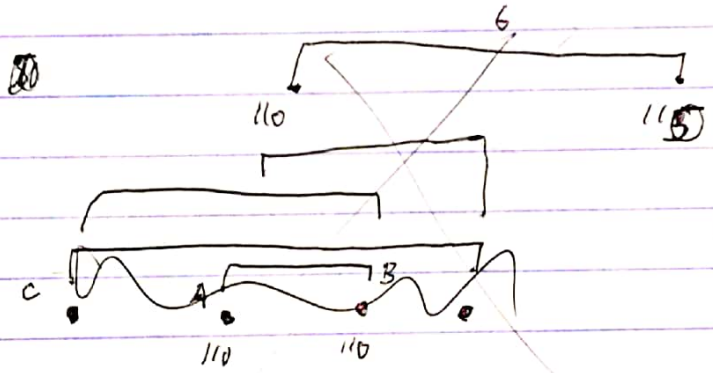
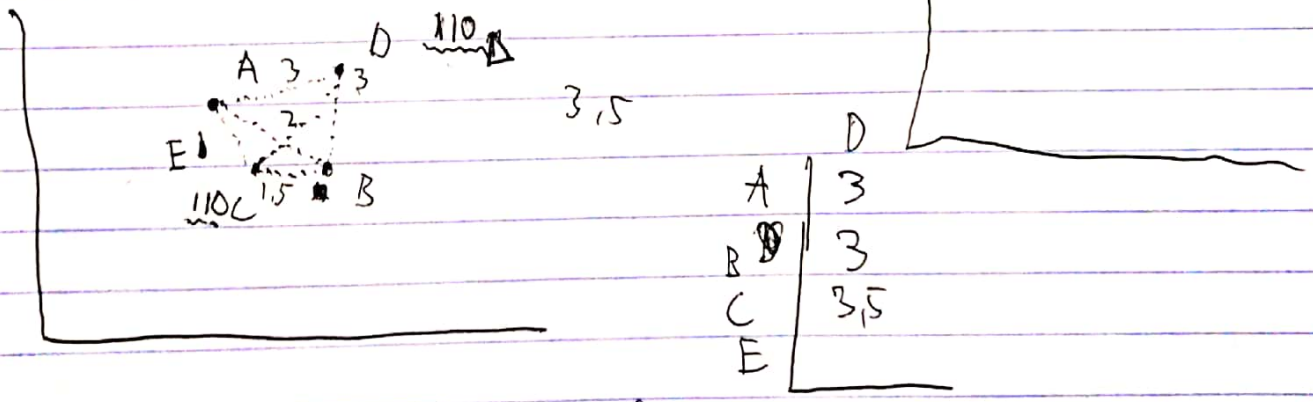
- List of points
- Each point/ID must contain elements.
∴ Class or struct object
- List of objects
- Insert element, update list of distance from every element
could be ↓
So Inefficient!

Shortest



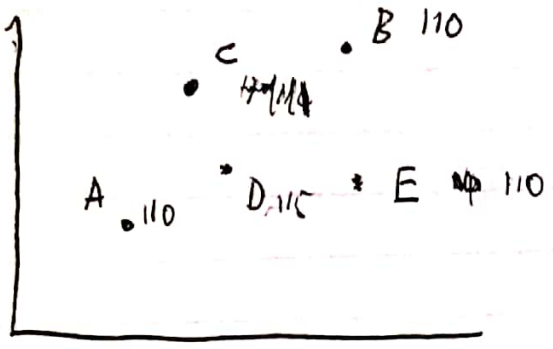
∴ This linear



Reminds me of Travelling salesman.
Not really, but NP-hard
TLDR

Graph Theory

median distance



Distance table

weighted Complete graph (undirected)

	A	
B	4,5	110
C	2	112
D	2	112
E	4	111

	B	
A	4,5	110
C	2	112
D	2,5	112
E	3	111

	113	113
	112	112
	111	111

	C	
A	2	110
B	2	110
D	1	112
E	3	111

	D
A	2
B	2,5
C	1
E	0,5

	E
A	4
B	3
C	3
D	0,5

4 6

5 vertex

10 Edges

5-16

7-23

A-B	4,5
A-C	2
A-D	2
A-E	4
B-C	2
B-D	2,5
B-E	3
C-D	1
C-E	3
D-E	0,5

Find median dist

115 110

115 114

110 115

110 115

Read ~~A~~ Insert all distances into a list (array?) / no vector

dist struct dist { left* node
right* node
distance

C++ vector

Make it point to node

node { position
ID
Frequency

Vect of distances

A-B
A-C
A-D
B-C
B-D ... etc.

