CS202 Course Projects

1) **Expression Parity**

Given *n* positive integers *a_1* to *a_n*, design an algorithm that determines if the following expression S is even or odd. Your algorithm must run in O(*n*) time.

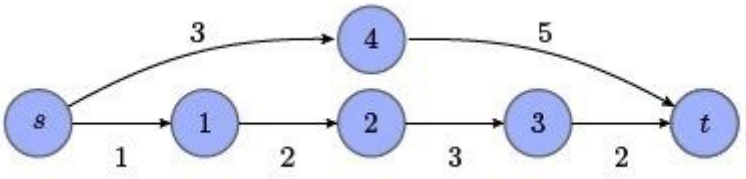
$$
\begin{aligned}
S \;=\; & 1 + a_1 + a_2 + \cdots + a_n \\
& + a_1a_2 + a_2a_3 + a_2a_4 + \ldots + a_2a_n + \ldots + a_{n-1}a_n \\
& + a_1a_2a_3 + \ldots + \text{all three combinations} \\
& + \ldots \text{and so on} \\
& + a_1a_2a_3 \ldots a_n
\end{aligned}
$$

For example if *a_1*=2, *a_2*=3 and *a_3*=1, then your algorithm must output even. (since S is 24).

2) **Shortest Short Path**

We have a graph whose weights are positive integers. Two vertices *s* and *t* are designated as source and target. Design an algorithm to find the shortest path with smallest number of edges from *s* to *t*. Or in other words, if there are multiple shortest paths from *s* to *t* pick the one with least number of edges.

In the example graph shown, there are two shortest paths of length 8. We need to pick the path s->4->t, since it has fewer edges than the path s->1->2->3->t.

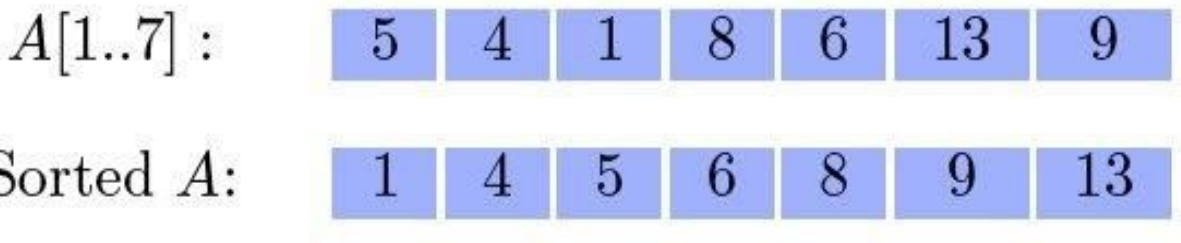


3) **Disturbed Array**

Suppose we take a sorted array and shuffle the elements a little bit, how fast can we get back the sorted array?

More precisely, you are given an array A[1..*n*] in which each element is atmost *k* positions away from the position it would have been if A was sorted.

Design an algorithm to sort the array in O(*n* log *k*) time. For example, A[1..7] shown below could be an input array for k = 2



4) **Adjacent Pair-Sum**

You are given an unsorted array *A*[1..*n*] and a number *t* as input. The problem is to find a pair of numbers in array A whose sum is *t*. The array satisfies the following property:

