** Pairing heap Insertion

### 2.3.4 DELETION

Algorithm

1.Deletion in Pairing Heap only happens at the root node.

2.First delete links between root, left child and all the siblings of the left child.

3.Then Merge tree subtrees that are obtained by detaching the left child and all siblings by the two way merge and delete the root node.

4.Merge the detached subtrees from left to right in one pass and then merge the \ subtrees from right to left to form the new heap without violation of conditions of min-heap.

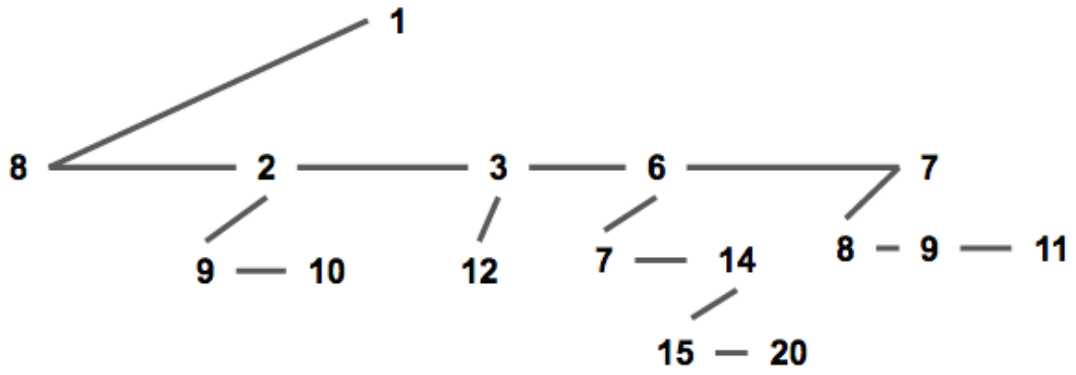



**Figure: Pairing Heap Before Deletion**

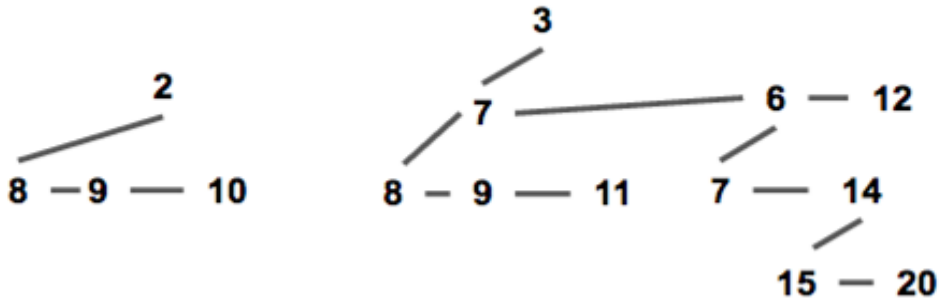



**Figure:** Pairing Heap After Deltion

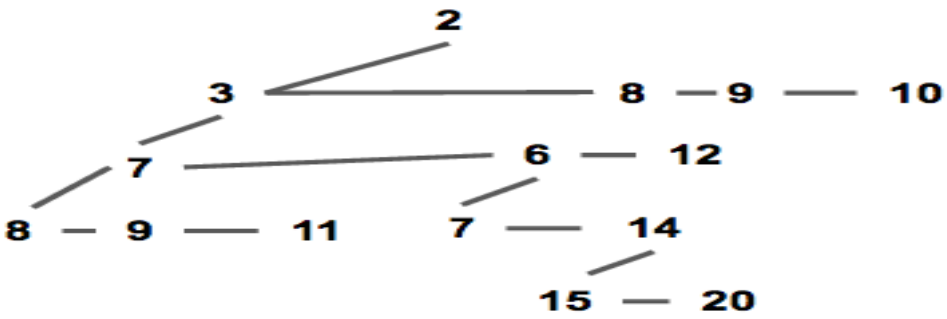


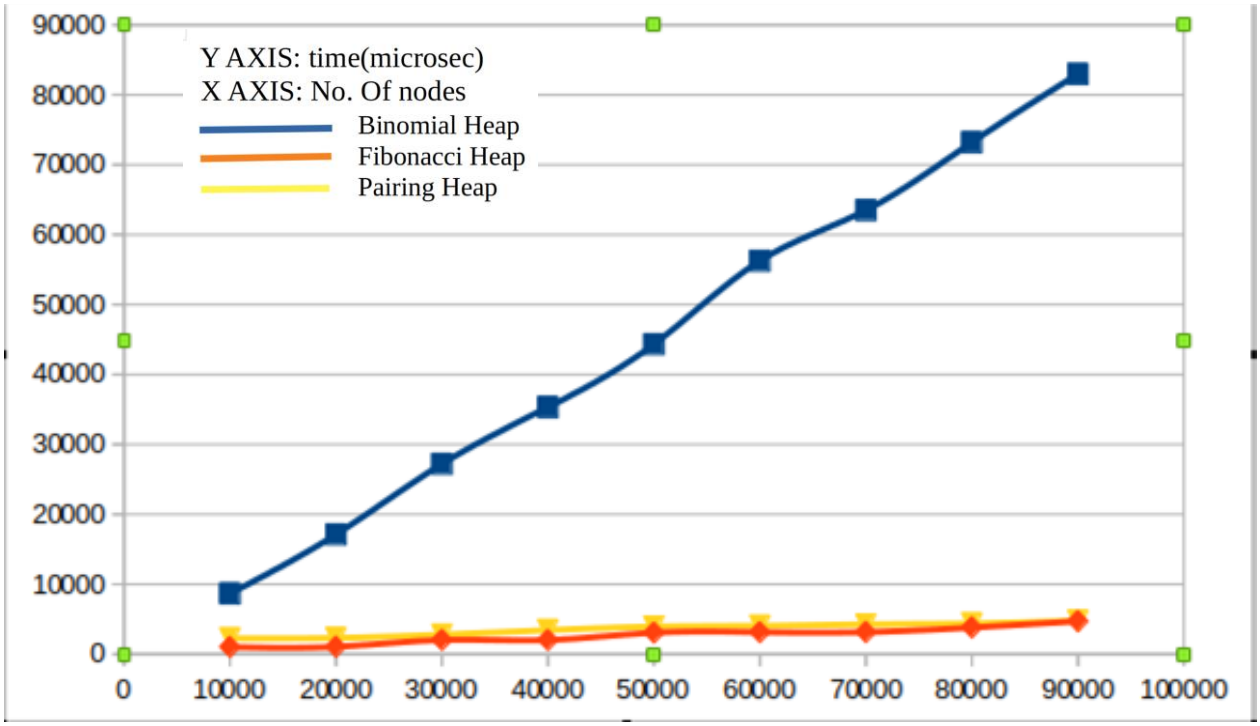
**Figure:** Pairing Heap After Final Merge

## 3. COMPARISON OF BINOMIAL, FIBONACCI AND PAIRING HEAP

| Operations | Binary Heap | Binomial Heap | Fibonacci Heap | Pairing Heap |
|---|---|---|---|---|
| make_heap | 1 | 1 | 1 | 1 |
