Algorithms

Problem 1: Images Fusion

- I did 3 nested loops in order to make the multiplication of the matrices, So the complexity is O(n^3)

```
#include <iostream>
#include<vector>
using namespace std;

vector<vector<int> > multiply(vector<vector<int> >mat1,vector<vector<int>
>mat2,vector<vector<int> >res, int N)
{

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            res[i][j] = 0;
            for (int k = 0; k < N; k++)
                res[i][j] += mat1[i][k] * mat2[k][j];
        }
    }
    return res;
}

int main()
{


   vector<vector<int> >mat1;
    mat1.resize(2);
     mat1[0].push_back(3);
      mat1[0].push_back(4);
      mat1[1].push_back(4);
      mat1[1].push_back(2);
     vector<vector<int> >mat2;
       mat2.resize(2);
     mat2[0].push_back(3);
      mat2[0].push_back(1);
      mat2[1].push_back(9);
      mat2[1].push_back(3);
     vector<vector<int> >res;
      res.resize(2);
     res[0].resize(2);
      res[1].resize(2);
    // multiply(mat1, mat2, res, 2);

    cout << "Result matrix is \n";
    for ( int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        cout <<  multiply(mat1, mat2, res, 2)[i][j] << " ";
        cout << "\n";
    }

    return 0;
}
```

Problem 2: Function evaluation

- I converted the degree to the radian which costs 2 operations
- Inside the for loop, there are 9 athematic operations
- I handled the code to make the accuracy of the sin will be less than 0.0001
- In order to make the complexity O(n), I utilized the past computations in the factorial and the power instead using o(n) functions that will increase the complexity
- Complexity is O(n)

```
#include <iostream>

using namespace std;
float sinx(float n){
    //converting it to radian
    int operations=0;
     n=n * (3.142 / 180.0);      // contain 1 multiplication and 1 divison
       operations=operations+2;

    float x=n;  //will be the nominator
    float sinvalue=n;
 float dominator=1;
 int counter=1;
 while(x/dominator >0.0001) //can be manipulated to make the accuracy i want
 {
     //handling the dominator
     dominator=dominator*(2*counter)*(2*counter+1); //4 multiplication and 1
addition
     operations=operations+5;
     //handling the power  of n
     x=x*n*n;//2 multiplication
     operations=operations+2;
     //handling the sinx
     operations=operations+2;
     if(counter%2==0)
     sinvalue=sinvalue+(x/dominator);
     else
      sinvalue=sinvalue-(x/dominator);


     counter++;

}

cout<<"number of float arthimatic operations ="<<operations<<endl;
 return sinvalue;

}
int main()
{
   int x;
   cin>>x;
    cout<<"sin("<<x<<") = "<<sinx(x);



    return 0;
}
```

Problem 3: Say Cheeeese

Brute force

- Firstly, I think about how to do it use brute force algorithm. So, I compared every student to other to see if it is smaller than the next ones.
- In the example of the slides, the exact number is 15 comparisons.

so complexity is O(n^2)

```
#include <iostream>

using namespace std;

void sayCheese( int  students[], int n){
    int count=0;
    int instrCoun=0;
    for(int i=0;i<n;i++)
       for(int j=i+1;j<n;j++)
      {
          instrCoun++;
          if(students[i]<students[j])
             count++;


      }
       cout<<count<<endl;
    cout<<"comparisons  "<<instrCoun<<endl;

}
int main()
{
     int  students[6]={160,140,190,150,180,170};
    sayCheese( students, 6);


    return 0;
}
```

