201402414 장수훈

Inverstions & Closest pair

1. Inverstions

1) Counting_Inverstions 함수

```
static int Counting_Inversions(int arr[], int count) {
    int temp[] = new int[count];
    return merge_Sort(arr, temp, 0, count - 1);
}
```

배열 arr와 인덱스크기를 인트형으로 받아서 Incerstions이 몇회가 일어났는지 구하기 위해 merge sort함수로 들어가게 된다.

2) Merge 함수

```
    static int merge(int arr[], int result[], int indexA, int indexB, int right) {

        int inversion_count = 0;

        int i = indexA;
        int j = indexB;
        int k = indexA;
        while ((i <= indexB - 1) && (j <= right)) {
            if (arr[i] <= arr[j]) {
                result[k++] = arr[i++];
            } else {

                result[k++] = arr[j++];

                inversion_count = inversion_count + (indexB - i);
            }
        }

        while (i <= indexB - 1)
            result[k++] = arr[i++];

        while (j <= right)
            result[k++] = arr[j++];

        for (i = indexA; i <= right; i++)
            arr[i] = result[i];

        return inversion_count;
    }

}
```

저번의 merge함수에서 inverstion 횟수만 측정하는것만 추가되었다. Merge가 일어날 때 inverstion이 몇번 일어났는지 측정하기 위해 변수를 선언하여 그 값을 리턴하였다.

3) Merge_sort 함수

```
static int merge_Sort(int arr[], int temp[], int left, int right) {
    int mid, inversion_count = 0;
    if (right > left) {

        mid = (right + left) / 2;

        inversion_count = merge_Sort(arr, temp, left, mid);
        inversion_count = inversion_count + merge_Sort(arr, temp, mid + 1, right);
        inversion_count = inversion_count + merge(arr, temp, left, mid + 1, right);
    }

    return inversion_count;
}
```

중간 인덱스와 inverstion의 횟수를 저장할 변수를 선언하고 merge전의 좌측 ,우측 을 merge 하면서 합계를 구해 리턴 해준다.

결과출력

```
Input Data : 1 5 4 8 10 2 6 9 12 11 3 7
Output : 22
Sort : 1 2 3 4 5 6 7 8 9 10 11 12
```

2. Closest Pair

1) Main 함수

```
for (int i = 0; i < str_array.length; i += 2) {
    Location Location = new Location(Double.parseDouble(str_array[i]), Double.parseDouble(str_array[i + 1]));
    arrayList.add(Location);
}
```

txt로부터 좌표를 읽어와 x좌표를 기준으로 끊어서 x,y,좌표를 Location 객체에 x,y 좌표로 배열에 저장해준다.

2) Closest Pair

```
public static double Closest_Pair(List<Location> arrayList) {
    if (arrayList.size() <= 3) {
        return Bruteforce(arrayList);
    }

    double left_point = Closest_Pair(arrayList.subList(0, arrayList.size() / 2));

    double right_point = Closest_Pair(arrayList.subList(arrayList.size() / 2, arrayList.size()));
    double Min = Min(left_point, right_point);

    List<Location> List_Location = new ArrayList<>();

    for (int i = 0; i < arrayList.size(); i++) {
        double base = ((int) arrayList.get(arrayList.size() / 2 - 1).getX_Location()
                + (int) arrayList.get(arrayList.size() / 2).getX_Location()) / 2 - arrayList.get(i).getX_Location();

        if (abs(base) < Min) {
            List_Location.add(arrayList.get(i));
        }
    }

    for (int i = 0; i < sort(List_Location).size() - 1; i++) {

        for (int j = i + 1; j < sort(List_Location).size()
                && (List_Location.get(j).y_Location - List_Location.get(i).y_Location) < Min; j++) {

            Min = Min(Min, Distance(List_Location.get(i), List_Location.get(j)));
        }
    }
    return Min;
}
```

기준을 잡아 그 기준으로부터 왼쪽, 오른쪽 좌표를 저장하고 기준점으로부터 최소값 이내에 있는 좌표만 분리하였다. 이후 y값을 기준으로 정렬하여 리턴해주었다.

3) BruteForce 함수

```
public static double Bruteforce(List<Location> arrayList) {
    double brute = Distance(arrayList.get(0), arrayList.get(1));

    for (int i = 0; i < arrayList.size(); i++) {
        for (int j = 0; j < arrayList.size(); j++) {
            if (i != j) {
                if (brute > Distance(arrayList.get(i), arrayList.get(j))) {
                    brute = Distance(arrayList.get(i), arrayList.get(j));
                }

            }
        }
    }
    return brute;
}
```

좌표 수가 3 이하이면 brute force로 계산한다. Loop invariant로 ermination condition시 모든 포인트에 대하여 반복하여 거리를 구하기 위해서 이중 포문으로 구현하였다. 주어진 txt 로는 이 함수를 거치지 않지만 조건을 걸어 이 함수를 거쳐도 이상이 없음을 확인하였다.

4) Min, Distance 함수

```
public static double Min(double tmp1, double tmp2) {
    if (tmp1 > tmp2) {
        return tmp2;
    } else {
        return tmp1;
    }
}

public static double Distance(Location Location1, Location Location2) {
    double distance = Math.sqrt(Math.pow(abs(Location2.getX_Location() - Location1.getX_Location()), 2)
            + Math.pow(abs(Location2.getY_Location() - Location1.getY_Location()), 2));
    return distance;
}
```

최소값과 좌표 사이 거리를 구하는 함수

5) Sort 함수

```
public static List<Location> sort(List<Location> L) {
    for (int i = 1; i < L.size(); i++) {
        Location base = L.get(i);
        int tmp = i - 1;
        while (tmp >= 0 && base.y_Location < L.get(tmp).y_Location) {
            Location temp = L.get(tmp);
            L.remove(L.get(tmp));
            L.add(tmp + 1, temp);
            tmp--;
        }

        L.add(tmp + 1, base);
        L.remove(base);

    }

    return L;
}
```

순차정렬 함수이다. 제일 구현하기 쉽고 첫번째 목록에 있길래 이정렬로 하였다.

6) Lcation 클레스

```
static class Location implements Comparable<Location> {

    public double x_Location;
    public double y_Location;

    public Location(double x_Location, double y_Location) {
        this.x_Location = x_Location;
        this.y_Location = y_Location;
    }

    public double getX_Location() {
        return x_Location;
    }

    public double getY_Location() {
        return y_Location;
    }

    public void setX_Location(double x_Location) {
        this.x_Location = x_Location;
    }

    public void setY_Location(double y_Location) {
        this.y_Location = y_Location;
    }

    public int compareTo(Location Location) {
        if (this.x_Location < Location.getX_Location()) {
            return -1;
        } else if (this.x_Location > Location.getX_Location()) {
            return 1;
        }
        return 0;
    }
}
```

좌표들의 x,y 값을 저장하여 객체로 사용하기 위해 만든 클레스이다. 생성자와 호출자 설정자를 선언하였다.

결과값출력

```
input :
1.23,12.3
1.0,2.0
3.1,21.2
5.2,10.0
output : 4.588
```