| Insert | Log n | Log n | 1 | 1 |
| find_min | 1 | Log n | 1 | 1 |
| extract_min | Log n | Log n | Log n | Log n |
| union | Log n | Log n | 1 | 1 |
| decrease_key | Log n | Log n | 1 | Log n |
| delete | Log n | Log n | Log n | Log n |
| is_empty | 1 | 1 | 1 | 1 |

## 4.RESULTS

INSERTION




## 5. APPLICATION

Most Asyspmtotically fastest algorithms for computing minimum spanning trees and finding single source shortest path make use of fibonacci Heaps

Examples :

1. Dijkstra Single Source Shortest Path Algorithm
2. Prim's Minimum Spanning Tree

Time Complexity Of Prim's And Dijkstra's Algortihm

1.Without Fibonacci Heap : O (ElogV + VlogV)

2.Using Fibonacci Heap : O (E + VlogV)

## 6. Repositary :

https://github.com/Amrit-Sahai/APS-PROJECT.git

## 7. References :


1.https://www.wikipedia.com/en/Binomial_heap

2.https://brilliant.org/wiki/binomial-heap/

3.https://people.ksp.sk/~kuko/gnarley-trees/?page_id=310

4.https://en.wikipedia.org/wiki/Pairing_heap

5.https://en.wikipedia.org/wiki/Fibonacci_heap