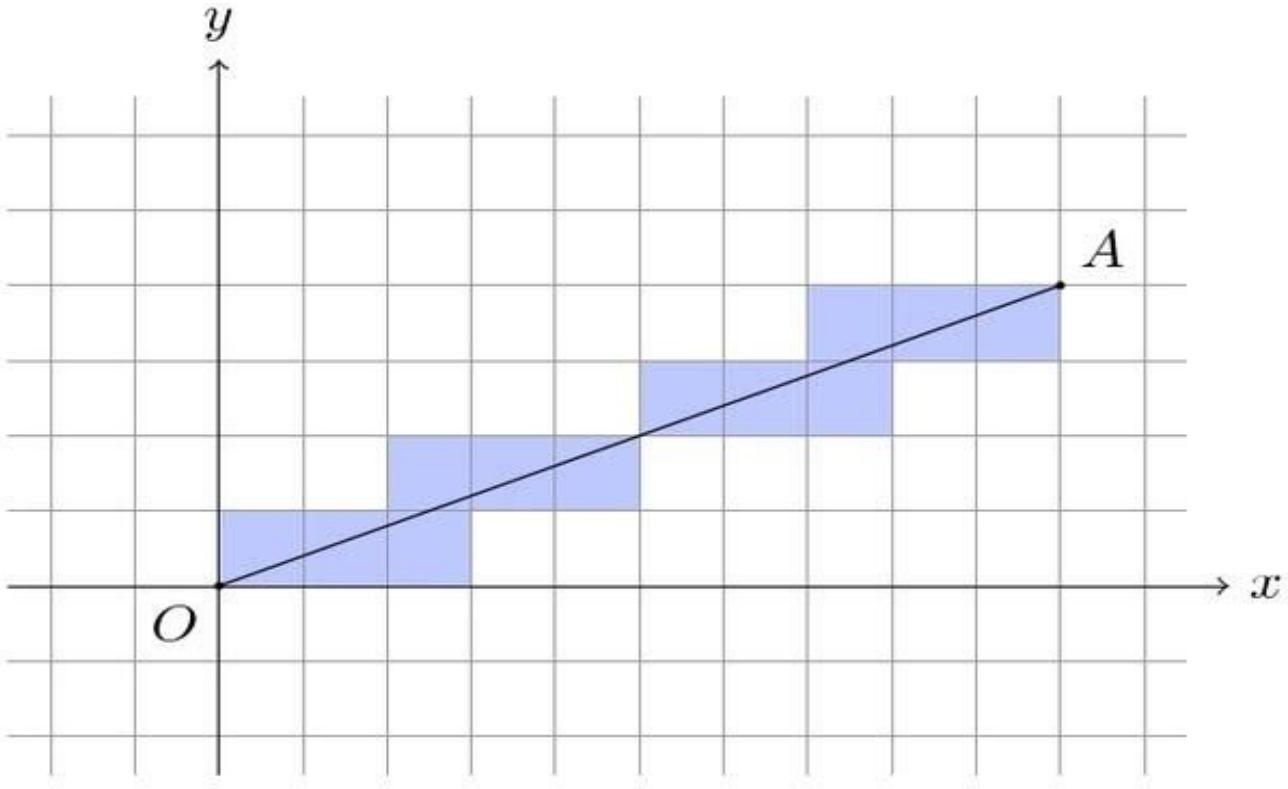
Property. If A[1..*n*] is sorted there will be two adjacent elements whose sum is t.

Devise an O(*n*) algorithm to find two numbers in the unsorted array A[1..*n*] whose sum is *t*.

5) **Cell Count**

In a co-ordinate plane, a line segment is drawn from the origin to a point A(*x*,*y*). Find, as efficiently as you can, the number of cells the line passes through. Assume that *x* and *y* are integers.

For example, the line segment from the origin to the point A(10,4) passes through 12 cells.

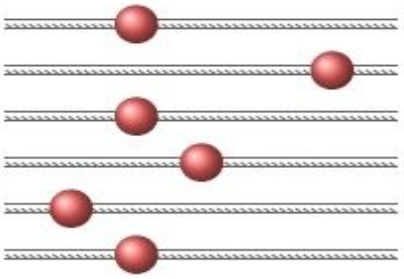


6) **Moving Beads**

We have *n* threads with each one having a bead. We want to line up all the beads vertically by moving them along the threads. The objective is to minimize the total distance moved.

