NAME - ABHINAY CHINTALAPATI
UFID - 02504891
UF EMAIL - a.chintalapati@ufl.edu

REPORT

NODE STRUCTURE AND GLOBAL VARIABLES -

```
struct node
{
  int   val ;    // Value contained in the node.
  int   degree = 0; // number of children to the node.
  node* parent = NULL; // pointer to the parent of the node.
  node* child = NULL; // pointer to the child of the node.
  node* left =  NULL; // pointer to the left node in the circular linked list.
  node* right = NULL;//pointer to the right node in the circular linked list.
  bool  ccut = 0; // Childcut value of the node.
};

Struct node* HEAD = NULL;
```

FUNCTIONS -

node* createnode(int key)

Description	Creates a node with the given int value and return the pointer to the node.	
Parameters	int	The value which is to be stored in the node.
Return	Returns the pointer to the node..	

Pseudocode-

```
node*createnode(int  )
{
    Allocate memory to a new node;
    Initialize the node by assigning values;
    End;
}
```

void* insertnode(node* x)

Description	Inserts the given node into the existing heap	
Parameters	node*	The pointer to the node which is to be inserted into the heap.
Return	Returns nothing.	

Pseudocode-

```
Void insertnode(node* x )
{
    if(HEAD = NULL)
        HEAD = x;
    Else
        Link the node to x;
}
```

void pairwisemerge(unordered_map<int,node*> &dmap)

Description	Merges the nodes with same degree in the top level.	
Parameters	unordered_map<int,node*>	The address of the degree hashmap which stores the degrees of the nodes in top level
Return	Returns nothing.	

Pseudocode-

```
Void pairwisemerge(unordered_map<int,node*> &dmap
{
    Node = HEAD->right;
    Count the number of nodes in top level;
    For each top level node, find a corresponding entry in the degree hashmap and meld;
    If no entry is found, make a new entry with the current node;
}
```

void meld(node* x, node* y,unordered_map<int,node*> &dmap)

Description	Merges the given nodes which increases the degree by one and checks the hashmap for the new degree key and merges again as long as there are entries with the required degree.	
Parameters	node* node* unordered_map<int,node*>	The pointer to the first node. The pointer to the second node. The address of the degree hashmap.
Return	Returns nothing.	

Pseudocode-

```
void  meld(node* x, node* y,unordered_map<int,node*> &dmap)
{
    Erase the entry corresponding to x degree;
    Swap x and y if x < y;
    Combine x and y making x the parent;
    Check the hashmap for a entry with newdegree = degree+1;
    If found, meld it with that;
    If not found-
    Create a new entry in the hashmap with node x and degree= degree+1;

}
```

node* extractmax()

Description	Gives the maximum value of the heap	
Parameters	none	
Return	Returns the head of the heap(address of node with max value).	

Pseudocode-

```
node*extractmax(  )
{
    Make parents of all children of MAX node(HEAD) and add the children to the top level
    Call pairwisemerge;
    Point the HEAD pointer to the node with maximum value;
    Make the degree,parent,child,left,right,childcut  of previous maxvalue(previous HEAD)
    NULL
    Return the pointer to previous max node ( before calling pairwisemerge)
}
```

void increasekey(int pluskey, node* x)

Description	Increases the value of a given node by the given pluskey.	
Parameters	Int node*	The excess number by which the node value is to be increased. The address of the node whose value is to be increased.
Return	Returns nothing.	

Pseudocode-

```
Void increasekey( int y , node* x)
{
    if(x->parent == NULL)
        Increase the x value by the given y
    Else
      Increase the x value by the given y
       if(x->parent value < x->value)
            Call cut
            Make the childcut value of parent true if its false
             If its already true, call the cut for x->parent;

}
```

node* cut(node* x)

Description	Removes the node and its subtree from the current tree and adds it to the top level circular list and according to its parent childcut value it might do cascading cut.	
Parameters	node*	The address of the node which is to be cut from the heap.
Return	Returns parent of the child which is cut.	

Pseudocode-

```
node* cut(node* x)
{
    Remove x and insert it in the top level circular list;
    Make x->parent = NULL and childcut = 0;
    Adjust the circular list of x->parent->child level;
    Return x->parent;
}
```