EXP NO. 4

Ajay Singh
UID 2019430012

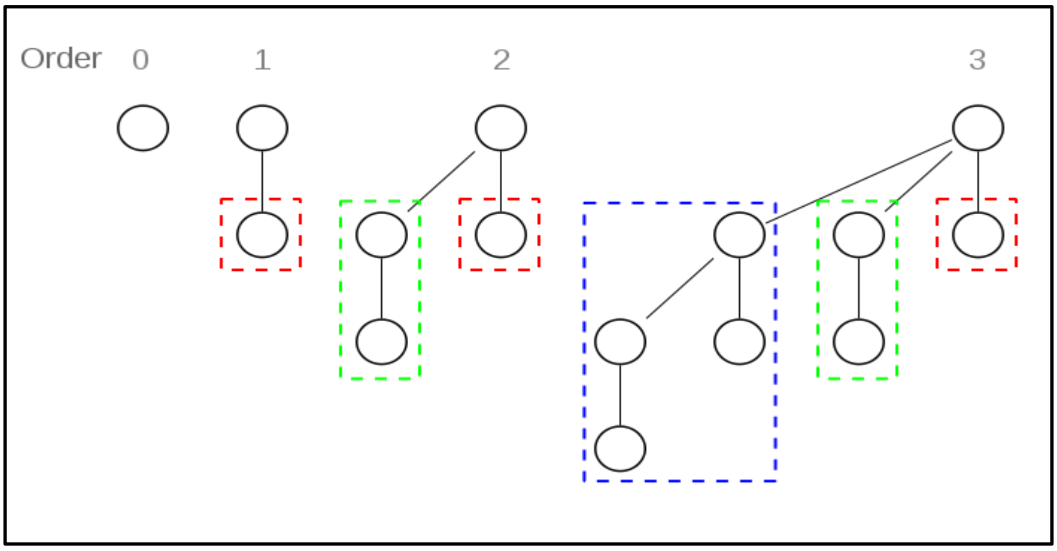
Aim: Write a program to implement & Analysis Binomial heap.

Objectives:

1. To find the feasible set's adjacent vertices for a LP Problem.
2. Calculate the optimal solution to the linear programming problem.

Definition:

A binomial heap is a specific implementation of the heap data structure. Binomial heaps are collections of binomial trees that are linked together where each tree is an ordered heap. In a binomial heap, there are either one or zero binomial trees of order k,k, where kk helps describe the number of elements a given tree can have: 2^k2k.
Binomial heaps are similar to binary heaps but binomial heaps have a more specific structure and allow for efficient merging of heaps. Heaps are often used to implement priority queues which are in turn used in implementations of many types of algorithms such as shortest-path finding algorithms—these fast operations help make these algorithms more efficient.



Properties:

Binomial Tree of order k has following properties.
1. It has exactly 2k nodes.
2. It has depth as k.
3. There are exactly ${}^{k}C_{i}$ 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.

Time Complexity:

1. Insertion

Inserting a new element to a heap can be done by simply creating a new heap containing only this element **O(1)** and then merging it with the original heap. Because of the merge, a single insertion takes O(log n) time .

2. Deletion

Deleting an element from a heap can be done in **O(log n)** time .

3. Searching

Searching an element from a heap can be done in **O(n)** time .

4. Find minimum

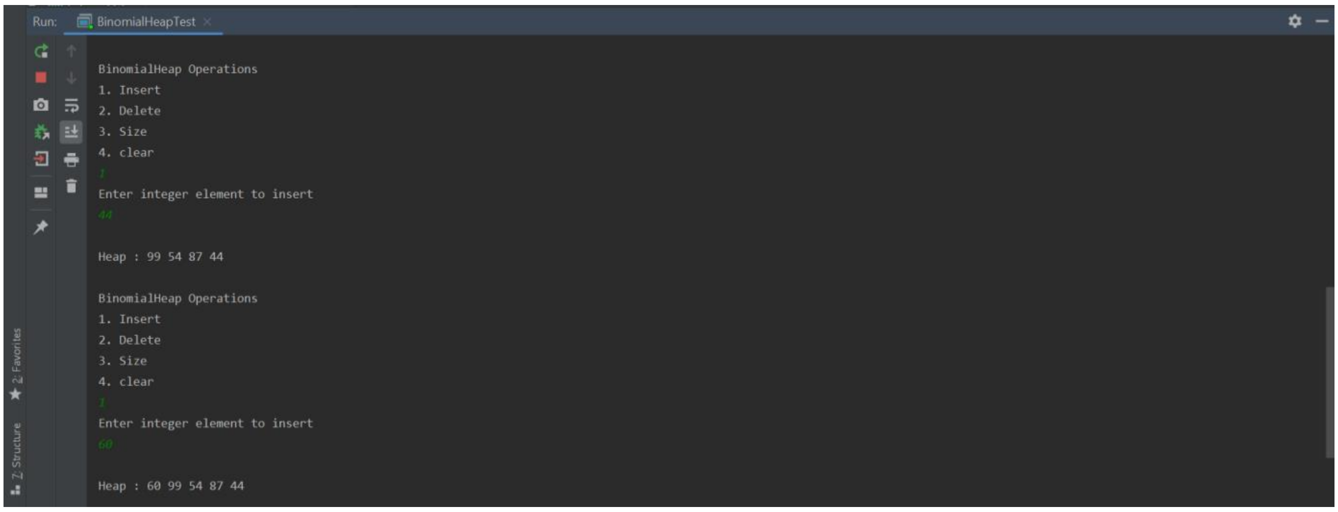
To find the **minimum** element of the heap, find the minimum among the roots of the binomial trees. This can be done in **O(log n)** time, as there are just **O(log n)** tree roots to examine.

