Riley Campbell
CSC 372
Project 3

Non-code part

1. [20pt] A dendrogram (aka hierarchy clustering tree: https://en.wikipedia.org/wiki/Dendrogram) is a tree that joins nodes based on similarity, where each level joins two nodes.
   - **a.** [12pt] Describe, in pseudocode, how to make an algorithm using closest pair of points algorithm to make a dendrogram.
     - Let A be a list of points
     - Let M be a map where the key is the (x, y) cords of the center of a cluster and the value are the points in the cluster
     - While length of A > 1
       - a, b = Closest_Pair(A)
       - coord = Center(a, b)
       - M[coord] = a, b
       - A.remove(a, b)
       - A.add(coord)

   - **b.** [8 pt] What is the run time of your algorithm? Prove, **mathematically,** what would the run time be for your algorithm?
     - While length of A > 1 -> n
       - a, b = Closest_Pair(A) -> n*ln(n)
       - coord = Center(a, b) -> 1
       - M[coord] = a, b -> 1
       - A.remove(a, b) -> 1
       - A.add(coord) -> 1

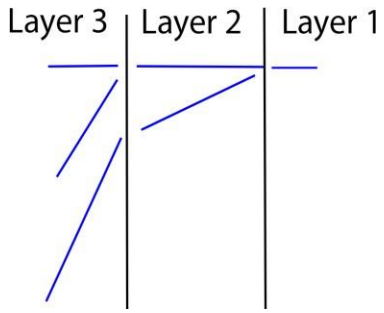
$n(n*\ln(n) + 1 + 1 + 1 + 1) = n*n*\ln(n) + n + n + n + n = n*n*\ln(n) + 4n = n*n*\ln(n) = n^2\ln(n)$

2. [20pt] The following can be solved greedily.

You are making the longest beam possible given a set of rods to be welded together. Each rod has a set length. For construction purposes, the rods are welded in layered fashion, and each layer must meet some criteria. The total summed length of rods in the i^{th} layer must be less than $(i + 1)^{th}$ layer. Also, for stability, the total number of rods in the i^{th} layer is less than the $(i + 1)^{th}$ layer.

For example, if rodLeng[] = { 15, 10, 20, 5, 25, 30 }, I could have this (one of many) answer:

Layer 1: 15
Layer 2: 20, 5
Layer 3: 10, 25, 30



a. [5 points] What is the optimal substructure (you must name the specific item being removed)?

- You remove the section that still needs to be checked. At each iteration i, i number of rods can be welded together if their sum is less than the sum of the next iteration of rods. Your split for `rodLeng[]` would be rodLeng(0 ... i) and rodLeng(i+1 ... rodLeng.length)

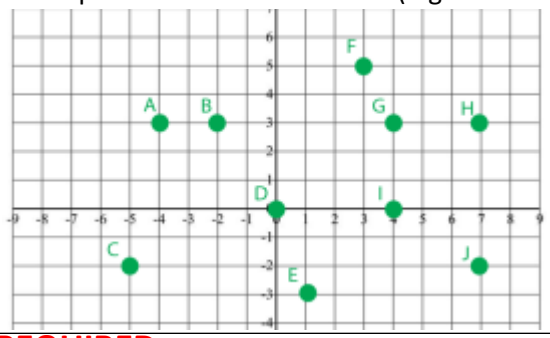
b. [6 points] What is the greedy property?

- Always take the first i indexes of rods where their sum is smaller than the sum of the next i + 1 index rods until there are no more rods to check against

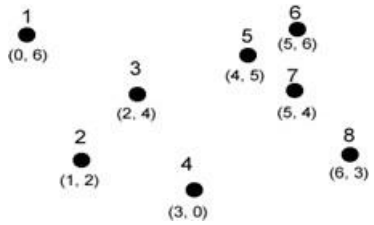
c. [9 points] Prove correctness of your algorithm. A proof of contradiction is acceptable.

- Let A = { 15, 10, 20, 5, 25, 30 }
- Let B = {} be a list of rod sets
- Assume the algorithm will not work
- i = 1
- 15 < 10 + 20
- Split on {15} | {10, 20, 5, 25, 30}
- B is now{{15}}
- A is now {10, 20, 5, 25, 30}
- i = 2
- 10 + 20 < 5 + 25 + 30
- Split on {10 + 20} | {5 + 25 + 30}
- B is now {{15}, { 10, 20}}
- A is now {5, 25, 30}
- I = 3
- 5 + 25 + 30 < nothing left to compare to
- B is now {{15}, { 10, 20}, {5, 25, 30}}
- A is now {}
- Return B
-
- Through proof by contradiction, the algorithm proved to be true

3. [35 points] Using the closest pair of points, for the following points, record what points (in order), are sent to both sides of the recursion. For the merge step, record the minimum selected from the left or right side, what set of points are considered when merging, and the selected points and their distances. All points are whole numbers (e.g. A is x = -4, not x = 4.2)



REQUIRED notation if not done in a drawn tree.



