GROUP 10

Atul Kumar – IIT2018030

Raushan Raj – IIT2018031

INTRODUCTION

OUR ALGORITHM IS DESIGNED TO FIND K-NEAREST NEIGHBOURS AND K-FARTHEST NEIGHBOURS FROM A START POINT IN A MATRIX .

> WE HAVE GENERATED A 100*100 MATRIX WITH RANDOM ELEMENTS USING FUNCTION RAND.

> WE HAVE GIVEN A START POINT AND FROM WHICH WE HAVE TO FIND K- NEAREST AND K- FARTHEST NEIGHBOURS FROM THAT POINT IN A MATRIX .

ALGORITHM

> > IN OUR ALGORITHM WE HAVE GENERATED A MATRIX OF 100*100 WITH RANDOM ELEMENTS USING A RANDOM FUNCTION

> WE TAKE K AS INPUT TO GET K – NEAREST NEIGHBOURS FROM THE START-POINT AS WELL AS FOR K-FARTHEST POINTS

ALGORITHM: Main

```
function GENERATE MATRIX(k)
Random dice = new Random();
for i ← 0 to n-1
for j ← 0 to n-1
a[i][j] ← dice.nextInt(2)

KNN(ExtractKernel)
KFF (ExtractKernel)

AlgEnds
```

ALGORITHM

.> IN K-NEAREST NEIGHBOUR ALGORITHM WE TAKE START POINTS .

> WE HAVE CALCULATED DISTANCE BETWEEN TWO POINTS BY EUCLIDEAN DISTANCE FORMULA AND STORED IN AN 2D ARRAY USING EUCLIDEAN
DISTANCE FORMULA (DIST ((X, Y), (A, B)) =
$\sqrt{(X - A)^2 + (Y - B)^2}$.

>WE SORT THE ARRAY IN ASCENDING ORDER IN K-NEAREST AND DESCENDING ORDER IN K-FARTHEST NEIGHBOURS.

ALGORITHM: KNN

IMPLEMENTATION

Function : *implements* ***knn*** *in the kernel matrix .*

```
Mat1[4][2] ← {(x,y),(x,n),(n,y),(n,n)}
B ←4
for p←0 to B
begin
for i ←x to n
for j ←y to n

Dist[i][j] ← sqrt(pow(Mat1[p][0]-
i,2)+pow(Mat[p][1]-j,2))
end
Min_Sort(Dist[][])
Begin
dist=Dist[0][0];
for p←0 to B
begin
for i ←0 to n
for j←0 to n
begin
closest_dist = Dist[i][j]
If(dist < closest_dist && dist != 0)
Begin
Mat1[p][0]←i
Mat1[p][1]←j
a←i
b←j
end
end
dist[a][b]=1000000000
end
for i←0 to k
for j←0 to 1
print Mat1[i][j]
```

