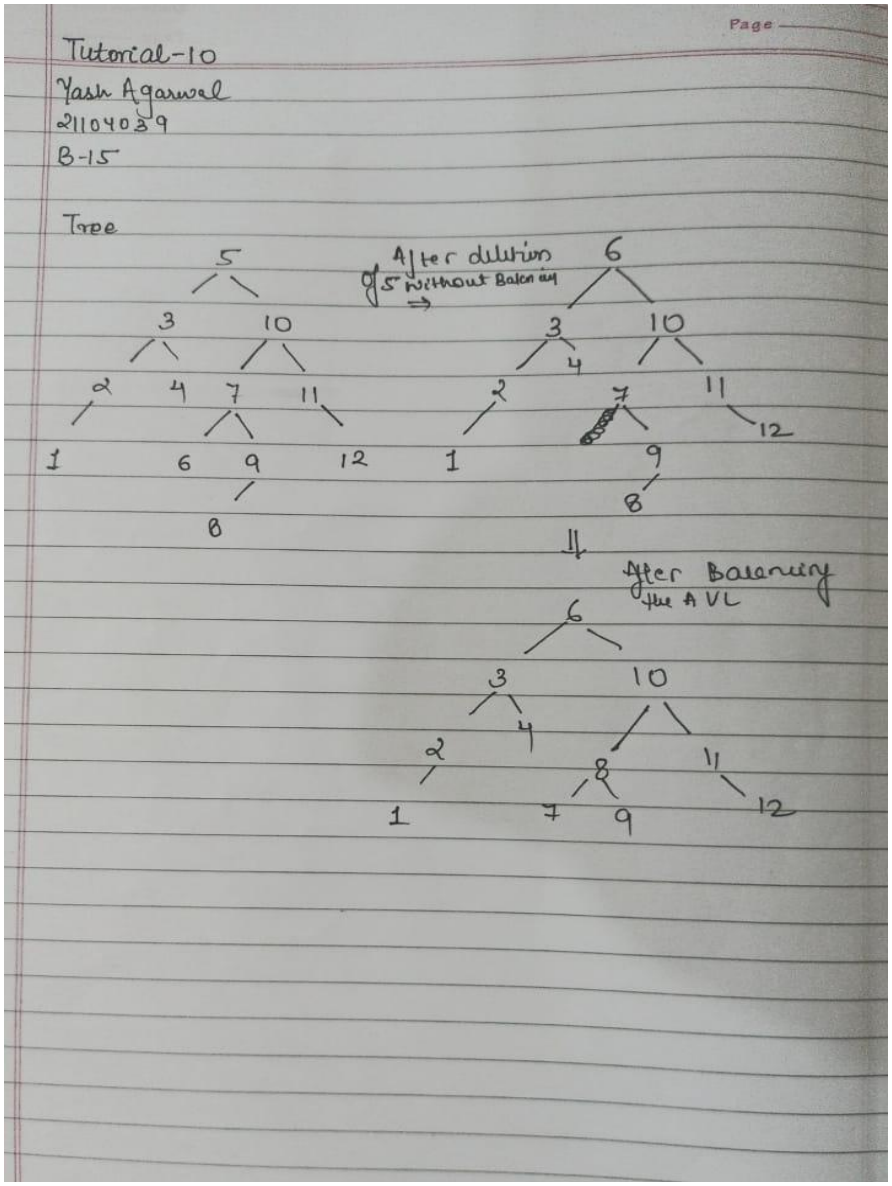

Question 2

```
#include<iostream>

using namespace std;

template<typename t>
class queue
{
    public:
    t data;
    queue<t> * next;
```

```
queue(t d)
{
   this->data=d;
   this->next=NULL;
}

void push(queue<t> * & q,t d)
{
   queue <t> * n = new queue<t> (d) ;

   if(q==NULL)
   {
      q=n;
      return;
   }
   else
   {
      queue<t> * temp = q;
      while(temp->next!=NULL)
      {
         temp=temp->next;
      }
      temp->next=n;
   }
}
void pop(queue<t> *& q)
{
   if(q!=NULL)
   {
      queue<t> *to=q;
      q=q->next;
      delete to;
   }
}
t top (queue<t> * q)
{
   if(q!=NULL)
   {
      return q->data;
   }
   return NULL;
}

bool isempty(queue<t> * & q)
```

```
    {
        return q==NULL ? true : false;
    }
};

class node
{
    public:
    int data;
    node * left;
    node * right;

    node (int d)
    {
        this->data=d;
        this->left=NULL;
        this->right=NULL;
    }

    void height_cal(node *root ,int h,int & ans)
    {
        if(root==NULL)
        {
            ans=max(h,ans);
            return ;
        }
        height_cal(root->left,h+1,ans);
        height_cal(root->right,h+1,ans);

    }



    int height_c(node * &root)
    {
        int ans=0;
        height_cal(root,0,ans);
        return ans;
    }

    int get_balance_factor(node *& root)
    {
        if(root==NULL) return -1;
```

```
    return height_c(root->left)-height_c(root->right);
}

node * left_r(node * x)
{
    node * y = x->left;
    node * b = y->right;
    y->right = x;
    x->left = b;
    return y;
}

node * right_r(node * x)
{
    node * y = x->right;
    node * b = y->left;
    y->left = x;
    x->right = b;
    return y;
}

node * insert_into_avl(node *  root,int d)
{
    if(root==NULL)
    {
        return new node (d);
    }
    if(root->data<d)
    {
        root->right=insert_into_avl(root->right,d);
    }
    else if(root->data>d)
    {
        root->left=insert_into_avl(root->left,d);
    }


    int bf=get_balance_factor(root);

    if(bf>1 && get_balance_factor(root->left) >=0)
    {
        return left_r(root);

    }
```

```
    if(bf>1 && get_balance_factor(root->left) <0)
    {
        root->left=right_r(root->left);
        return left_r(root);

    }
    if(bf<-1 && get_balance_factor(root->right) <=0)
    {
        return right_r(root);
    }
    if(bf<-1 && get_balance_factor(root->right) >0)
    {
        root->right=left_r(root->right);
        return right_r(root);

    }
    return root;
}

void insert (node * & root)
{
    int d;
    cout<<"enter the data : (-1 for NULL) ";
    cin>>d;

    while(d!=-1)
    {
        root=insert_into_avl(root,d);
        cout<<"enter the data : (-1 for NULL) ";
        cin>>d;

    }

}

void level_order(node * root)
{
    queue<node *> *q=NULL;

    q->push(q,root);
    q->push(q,NULL);

    while(!q->isempty(q))
    {
```

```
            node * f=q->top(q);
            q->pop(q);

            if(f==NULL)
            {
                if(!q->isempty(q))