I tried to make it using one while loop, but I don’t know what is the complexity, it think it is o(n^squared) also. But I tried my best in it.

```
#include <iostream>

using namespace std;


//---------------------------------------//
void sayCheese( int  students[], int end,int start){
     bool  students_status[end-1]={false};
     int instrCount=0;
    int count=0;
    int all=end;
    int indication=0;
    int smallest=12345678;
    int lastSmallest;
    int lastindex;
    int smallestindex=-1;
    while (indication!=all){

    //      if((smallestindex<end-1)&&(students_status[end-1]==false))
    //    {
    //       count++;
    //       cout<<" "<<end-1<<" "<<endl;
    //    }
```

```
       if( students[start]<smallest&&students_status[start]==false)
      {
          instrCount++;
          smallest =students[start];
          smallestindex=start;
      }
    //     if( students[end-1]<smallest&&students_status[end-1]==false)
    //   {
    //       smallest =students[end-1];
    //       smallestindex=end-1;
    //   }

    // end--;
    start++;
    if(start==all)
      {

           cout<<smallest<<"->"<<smallestindex<<"->"<<count<<endl;
          students_status[smallestindex]=true;
          lastSmallest=smallest;
          lastindex=smallestindex;
          smallest=123456;
        //  end=all;
         start=0;
          indication++;

      }
      if((lastindex<start) &&(students_status[start]==false))
      {
          count++;
          cout<<start<<" " <<endl;

      }

    }


    cout<<"inst"<<instrCount<<endl;
}

int main()
{
    int  students[10]={160,140,190,150,180,170,12,56,89,43};


    sayCheese( students, 10,0);



    return 0;
}
```

Problem 4: google form

Problem 5: Friendships formation

- I iterates over every element in the 2d array, so it is a brute force algorthim

```
#include <iostream>

using namespace std;
#define n 5
void Friendships(int array[][n]){
    int Fully_connected=1;   //flag
    int Star_topology=1;   //flag
    int Ring_topology=1;     //flag
    int starRows=0;
    int starCol=0;
    for(int i=0;i<n;i++){
        int countRow=0;
        int countCol=0;
        for(int j=0;j<n;j++){
            //checking if it is fully connected or not
            if(i!=j && array[i][j]!=1)
            Fully_connected=0;
             //some steps to check the Ring_topology
             if(array[i][j]==1)
             countRow++;   // this is used in Star_topology as well
             if(array[j][i]==1)
             countCol++;
             //checking if it is Star_topology or not



        }
        //checking if it is Ring_topology or not
        if(countRow>2||countRow<2||countCol>2||countCol<2)
        Ring_topology=0;
       //checking if it is Star_topology or not
       if(countRow==n-1)
       {
           starRows++;
           starCol++;
           if(starRows>1||starCol>1)
           Star_topology=0;
       }
       if((countRow!=1&&countRow!=n-1)||(countCol!=1&&countCol!=n-1))
        Star_topology=0;
    }


    if( Fully_connected==1)
    cout<<"Fully_connected"<<endl;
    else  if( Star_topology==1)
    cout<<"Star_topology"<<endl;
    else  if( Ring_topology==1)
    cout<<"Ring_topology"<<endl;
    else cout<<"No topology  found"<<endl;

}

int main()
{
    //ring
    // int array[5][5]=
    // {
    //     {0,0,1,0,1},
    //     {0,0,0,1,1},
    //     {1,0,0,1,0},
    //     {0,1,1,0,0},
    //     {1,1,0,0,0}
    // };
    //Fully_connected
    //  int array[5][5]=
    // {
    //     {0,1,1,1,1},
    //     {1,0,1,1,1},
    //     {1,1,0,1,1},
    //     {1,1,1,0,1},
    //     {1,1,1,1,0}
    // };
    //Star_topology
     int array[5][5]=
    {
        {0,1,1,1,1},
        {1,0,0,0,0},
        {1,0,0,0,0},
        {1,0,0,0,0},
        {1,0,0,0,0}
    };
    //ring
    // int array[5][5]=
    // {
    //     {0,1,0,0,1},
    //     {1,0,1,0,0},
    //     {0,1,0,1,0},
    //     {0,0,1,0,1},
    //     {1,0,0,1,0}
    // };
Friendships(array);
    return 0;
}
```