assign values based on normalised data ∴ find normalise

data range { check formula / 0

(110-115)

Normalize from 0:5

set frequency based on distance ~~from node~~ vector list (A-B, A-C) etc

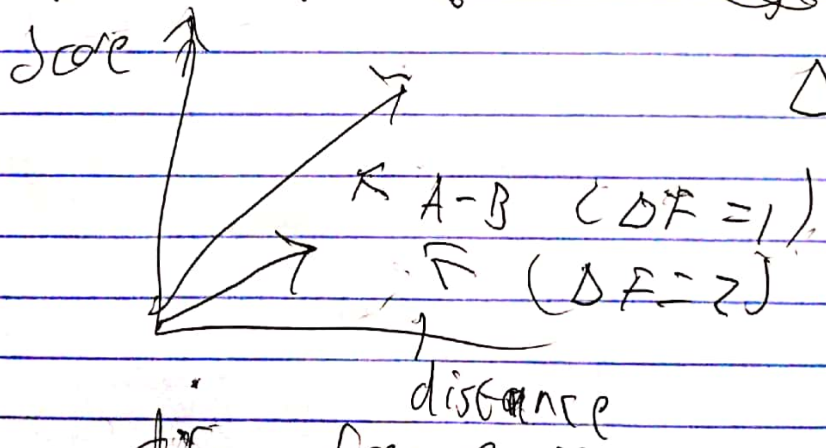
Arrange list from smallest distance to longest & then algorithm

Solution done by C
verify?

graph?
Visually? → x:y → frequency
make random frequency and assign scoring metric ✓

random frequency done → optimal frequencies in one of them

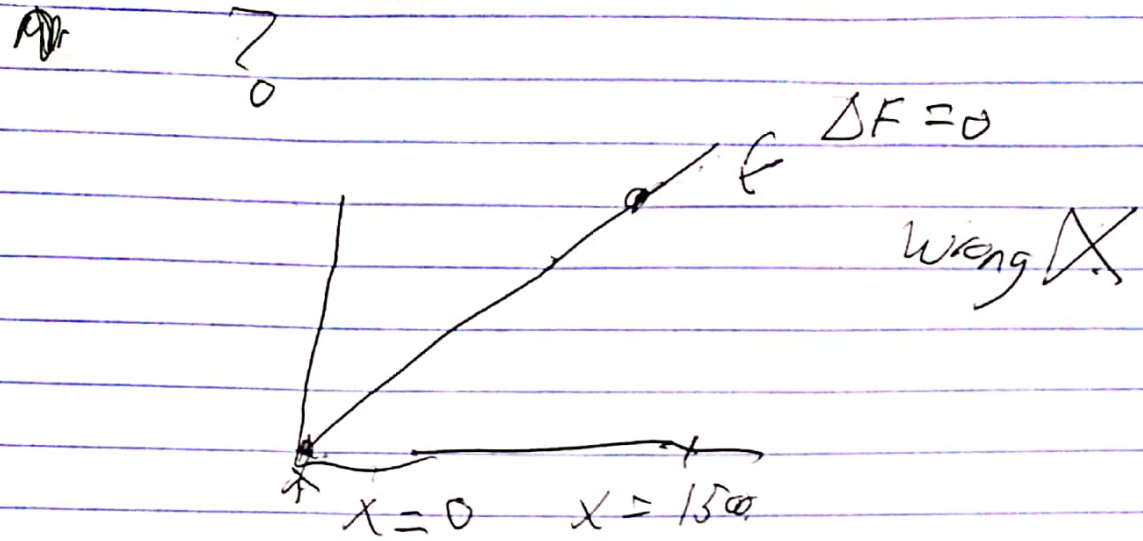
metric?