                {
                    q->push(q,NULL);
                }
                cout<<endl;

            }
            else
            {
                cout<<f->data<<" ";
                if(f->left)
                {
                    q->push(q,f->left);
                }
                if(f->right)
                {
                    q->push(q,f->right);

                }

            }

        }
    }


};

int main()
{
    // queue<int> * q=NULL;
    // for(int i=0;i<10;i++)
    // {
    //     q->push(q,i+1);
    // }

    // while(!q->isempty(q))
    // {
```

```
    //     cout<<q->top(q)<<" ";
    //     q->pop(q);
    // }

    node * root=NULL;

    root->insert(root);

    root->level_order(root);

}


Question 3
#include <iostream>
using namespace std;

template <typename t>
class queue
{
public:
    t data;
    queue<t> *next;

    queue(t d)
    {
        this->data = d;
        this->next = NULL;
    }

    void push(queue<t> *&q, t d)
    {

        queue<t> *n = new queue<t>(d);

        if (q == NULL)
        {
            q = n;
            return;
        }
        else
        {
            queue<t> *temp = q;
            while (temp->next != NULL)
```

```
            {
                temp = temp->next;
            }
            temp->next = n;
        }
    }

    void pop(queue<t> *&q)
    {
        if (q != NULL)
        {
            queue<t> *n = q;
            q = q->next;
            delete n;
        }
    }
    t top(queue<t> *s)
    {
        if (s != NULL)
        {
            return s->data;
        }
        return -1;
    }

    bool isempty(queue<t> *&s)
    {
        return s == NULL ? true : false;
    }
};

class max_heap
{
public:
    int *arr;
    int size;
    int capacity;
    max_heap(int c)
    {
        this->arr = new int[c];
        this->capacity = c;
        this->size = 0;
    }
    int parent(int i) { return i / 2; }
```

```
int left_child(int i) { return (2 * i) + 1; }
int right_child(int i) { return (2 * i) + 2; }
void insert(int d)
{
    if (size == capacity)
    {
        cout << "\noverflow :\n ";
        return;
    }
    arr[size] = d;
    int i = size;
    size++;
    while (i >= 0)
    {
        if (arr[parent(i)] < arr[i])
        {
            int temp = arr[parent(i)];
            arr[parent(i)] = arr[i];
            arr[i] = temp;
            i = parent(i);
        }
        else
        {
            break;
        }
    }
}
int max_element()
{
    if (size < 0)
    {
        return -1;
    }
    return arr[size - 1];
}

void print()
{
    queue<int> *q = NULL;
    if (size <= 0)
    {
        cout << "\n empty";
        return;
    }
```

```
    q->push(q, 0);
    q->push(q, -1);

    while (!q->isempty(q))
    {
        int f = q->top(q);
        q->pop(q);

        if (f == -1)
        {
            if (!q->isempty(q))
            {
                q->push(q, -1);
            }
            cout << endl;
        }
        else
        {
            cout << arr[f] << " ";
