Sort

```
class Complex
{
    private:
        double x,y;
    public:
        Complex(double x=0.0,double y=0.0):x(x),y(y){};
        double getModulus()
        {
            double modul=sqrt(x*x+y*y);
            return modul;
        }
        bool operator==(Complex& a)
        {
            return (this->x==a.getModulus() && this->y==a.getModulus());
        }
        bool operator !=(Complex& a)
        {
             return !(this->x==a.getModulus() && this->y==a.getModulus());

        }
        bool operator >(Complex& a)
        {
            return this->getModulus()>a.getModulus();
        }
        bool operator <(Complex& a)
        {
            return this->getModulus() < a.getModulus();
        }
        bool operator >=(Complex& a)
        {
            return this->getModulus()>=a.getModulus();
        }
        bool operator <=(Complex& a)
        {
            return this->getModulus()<=a.getModulus();
        }

        string toString()
        {
            string x_str=to_string(this->x),y_str=to_string(this->y);
            while(x_str[x_str.length()-1]=='0' )
            {

                unsigned int size=x_str.length()-1;
                x_str.erase(size);
            }
            if(x_str[x_str.length()-1]=='.') x_str.erase(x_str.length()-1);
            while(y_str[y_str.length()-1]=='0' ){ unsigned int size=y_str.length()-1;
y_str.erase(size);}
            if(y_str[y_str.length()-1]=='.')y_str.erase(y_str.length()-1);
            if(y_str!="0")
            {
                if(x_str!="0")
                {
                    if(y_str[0]!='-'){ y_str="+ "+y_str;}

                    else{ y_str.insert(1," ");
                    }
                    return x_str+" "+y_str+" * i";
                }
                else
                return  y_str+" * i";
            }
            return  x_str;
        }
};
class StraightSelectionSort
{
    public:
static void sort(Complex *list, int length)
{
    for(int i=0;i<length-1;i++)
       {
           int min=i;
           for(int j=i+1;j<length;j++)
           {
               if(list[j]<list[min])
                min=j;
           }
            swap(list[i],list[min]);
       }

}
static void sort(Complex *list, int length, int left, int right)
{
    for(int i=left;i<right;i++)
       {
           int min=i;
           for(int j=i+1;j<=right;j++)
           {
               if(list[j]<list[min])
                min=j;
           }
```

```
            swap(list[i],list[min]);
        }
    }
};
class StraightInsertionSort
{
    public:
    static void sort(Complex *list, int length)
    {
        if(length>1)
        {
            int curr=1;
            while (curr<length)
            {
                /* code */
                Complex tmp=list[curr];
                int step=curr-1;
                while(step>=0&& tmp<list[step] )
                {
                    list[step+1]=list[step];
                    step--;
                }
                list[step+1]=tmp;
                curr++;
            }

        }
    }
    static void sort(Complex *list, int length, int left, int right)
    {
        int curr=left;
            while (curr<=right)
            {
                /* code */
                Complex tmp=list[curr];
                int step=curr-1;
                while(step>=left&& tmp<list[step] )
                {
                    list[step+1]=list[step];
                    step--;
                }
                list[step+1]=tmp;
                curr++;
            }
    }
};
class  BubbleSort
{
    public:
    static void sort(Complex *list, int length)
    {
       int curr=0;
       bool flag=false;
       while(curr<length && flag==false)
       {
           int step=length-1;
           flag =true;
           while(step>curr)
           {
               if(list[step]<list[step-1])
               {
                   flag=false;
                    swap(list[step],list[step-1]);
               }
```

```
                step--;
            }
            curr++;
        }
    }
    static void sort(Complex *list, int length, int left, int right)
    {
          int curr=left;
       bool flag=false;
       while(curr<=right && flag==false)
       {
           int step=right;
           flag =true;
           while(step>curr)
           {
               if(list[step]<list[step-1])
               {
                   flag=false;
                    swap(list[step],list[step-1]);
               }
               step--;
           }
           curr++;
       }
    }
};
class ShellSort
{
    private:
        int *increments;
        int lengthOfIncrements;
    public:
```

```
ShellSort(int *increments, int lengthOfIncrements)
{
    this->lengthOfIncrements=lengthOfIncrements;
    this->increments=increments;
}
static void sortSegment(Complex *list, int length, int increment, int segment = 0)
{
    int k=increment;
      int curr=segment+k;
      while(curr<length)
      {
          Complex temp=list[curr];
        int step =curr-k;
        while(step>=0 && temp<list[step])
        {
            list[step+k]=list[step];
            step=step-k;
        }
        list[step+k]=temp;
        curr+=k;
      }

}
void sort(Complex *list, int length)
{
    int k=10;
    for(int i=0;i<lengthOfIncrements && k>=1;i++)
    {
        k=this->increments[i];
        int segment=0;
        while(segment<k)
        {
                sortSegment(list,length,k,segment);
                segment++;
            }

        }
    }
};
class MergeSort
{
    public:
    static int merge(Complex *&list, int left, int mid, int right)
    {
        Complex *a1=new Complex[right+1],*a2=new Complex[right+1];
        for(int i=left;i<=mid;i++)
        {
            a1[i]=list[i];
        }
        for(int j=mid+1;j<=right;j++)
        {
            a2[j]=list[j];
        }
        int i=left,j=mid+1,k=left;
        while(k<=right)
        {
            if(i>mid)
            {
                list[k]=a2[j];
                j++;
            }
            else if(j>right)
            {
                list[k]=a1[i];
            i++;
        }
        else
        {
            if(a1[i]<=a2[j])
            {
                list[k]=a1[i];
                i++;
            }
            else
            {
                list[k]=a2[j];
                j++;
            }
        }
        k++;
    }
    delete[] a1;
    delete[] a2;
    return 0;
}
static void mergesortRec(Complex *&list,int lo,int hi)
{
    if(hi>lo)
    {
        int mid=lo+(hi-lo)/2;
        mergesortRec(list,lo,mid);
        mergesortRec(list,mid+1,hi);
        merge(list,lo,mid,hi);
    }
}
static void sort(Complex *list, int length)
```

```
    {
        mergesortRec(list,0,length-1);
    }
};
```