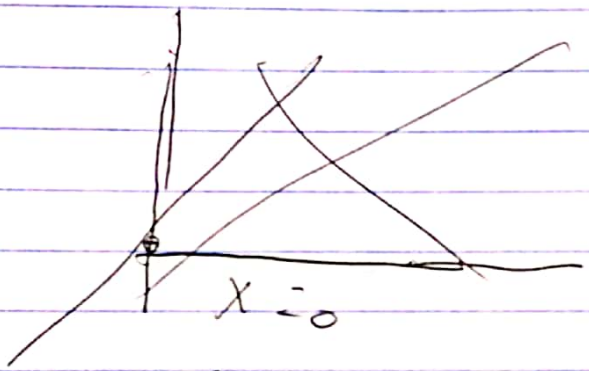


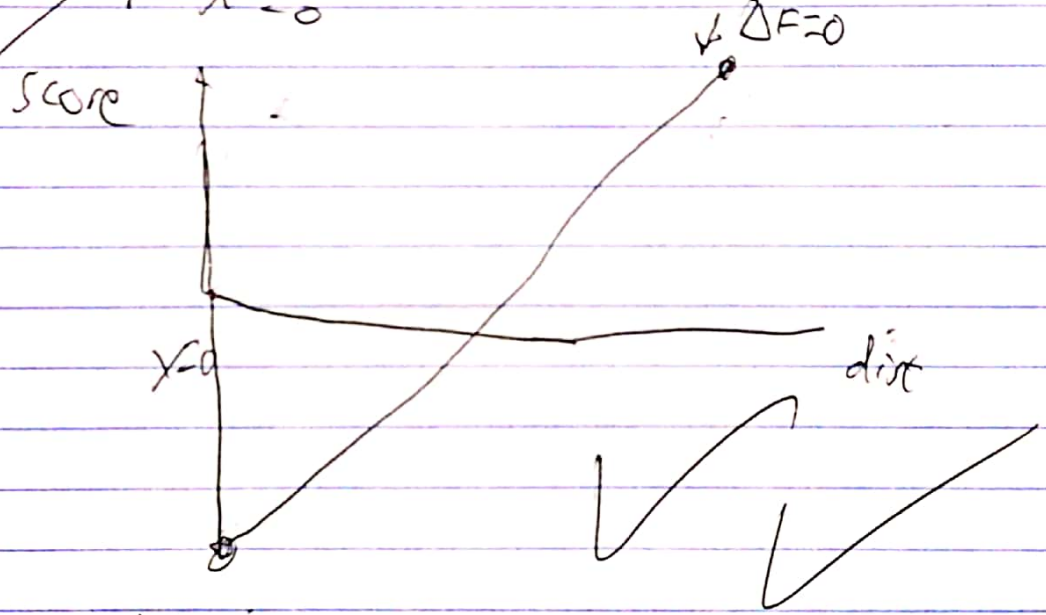
Δd Δf

ΔF

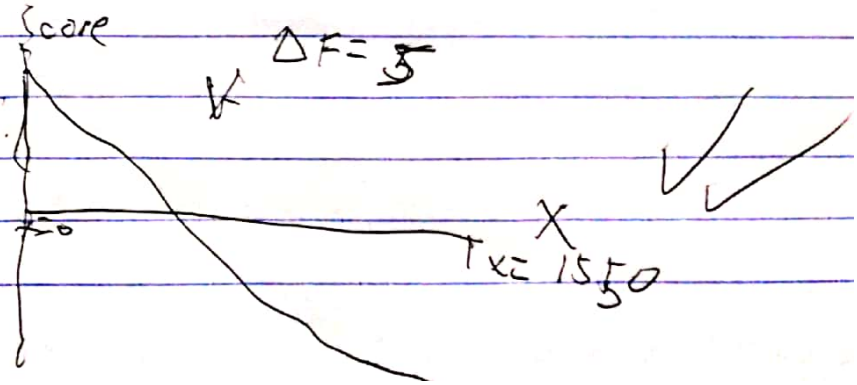
dif frequencies.







This works/model, it



No time for more :(

Notes: optimal freq based on my scoring method doesn't mean optimal :/

- If (Time != 0) {

  Could have done greedy algorithm;
  Could have tried clustering (K-means)?;
  }

~~End~~

If my method is bad then random freq is a solution based on scoring method?
since scoring method is my method

Solution Σ = random freq
∴ very time inefficient

End