            if (left_child(f) < size)
            {
                q->push(q, left_child(f));
            }
            if (right_child(f) < size)
            {
                q->push(q, right_child(f));
            }
        }
    }

    // for(int i=0;i<size;i++)
    // {
    //     cout<<arr[i]<<" ";

    // }
}

void remove(int d)
{
    int i;
    for (i = 0; i < size; i++)
    {
        if (arr[i] == d)
            break;
```

```
        }
        if (i == size)
        {
            cout << "\n not found \n";
            return;
        }

        arr[i] = arr[size - 1];
        size--;

        while (i < size)
        {
            int largest = i;
            int l = left_child(i);
            int r = right_child(i);

            if (l < size && arr[largest] < arr[l])
            {
                largest = l;
            }
            if (r < size && arr[largest] < arr[r])
            {
                largest = r;
            }
            if (largest != i)
            {
                int temp = arr[i];
                arr[i] = arr[largest];
                arr[largest] = temp;
                i = largest;
            }
            else
            {
                break;
            }
        }
    }
};

int main()
{

    // queue<int> *q=NULL;
```

```
    // for(int i=0;i<10;i++)
    // {
    //     q->push(q,i);
    // }

    // while(!q->isempty(q))
    // {
    //     cout<<q->top(q);
    //     q->pop(q);
    // }

    int c;
    cout << "enter the capacity : ";
    cin >> c;
    max_heap *m = new max_heap(c);

    int d;
    cout << "enter data : (-1 to stop) ";
    cin >> d;
    while (d != -1)
    {
        m->insert(d);
        cout << "enter data : (-1 to stop) ";
        cin >> d;
    }
    cout << "\n level order : \n";
    m->print();
    cout << "enter data for delete : (-1 to stop) ";
    cin >> d;
    while (d != -1)
    {
        m->remove(d);
        cout << "\n level order : \n";
        m->print();
        cout << "enter data for delete : (-1 to stop) ";
        cin >> d;
    }
}
```

Question 4

```
#include <iostream>
```

```
using namespace std;

template <typename t>
class queue
{
public:
   t data;
   queue<t> *next;

   queue(t d)
   {
      this->data = d;
      this->next = NULL;
   }

   void push(queue<t> *&q, t d)
   {

      queue<t> *n = new queue<t>(d);

      if (q == NULL)
      {
         q = n;
         return;
      }
      else
      {
         queue<t> *temp = q;
         while (temp->next != NULL)
         {
            temp = temp->next;
         }
         temp->next = n;
      }
   }

   void pop(queue<t> *&q)
   {
      if (q != NULL)
      {
         queue<t> *n = q;
         q = q->next;
         delete n;
      }
```

```
    }
    t top(queue<t> *s)
    {
        if (s != NULL)
        {
            return s->data;
        }
        return -1;
    }

    bool isempty(queue<t> *&s)
    {
        return s == NULL ? true : false;
    }
};

class p_queue
{
public:
    int *arr;
    int size;
    int capacity;
