PRANAV JAGADEESH 1BM18CS071 5B BATCH 4 PROGRAM-9

Write a program to implement the following functions on Binomial Heap:

(1) Insert(H, K): Insert 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 and return Heap

2 getMin(H): A simple way to getMin() is to traverse the list of root of Binomial Trees and return the minimum key.

3. ExtractMin(H): This operation also uses union(). We first call getMin() to find the minimum key Binomial Tree, then we remove the node and create a new Binomial Heap by using connecting all subtrees of the removed minimum node. Finally we call union() on H and the newly created Binomial Heap.

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

```
Node * newNode (int Key)
{ Node *temp ← new Node
  temp → data ← Key
  temp → degree ← 0
  temp → child ← temp → parent ← temp → sibling ← NULL
  return temp
}

list<Node *> InsertATreeInHeap (list<Node *> _heap,
                                Node * tree)
{  list<Node *> temp
   temp.push_back(tree)
   temp ← unionBinomialHeap(_heap, temp)
   return adjust(temp)
}

Node *. getMin(list<Node *> _heap)
{ list<Node *>:: iterator it ← _heap.begin()
   Node *temp ← * it
   while (it ! ↔ _heap.end())
   { if ((*it) → data < temp → data)
       temp ← * it
       it++
   }
    return temp
}
list<Node *>  extractMin (list<Node *> _heap)
{ list<Node *> new_heap, lo
   Node * temp
   temp ← getMin(_heap)
   list<Node *>:: iterator it
   it ← _heap.begin()
```

```
while (it != _heap.end())
{ if(*it != temp)
  { new_heap.push_back(*it)
  }
  it++
}
bo <- removeMinFromTreeReturnBHeap(temp)
new_heap <- unionBinomialHeap(new_heap, bo)
new_heap <- adjust(new_heap)
return new_heap
}
```