NAME : NIHA
USN : 1BM18CS060
09-12-2020

papergrid
Date: / /

LAB-10

1. Write a program to implement the Binomial Heap with
(i) Insert(H, k)
(ii) getMin (H)
(iii) extractMin(H)

```
#include <bits/stdc++.h>
using namespace std;

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;
 }

Node * mergeBinomialTrees( Node *b1, Node *b2)
{
        if ( b1->data > b2->data)
            swap (b1, b2);
        b2 -> parent = b1;
        b2 -> sibling = b1 -> child;
        b1 -> child = b2;
        b1 -> degree ++;
       return b1;
}

~~[illegible]~~ list <Node *> unionBinomialHeap (list <Node *>
                                               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;
  }

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

list <Node *> insert(list<Node *> _head, int key)
{
    Node * temp = newNode(key);
   return insertATreeInHeap (_head, temp);
}

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