5. Delete minimum

Since each root has at most **O(log n)** children, creating this new heap takes time **O(log n)**. Merging heapstakes time **O(log n)**, so the entire delete minimum operation takes time **O(log n)**.

Implementation:

1. Inserting element



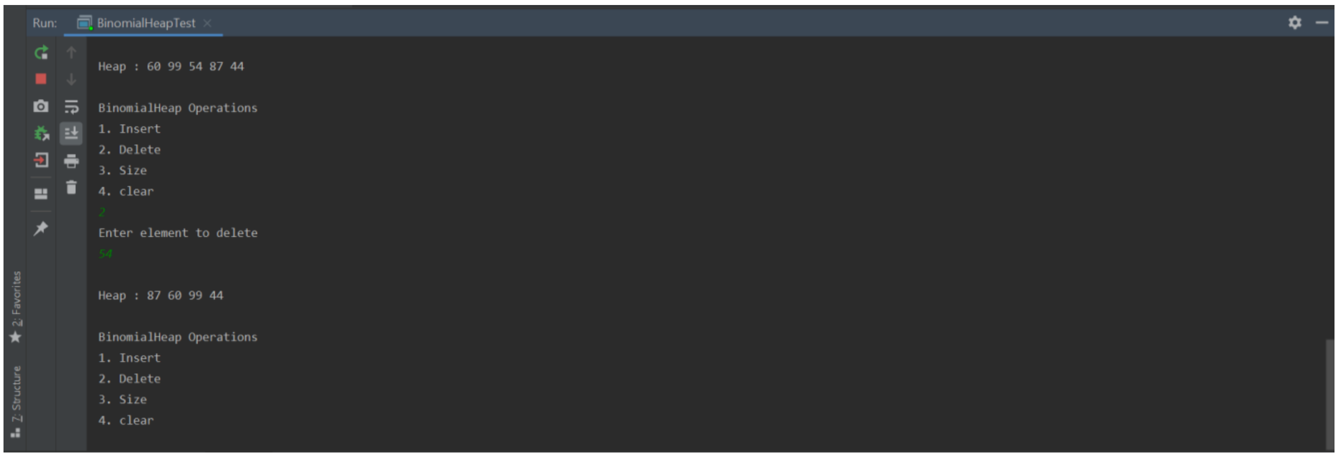
```
BinomialHeap Operations
1. Insert
2. Delete
3. Size
4. clear
1
Enter integer element to insert
44

Heap : 99 54 87 44

BinomialHeap Operations
1. Insert
2. Delete
3. Size
4. clear
1
Enter integer element to insert
60

Heap : 60 99 54 87 44
```

2. Deleting element



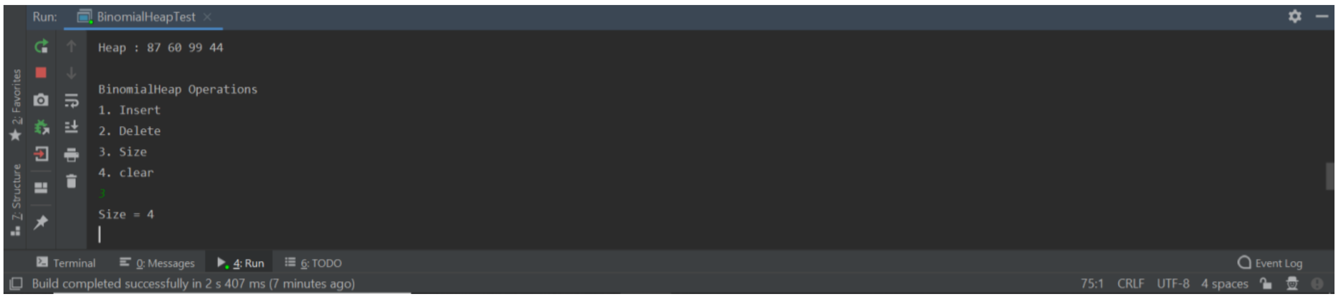
```
Heap : 60 99 54 87 44

BinomialHeap Operations
1. Insert
2. Delete
3. Size
4. clear
2
Enter element to delete
54

Heap : 87 60 99 44

BinomialHeap Operations
1. Insert
2. Delete
3. Size
4. clear
```

3. Computing Size



```
Heap : 87 60 99 44

BinomialHeap Operations
1. Insert
2. Delete
3. Size
4. clear
3
Size = 4
```

Build completed successfully in 2 s 407 ms (7 minutes ago)

Conclusion:

The aim of binomial heaps is to provide cheap merging of heaps, while keeping the cost for all the other operations low. Binomial heaps are basically an ordered list of binomial trees, which are so-called because the number of nodes at depth ii*i* follows the distribution of the binomial coefficient. Because of the way that the binomial trees are organised, there will never be more than *O*(log *n*) of them, all in heap order with the minimum element at the root.