SOWMYA D.V.
1BM19CS408

Date _____ Page _____

Program to implement different operations on Binomial heap

(1) Inserting a Binomial tree into binomial heap.

```
list <Node *> insertATreeInHeap (list <Node*>
                    heap, Node * tree)
{  list <Node *> temp;
   temp.push_back(tree);
   temp = unionBinomialHeap (_heap, temp);
   return adjust (temp);
}

list <Node *> removeMinFromTreeReturnBHeap
        (Node *tree)
{  list <Node *> heap;
   Node * temp = tree -> child;
   Node * lo;
   while (temp)
   {  lo = temp;
      temp = temp -> sibling;
      lo -> sibling = NULL;
      heap.push_front (lo);
   }
   return heap;
}
list <Node *> insert (list <Node *> _head,
                    int key)
{  Node * temp = newNode (key);
   return insertATreeInHeap (_head, temp);
}
```

2) getMin

```
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;
}
```

3) extractMin

```
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++;
    }
lo = removeMinFromTreeReturnBHeap (temp);
new_heap = unionBinomialHeap (newHeap, lo);
new_heap = adjust (new_heap);
return new_heap;
}
```

```
list <Node*> unionBinomialHeap (list<Node*> l1, list<Node*> l2)
{
    list <Node*> _new;
    list<Node*>::iterator it = l1.begin();
    list<Node*>::iterator ot = l2.begin();

    while(it != l1.end() && ot != l2.end())
    {
        if(*it)->degree <= (*ot)->degree)
        {
            _new.push_back(*it);
            it++;
        }
        else
        {   _new.push_back(*ot);
            ot++;
        }
    while(it != l1.end())
        {   _new.push_back(*it);
            it++;
        }
    while(ot != l2.end())
        {
            _new.push_back(*ot);
            ot++;
        }
        return _new;
}
```