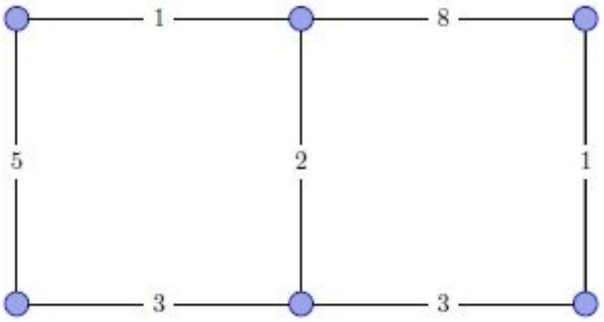
For example, we can align the beads shown by moving the second bead 3 units to the left; fourth and fifth bead by one unit to the left and right, respectively.

Given n numbers denoting the positions of beads, design a linear time algorithm to find the minimum total distance the beads have to be moved in order to line them up.



7) **Cycle Edges**

You are given a graph G with positive weights on edges. Pick a set S such that at least one edge from each cycle in G is included and the total weight of edges in S is minimized. In the example shown, it is best to pick the two edges of weight 1. Note that the graph has three cycles: two of length four and one of length six.



8) **Interval Sum**

Given an array of *n* positive numbers, design an O(*n*) algorithm to find three numbers in the array whose sum is between 1 and 2. If there are no such numbers, then the algorithm must report so

9) **Sum-free numbers**

A set is called *sum-free* if no element is the sum of all the other elements. More precisely, if $S=\{a_1,a_2,\ldots,a_k\}$, S is sum-free if:

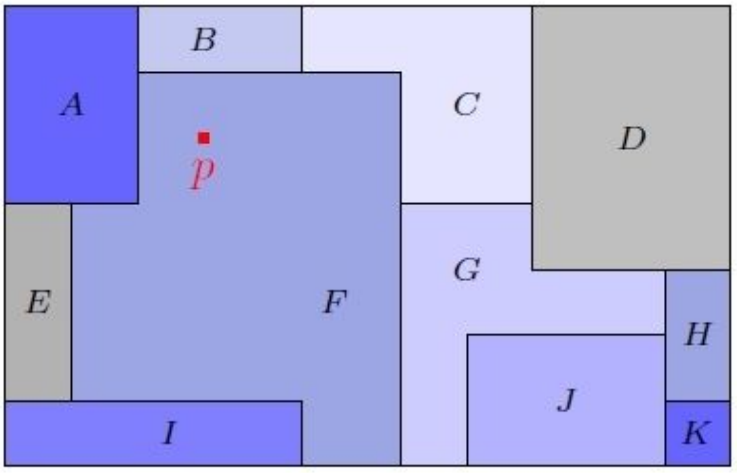
$$a_i \neq \sum_{\substack{e \in S \\ e \neq a_i}} e \text{ for all } i$$

Given an array *A* of *n* positive numbers, find the largest subset of *A* that is sum-free in O(*n*) time.

10) **Region Locator**

A rectangle is partitioned into a number of disjoint regions. All regions have axis-parallel boundaries. Build a data structure such that the following query can be answered in O(log *n*) time: Given a point *p*, which region does it belong to?

n refers to the number of boundary line segments. Assume any reasonable representation of the input. You need not explain *how* the data-structure is built, just describe what you will store and how you would answer a query.



11) **Distance Sum**

Given an array of *n* numbers, find a pair (a_i, a_j) that maximizes the following quantity. Your algorithm should run in O(*n*) time.

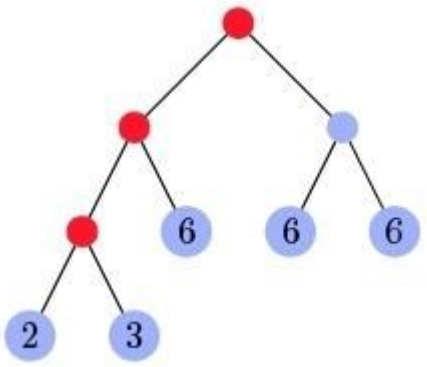
$$|a_i - a_j| + |i - j|$$

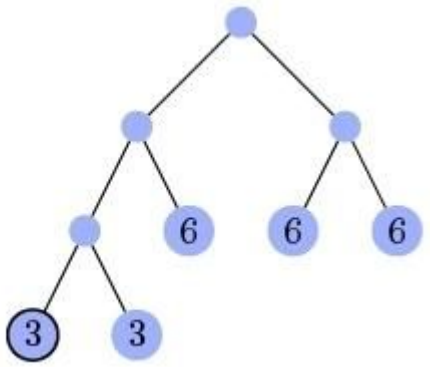
12) **Balancing a Binary Tree**

We have a binary tree in which each leaf is a number. An internal node is said to be *balanced* if the sum of its left subtree's leaves is the same as the sum of the leaves of its right subtree. In the figure shown, the red nodes are unbalanced, while the blue node is balanced.

