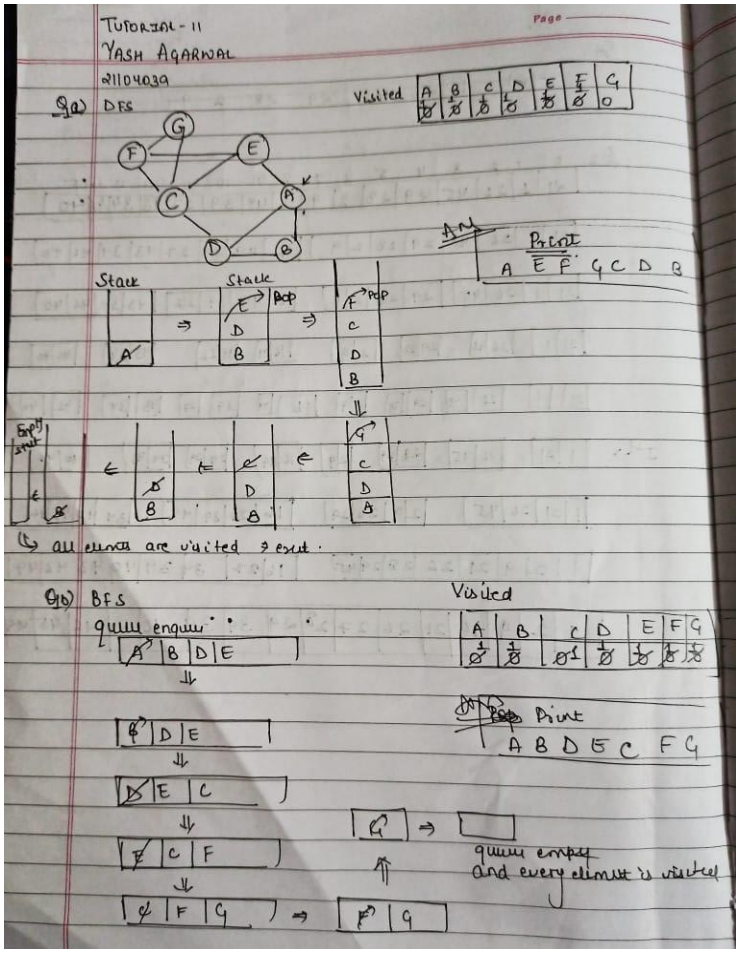
Q1



Q2

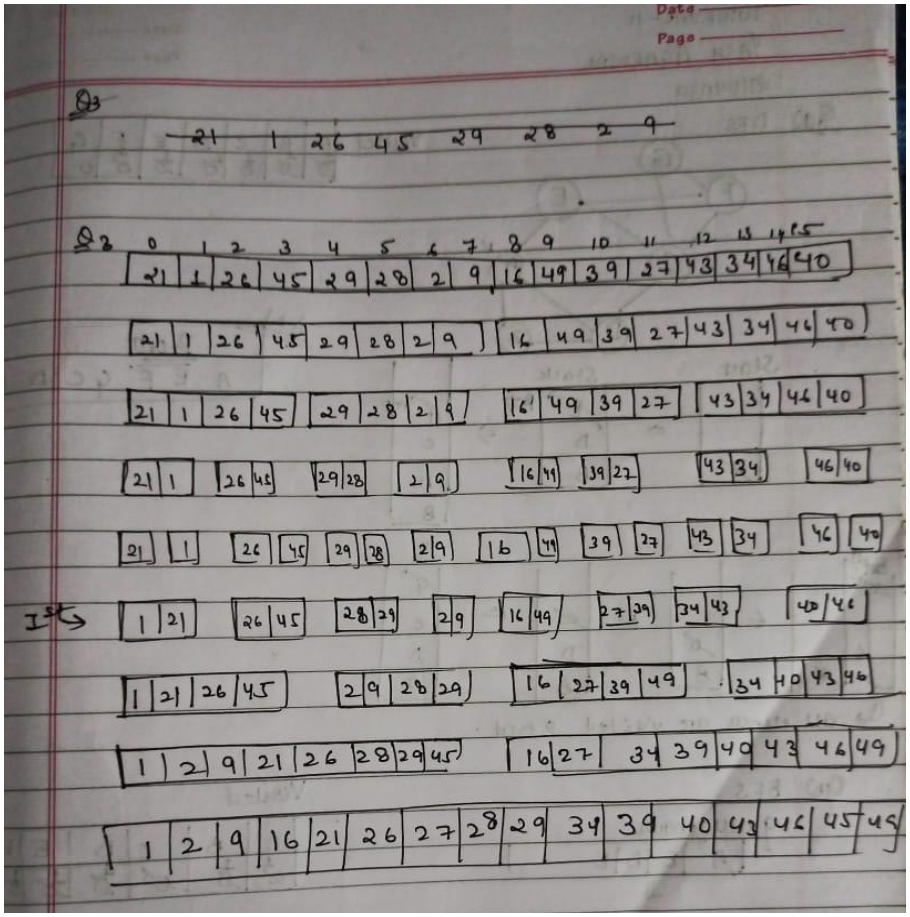
```
#include<bits/stdc++.h>
using namespace std;

class graph{
   public:
   int v;
   vector<vector<int>>adj;
   graph(int x)
   {
      v=x;
      adj.resize(x);
   }

   void addEdge(int s,int d);
```

```
    void dfs(int s);
    void bfs(int s);
    void print();
};

void graph :: addEdge(int s,int d)
{
    adj[s].push_back(d);
    adj[d].push_back(s);
}

void graph:: print()
{
    for(int i=0;i<v;i++)
    {
        cout<<" Edge at vertex "<<i<<"is with:";
        for(auto it: adj[i])
        {
            cout<<it<<" ";
        }
        cout<<endl;
    }
}

void graph:: dfs(int s)
{
    bool*visited=new bool[v];
    for(int i=0;i<v;i++)
    {
        visited[i]=false;
    }
    stack<int> st;
    st.push(s);
    while(st.empty()==false)
    {
        int s=st.top();
        st.pop();
        if(visited[s]==false)
        {
            cout<<s<<" ";
            visited[s]=true;
        }

        for(auto i: adj[s])
```

```
        {
            if(visited[i]==false)
            {
                st.push(i);
            }
        }
    }
}

void graph:: bfs(int s)
{
    bool * visited=new bool[v];
    for(int i=0;i<v;i++)