S is the set of point considered during the merge step:

Top: **X**= 1 2 3 4 5 6 7 8 **Y**= 1 6 5 3 7 2 8 4 (underline means they can be flipped)

- Left: **X** = 1 2 3 4 **Y** = 1 3 2 4
  - Left: **X**= 1 2 **Y** = 1 2
    - Base: **d**= ~4.1 **s**: (1,2)
  - Right: **X** = 3 4 **Y** = 3 4
    - Base: **d** = ~4.1 **s**: (3,4)
  - Merge:
    - Min returned: ~4.1 **split x** = 1.5
    - Checked points: 1 2 3 4
  - Returns: **d** = ~2.2 **s** = (2, 3)

- Right: **X** = 5 6 7 8 **Y** = 6 5 7 8
  - Left: **X**= 5 6 **Y** = 6 5
    - Base: **d**= ~1.4 **s**: (5, 6)
  - Right: X = 7 8 y = 7 8
    - Base: **d** = ~1.4 **s**: (7, 8)
  - Merge:
    - Min returned: ~1.4 **split x** = 5
    - Checked points: 5 6 7 8
  - Returns: **d** = ~1.4 **s** = (5, 6) OR (7, 8)
- Merge:
  - Min returned: ~1.4 **split x** = 3.5
  - Checked points: 4 5
- Returns: **d** = ~1.4 **s** = (5, 6) OR (7, 8)

Points[] = {A, B, C, D, E, F, G, H, I, J}

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

F = {3, 5} G = {4, 3} H = {7, 3} I = {4, 0} J = {7, -2}

S is the set of points to be considered during the merge step

Top **X** = C, A, B, D, E, F, G, I, H, J and **Y** = F, A, B, G, H, D, I, C, J, E. (__ means could be reordered)

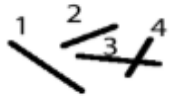
- Left: **X** = C, A, B, D, E **Y** = A, B, D, C, E
  - Left: **X** = C, A **Y** = A, C
    - Base: d = ~5.1 s = (C, A)
  - Right: **X** = B, D, E B, D, E
    - Left: **X** = B **Y** = B
      - Base: d = ~infinity s = (B)
    - Right: **X** = D, E **Y** = D, E
      - Base: d = ~3.2 s = (D, E)
    - Merge:
      - Min returned: ~3.2 **split x**: -0.5
      - Checked points: B, D, E
    - Returns: d = ~3.2 s = (D, E)
  - Merge:
    - Min returned: ~3.2 **split x**: -4
    - Checked points: C, A, B, D
  - Returns: d = ~2 s = (A, B)
- Right: **X** = F, G, I, H, J **Y** = F, G, H, I, J
  - Left: **X**; = F, G **Y** = F, G
    - Base: d = ~2.2 s = (F, G)
  - Right: **X** = I, H, J **Y** = H, I, J
    - Left: **X** = I **Y** = I
      - Base: d = ~infinity s = (I)
    - Right: **X** = H, J **Y** = H, J
      - Base: d = ~5 s = (H, J)
    - Merge:
      - Min returned: ~5 **split x**: 5.5
      - Checked points: I, H, J
    - Returns: d = ~3.6 s = (I, J)
  - Merge:
    - Min returned: ~2.2 **split x**: 5
    - Checked points: F, G, I, J
  - Returns: d = ~2.2 s = (F, G)
- Merge:
  - Min returned: ~2 **split x**: 0
  - Checked points: B
- Returns: d = 2 s = (A, B)

4. [35points] Using the any line segment intersect for the following line segments, record what is added/removed when, in the correct order, and when it returns true (no endpoint is vertically aligned(aka B is before C), so use a ruler).

C

Step	Add/remove	Segments (← above, below →)	Lines compared?	Intersection?
1	Add A	A	NA	N
2	Add B	B, A	A and B	N
3	Add C	B, A, C	B and C, A and C	N
4	Add D	B, D, A, C	B and D, D and A, D and C	N
5	Remove A	B, D, C	B and D, D and C	N
6	Remove B	D, C	D and C	N
7	Add E	D, C, E	D and C, C and E	N
8	Remove C	D, E	D and C	N
9	Remove D	E	NA	N
10	Add G	G, E	G and E	N
11	Add F	F, G, E	F and G, F and E	N
12	Remove E	F, G	F, G	N
13	Add H	F, H, G	F and H (assume return before checking the rest)	Y

Example of **REQUIRED** notation:



Step	Add/remove	Segments (← above, below→)	Lines compared?	Intersection?
1	Add 1	1	NA	N
2	Add 2	2, 1	2 and 1	N
3	Add 3	2, 3, 1	2 and 3, 3 and 1	N
4	Remove 1	2, 3	2 and 3	N
5	Remove 2	3	NA	N
6	Add **4**	3, **4**	3 and **4**	Y

You may abbreviate to a and r for add and remove

You **MUST** stop as soon as you hit an intersection, just like in the algorithm