Given an unbalanced binary tree find the minimum number of leaves whose value we must change so that the tree becomes balanced. In the example, changing 2 to 3 ensures balance.

Your algorithm should run in O(*n* log *n*) time, where *n* is the number of nodes in the tree. You may assume that any number you compute will fit into a constant-byte integer.





13) **Resilient Binary Search**

You are about to search for an element in a sorted array *A*[1..*n*] using binary search when the array suddenly gets *perturbed*. By perturbed, we mean a number in *i*th position in the array can now be either in *i*, *i-1* or *i+1*th position; but all the numbers are still in *A*[1..*n*]. Can you still search an element in *O*(log *n*) time in the perturbed array?

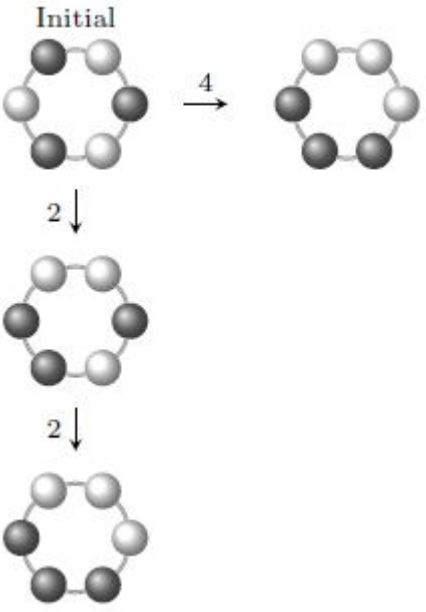
14) **Valid Strings**

A string over the characters (, [,] and) is said to be *valid* if one can reduce the string to the null string by repeatedly erasing two consecutive characters of the form () or []. For example, the string [([])[]] is valid but neither ([)] nor ([is valid.

Give a polynomial time algorithm to solve the following problem: Given a string, erase the smallest possible number of characters such that the remaining string after the erasures is valid. For example, given the string [(], we can erase (to get [].

15) **Beaded Necklace**

You are given a circular necklace containing *n* beads. Each bead maybe black or white. You have to rearrange the beads so that beads of the same color occur consecutively.

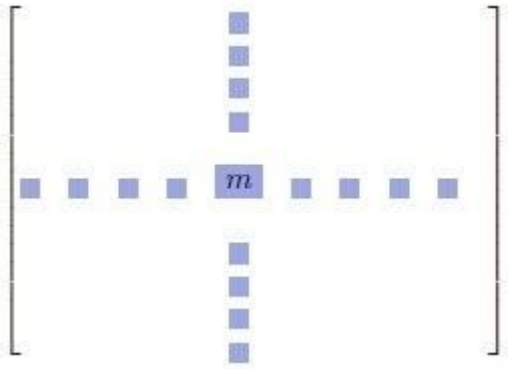


The only operation allowed is to cut the necklace at any point, remove some number of beads from both ends and put them back in any order, and join the cut ends. The cost of this operation is the number of beads removed. Any number of such operations can be used. You have to find a sequence of such operations with minimum total cost for a given initial distribution of beads.

For example, if the initial string is **wbwbwb**, this can be done by a single operation of cost 4, or two operations of cost 2 as shown. Describe a polynomial-time algorithm for this problem; with short proof of correctness.

