Recap : Theory of heaps → represented using array ∴ available in STL / Collections

extractMin / Max again & again and in between you may optionally insert more elements.

Heap: [_ _ _ _ _]

heap.extractmin( )

heap.insert (x)

Example problem : rods = [3, 1, 2, 1, 2, 3, 4] rod lengths.

Solder these rods all into 1 rod. #cost (2 pieces) = x + y

x + y

Common idea: sorting

rods = [~~1~~ ~~1~~ ~~2~~ ~~2~~ 3 3 4]

2

4

6

9

12

16

cost = 2 + 4 + 6 + 9 + 12 + 16

cost = 49

Amazon OA 2021/20

90 mins → 2 questions.

Better approach : heap = {1, 1, 2, 2, 3, 4}

16

cost = 2 + 4 + 5 + 7 + 9 + 16

= 43 best !!

heap.size() > 1 :

u = extractmin() 1 } top 2 min elements

v = extractmin() 1

cost += (u + v)

heap.insert (u + v)

how to use heaps in lang? JS → no heaps. Java ~ C++

specific functions.

C++: priority_queue < data type , vector <data type> , comparator >

minheap → greater<data type>

maxheap → ×

int

vector <int>, vector <vector <int>>

not default

int, string, float.

1st element of array

not a f^n. custom sort

operator

=> operator overloading

bool mycompare (x, y)

if x < y : True

else false;

max heap by default.

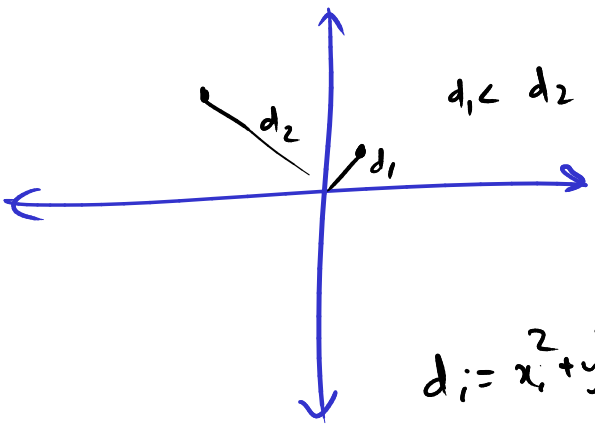
given k, k closest points to origin

min-heap . $x_i^2 + y_i^2$ ← makes a heap.

vector <int> = { $x_i^2 + y_i^2$, x_i, y_i }

overload operator/comparison

n points (x_i, y_i)



python: heapify().

pq.push() insert

pq.pop() extract

pq.top() → get min

heapq.heapify() extract min

heapq.heappop() ←

PriorityQueue < Integer > pq

pq.add → insert

pq.remove → removes min

pq.peek → get min