    {
        visited[i]=false;
    }
    queue<int>q;
    q.push(s);
    visited[s]=true;
    while(!q.empty())
    {
        s=q.front();
        cout<<s<<" ";
        q.pop();

        for(auto i: adj[s])
        {
            if(!visited[i])
            {
                visited[i]=true;
                q.push(i);
            }
        }
    }
}

int main()
{
    graph g(4);
    g.addEdge(0,1);
    g.addEdge(1,3);
    g.addEdge(1,2);
    g.addEdge(2,0);
    g.addEdge(3,0);
```

```
    g.print();
    g.dfs(1);
    cout<<endl;
    g.bfs(2);
    return 0;
}
```

Q3



Q 4

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

int partition(int arr[],int l,int r)
```

```
{
    int pivot=arr[r];
    int i=l-1;

    for(int j=l;j<r;j++)
    {
        if(arr[j]<pivot)
        {
            i++;
            swap(arr[i],arr[j]);
        }
    }
    swap(arr[i+1],arr[r]);
    return i+1;
}

void quickSort(int arr[],int l,int r)
{
    if(l<r)
    {
        int pi=partition(arr,l,r);
        quickSort(arr,l,pi-1);
        quickSort(arr,pi+1,r);
    }
}

int main()
{
    int arr[6]={2,8,1,5,3,4};
    quickSort(arr,0,5);
    for(int i=0;i<6;i++)
    {
        cout<<arr[i]<<" ";
    }
    return 0;
}
```