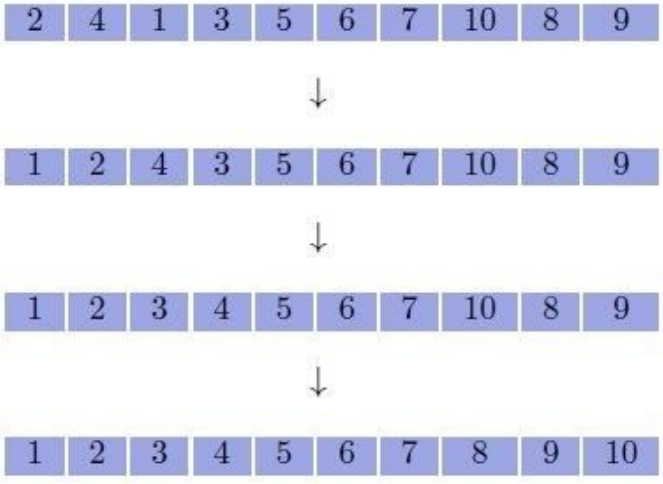
16) **Sorted Matrix**

We are given an *n*x*n* matrix and a number *t*. The problem is to search for the number *t* in the matrix. Both the rows and columns have entries in increasing order. Find an algorithm in $\theta(n^{\log 3})$ time.



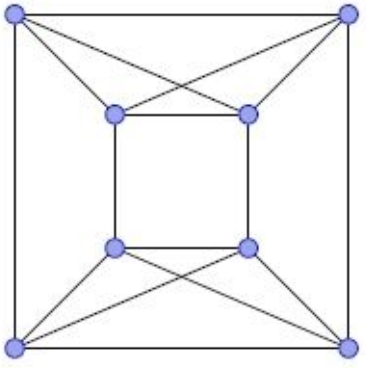
17) **Pick and Place**

We are given an array containing a permutation of numbers from 1 to *n*. In one *pick-and-place* operation we can pick a number and insert it into any position. Give an O(*n* log *n*) algorithm to find the minimum number of pick-and-place operations required to sort the array. For example, for the array given below we can pick-and-place 1 first and then 3 and 10. So the answer for this input is 3.



18) **Disjoint Cycles**

You're given a graph with n vertices, each with degree 4. Design an O(n) algorithm to partition the graph into vertex-disjoint cycles. For example, the graph shown aside can be partitioned into two cycles of length four (two squares).



19) **Gold Coins**

We have *n* gold coins out of which some coins are fake and the rest are genuine. The number of genuine coins is strictly greater than the number of fake coins. Coins of the same type have the same weight. We have a balance using which we can test if two coins have the same weight.

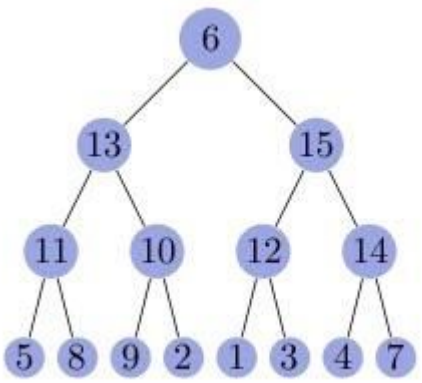
The problem is to find a genuine coin using as few weighings as possible. A trivial algorithm to solve the problem is to pick one coin and test it with every other coin. This uses *n-1* weighings. We want something slightly better. Design an algorithm that finds a genuine coin in *n-B(n)* weighings, where *B(n)* denotes the number of 1s in the binary representation of *n*.

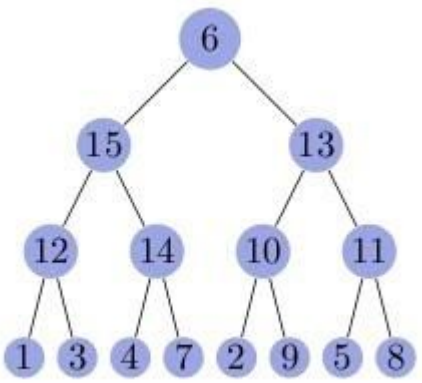
20) **Labelling a Binary Tree**

Suppose there is an undirected complete binary tree *T* whose labels we do not know. A node *x* is said to lie *in-between* node *u* and node *v* if deleting node *x* disconnects node *u* from node *v*. The problem is to output the tree *T* by asking queries of the form: **"**Does the node *x* lie in-between node *u* and node *v*?". Assume that each query can be answered in constant time by an oracle. We know the value of *n*, the number of nodes in the tree. The labels of the nodes are from *1* to *n*. The objective is to minimize the time taken to output the tree in terms of *n*.

