Name- Nimit Sajal.
USN- 1BM18CS063.
Date → 9/12/20

ADS - lab - Binomial Heap.

```
list <Node*> insert (list <Node*> head, int key)
{
    Node * temp = newNode( key);
    return insertATreeInHeap ( head, temp);
}.
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 (heap);
}.
```

[illegible]

```
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 ++;
    }.

list <Node *> extract Min (list <Node *> heap)
{
    list <Node * > newheap, lo;
    Node * temp ;
    temp = get Min (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;
};

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 ++;
         };
    };
```

```
list <Node*> adjust (list <Node*> heap)
{
    if (heap. size()<=1)
            return heap;
    list <Node*> new heap;
    list < Node*> :: iterator it 1, it2, it3;
    ~~*~~ it1 = it2 = it3 = heap. begin();
    if (heap . size() ==2)
    {
        it2 = it1;
        it 2++;
        it3 = heap. end ();
    }.
    else
    {
        it2++;
        it 3= it 2;
        it3;
    }.
    while (it1 != heap.end())
    {
        if( it2 == heap. end ())
            it1++;
else if ((*it1)->dgree <(*it2)->dgree)
{
    it1++;
    it2++;
    if(it3 != heap.end())
        it3++;
}.
else if (it3!= heap.end() &&
        (*it1)->dgree == (*it2)->degree &&
        (*it1)->dgree ==(*it3)->dgree)
{ .
    it1++;
    it2++;
    it3++;
}.
else if ((*it1)->dgree ==(*it2)->degre).
{
    Node * temp;
    *it1 = merge BinomialTrees (*it1,*it2);
    it2 = heap.erase (it2);
    if ( it3 != heap.end())
        it3++;
}.
}.
return heap;
}.
```