    p_queue(int c)
    {
        this->arr = new int[c];
        this->capacity = c;
        this->size = 0;
    }
    int parent(int i) { return i / 2; }
    int left_child(int i) { return (2 * i) + 1; }
    int right_child(int i) { return (2 * i) + 2; }
    void insert(int d)
    {
        if (size == capacity)
        {
            cout << "\noverflow :\n ";
            return;
        }
        arr[size] = d;
        int i = size;
        size++;
        while (i >= 0)
        {
            if (arr[parent(i)] < arr[i])
```

```
        {
            int temp = arr[parent(i)];
            arr[parent(i)] = arr[i];
            arr[i] = temp;
            i = parent(i);
        }
        else
        {
            break;
        }
    }
}
int max_element()
{
    if (size < 0)
    {
        return -1;
    }
    return arr[size - 1];
}

void print()
{
    queue<int> *q = NULL;
    if (size <= 0)
    {
        cout << "empty\n";
        return;
    }
    q->push(q, 0);
    q->push(q, -1);

    while (!q->isempty(q))
    {
        int f = q->top(q);
        q->pop(q);

        if (f == -1)
        {
            if (!q->isempty(q))
            {
                q->push(q, -1);
            }
            cout << endl;
```

```
        }
        else
        {
            cout << arr[f] << " ";
            if (left_child(f) < size)
            {
                q->push(q, left_child(f));
            }
            if (right_child(f) < size)
            {
                q->push(q, right_child(f));
            }
        }
    }

    // for(int i=0;i<size;i++)
    // {
    //     cout<<arr[i]<<" ";

    // }
}

void remove()
{
    if(size<0) {cout<<"underflow \n"; return;}
    if(size==0){
        size--;
        return;
    }
    arr[0] = arr[size - 1];
    size--;
   int  i=0;

    while (i < size)
    {
        int largest = i;
        int l = left_child(i);
        int r = right_child(i);

        if (l < size && arr[largest] < arr[l])
        {
            largest = l;
        }
        if (r < size && arr[largest] < arr[r])
```

```
            {
                largest = r;
            }
            if (largest != i)
            {
                int temp = arr[i];
                arr[i] = arr[largest];
                arr[largest] = temp;
                i = largest;
            }
            else
            {
                break;
            }
        }
    }
};

int main()
{
    int c;
    cout << "enter the capacity : ";
    cin >> c;
    p_queue*m = new p_queue(c);

    int d;
    cout << "enter data : (-1 to stop) ";
    cin >> d;
    while (d != -1)
    {
        m->insert(d);
        cout << "enter data : (-1 to stop) ";
        cin >> d;
    }
    cout << "\n level order : \n";
    m->print();
    cout << "enter any key for delete : (-1 to stop) ";
    cin >> d;
    while (d != -1)
    {
        m->remove();
        cout << "\n level order : \n";
        m->print();
        cout << "enter any key for delete : (-1 to stop) ";
```

```
        cin >> d;
    }
}
```