Two trees are same with respect to isomorphism. For example, the two trees below are equivalent. The query **"**Does 10 lie between 15 and 9**"** would return YES while the query **"**Does 10 lie between 5 and 6?**"** would return NO.

Design an algorithm that outputs the tree *T* in O(*n* log *n*) time.





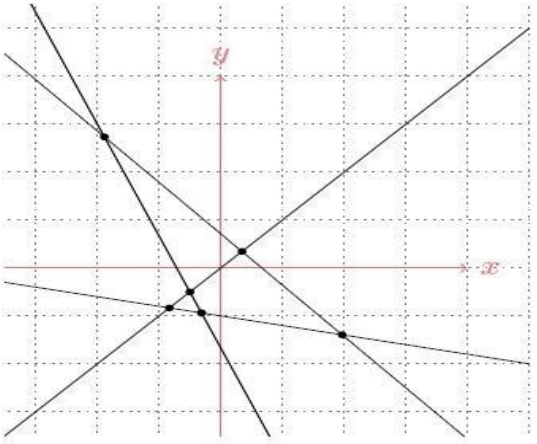
21) **Intersection Points**

You are given equations of *n* lines as input. The equation of a line is of the form *y=mx+c* where *m* is the slope and *c* is the y-intercept. Design an *O(n log n)* algorithm that counts the number of intersection points that lie on the right side of the line x=0 (y-axis).

You can make the following assumptions:

- No line is parallel to the x-axis or y-axis.
- No two lines are parallel to each other.
- No three lines intersect at the same point. Hence, there are exactly *n(n-1)/2* intersection points in the configuration.
- All the intersection points lie in the bounding box [-100,100]x[-100,100]. In other words, the x and y coordinates of any intersection point have an absolute value of at most 100.

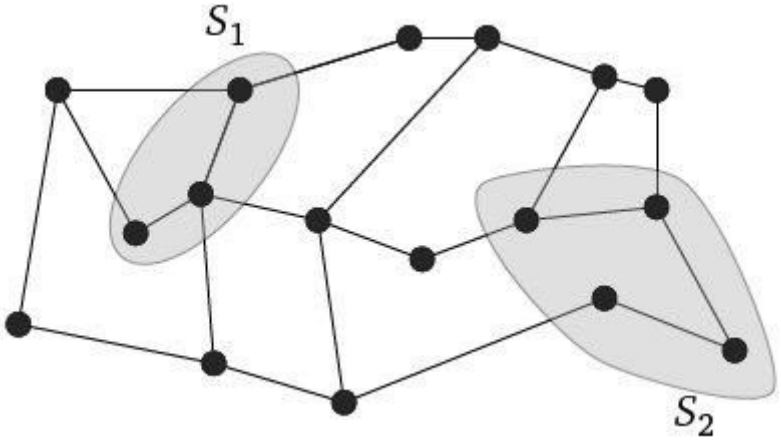
In the configuration shown aside, four lines give rise to six intersection points. Two intersection points lie on the right side of y-axis.



22) **Shortest path between sets**

Let *G* be a weighted undirected graph with *n* nodes and *m* edges. Along with *G*, we are given two disjoint node sets S_1 and S_2 as input. Both S_1 and S_2 are subsets of nodes from *G*. Find the shortest way to go from a node in S_1 to a node in S_2. More specifically, find the value of the following quantity:

min{ *dis(a,b)* : a ∈ S_1 and b ∈ S_2 }, where *dis(a,b)* denotes the shortest-path distance from node *a* to node *b*.