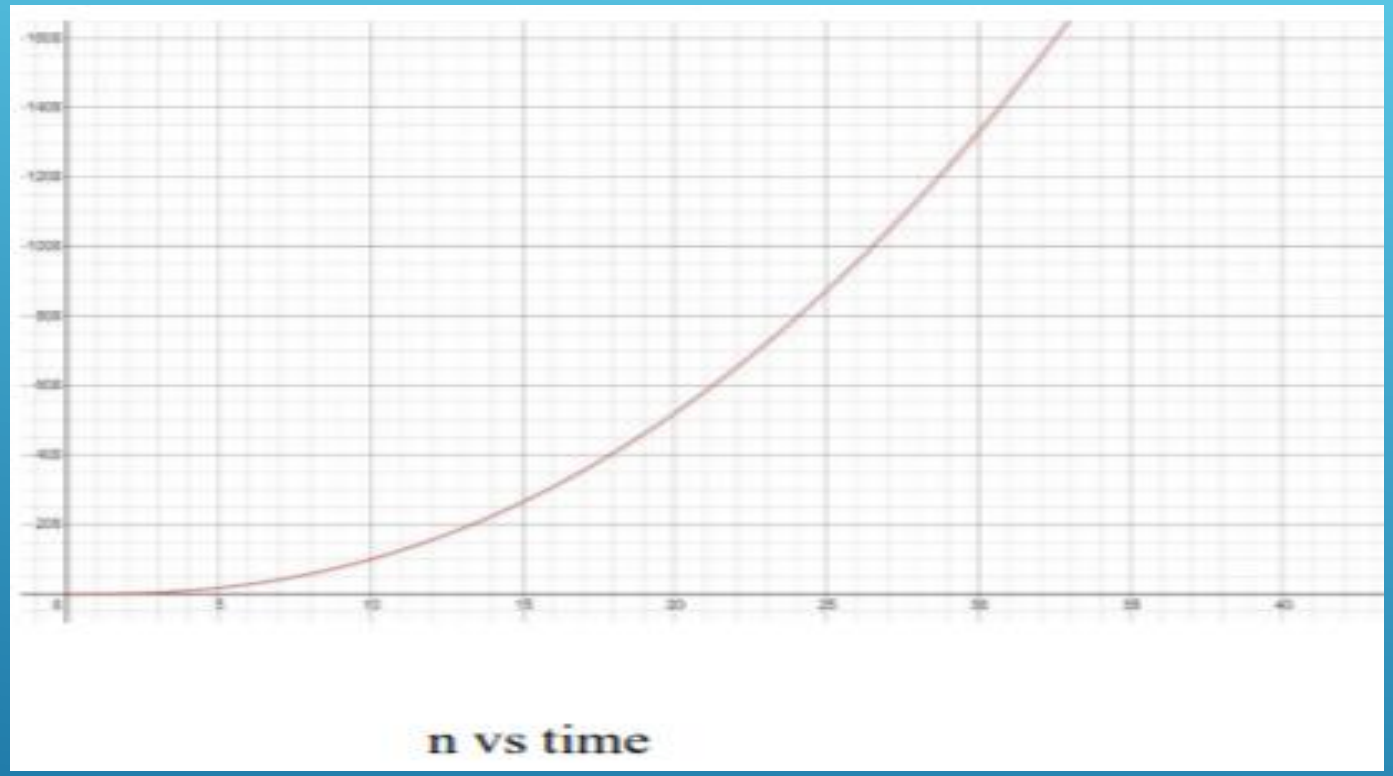
EXPERIMENTAL ANALYSIS

THE WORST CASE

TIME COMPLEXITY OF THE ALGORITHM KNN AND KFF TAKE **$O(N^2LOGN)$** AS A WORST CASE COMPLEXITY KNN AND KFF BASED MODEL ON DIFFERENT VALUES.

LOGARITHM: $O(N^2LOGN)$

THE GRAPH



TIME COMPLEXITY

ALGORITHM: KFF
IMPLEMENTATION

```
Mat1[4][2] ← {(x,y),(x,n),(n,y),(n,n)}
B ←4
for p←0 to B
begin
for i ←x to n
for j ←y to n
Dist[i][j] ← sqrt(pow(Mat1[p][0]-
i,2)+pow(Mat[p][1]-j,2))
end
Max_Sort(Dist[][])
Begin
dist=Dist[0][0];
for p←0 to B
begin
for i ←0 to n
for j←0 to n
begin
farthest_dist = Dist[i][j]
If(dist > farthest_dist && dist != 0)
Begin
Mat1[p][0]←i
Mat1[p][1]←j
a←i
    b←j
    end
    end
    dist[a][b]=0
    end
    for i←0 to k
    for j←0 to 1
    print Mat1[i][j]
```

For Worst case : - >

- Time complexity for worst case is **logarithm**:$O(n^2 Logn)$

ALGORITHM: KNN
IMPLEMENTATION

Function : *implements* ***knn*** *in the kernel matrix .*

```
Mat1[4][2] ← {(x,y),(x,n),(n,y),(n,n)}
B ←4
for p←0 to B
begin
for i ←x to n
for j ←y to n
Dist[i][j] ← sqrt(pow(Mat1[p][0]-
i,2)+pow(Mat[p][1]-j,2))
end
Min_Sort(Dist[][])
Begin
dist=Dist[0][0];
for p←0 to B
begin
for i ←0 to n
for j←0 to n
begin
closest_dist = Dist[i][j]
If(dist < closest_dist && dist != 0)
Begin
Mat1[p][0]←i
Mat1[p][1]←j
a←i
b←j
end
end
dist[a][b]=1000000000
end
for i←0 to k
for j←0 to 1
print Mat1[i][j]
```

CONCLUSION

>THE ALGORITHM IS AN SIMPLE ILLUSTRATION OF K-NEAREST AND K-FARTHEST NEIGHBOUR OF A GIVEN START POINT IN A SPACE.

>THE COMPLEXITY IS A LOGARITHMIC FUNCTION OF TIME --- $O(N^2LOGN)$.

THANK YOU