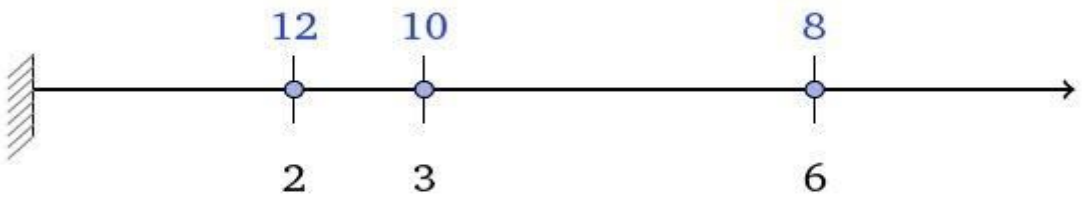
Let $f(n,m)$ denote the running time of the best algorithm for finding the shortest path between any two given nodes in G. A naive way of solving the above problem is to find the shortest path from every node in S_1 to every node in S_2 and take the minimum value among all the $|S_1||S_2|$ pairs. This could take as large as $O(n^2 f(n,m))$ time (which is bad). Give an algorithm that does something much better.

23) **Courier Delivery**

A courier person has to deliver packages to n locations. The courier office is located at the origin $(x=0)$ and the locations are at $x_1, x_2, \ldots, x_n$ with $x\_i>0$. The owner at location x_i is available only at time r_i i.e. the package for location x_i can be delivered only at time $t \geq r_i$. The courier person starts from the office $(x=0)$ and can deliver the packages in any order he wants. Assume that the person travels at unit speed i.e. it takes $|x_i - x_j|$ time to go from x_i to x_j. Also he is allowed to wait at any location for any amount of time.

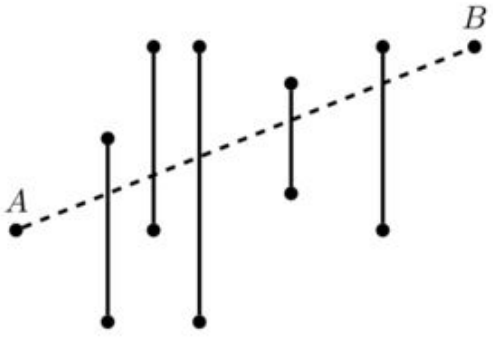
We would like to compute a delivery sequence such that the time taken by the courier person to deliver all the packages is minimized. Design an $O(n^2)$ time algorithm to compute such an optimal sequence.

Consider the following example, we have three owners at locations $x_1=2$, $x_2=3$ and $x_3=6$ units from the origin. Their r_i values are 12, 10 and 8 mins, respectively. The fastest way to deliver the couriers is to first go from the origin to x_3 wait there for two mins and then deliver to locations at x_2 and x_1. So your algorithm must output the sequence is x_3, x_2, x_1 for this input.



24) **Slicing line**

We are given as input n vertical line segments. Give a linear time algorithm to find a line segment AB that passes through all the vertical segments, if such a line exists. Each vertical segment of the input is given by a single x-coordinate and two distinct y-coordinates.



25) **Non-crossing chords**

Let *C* be a cycle with *2n* nodes. Every node in *C* has a positive weight associated with it. Node *i* has weight w_i. A *chord* is a line connecting two nodes of the cycle. The chord connecting nodes *i* and *j* has *value* $|w_i - w_j|$. Here $|x|$ denotes the absolute value of *x*.

Two chords are said to be *crossing* if they are either incident on the same node or if they intersect. Give an algorithm that takes as input the node weights $w_1 \ldots w_{2n}$ and outputs *n* non-crossing chords with maximum total value. The algorithm must run in linear time.

In the cycle shown in the figure, $w_1=6$, $w_2=3,\ldots,$ $w_8=2$. The dotted lines are the chords. The values of the chords, seen from left to right, are 6, 1, 3 and 4, respectively. The four non-crossing chords have a total value of 14 which is optimal.