**SUBJECT**: DESIGN AND ANALYSIS OF ALGORITHMS

**CODE**: 503040

Duration: 150 minutes

Allowed to use materials.

**LAB 03: Brute-force Algorithms**

# Objectives

Understand the properties of brute-force algorithm design technique

Be able to design, implement, and analyze brute-force algorithms solving common problems.

# Properties of brute-force algorithm design technique

Following Brute-force approach means we try to solve problems without worrying about the cost (running time, space). We directly solve the problem based on its statements, definitions of related concepts.

Brute-force approach provides a correct (usually ineffective) solution.

This type of algorithms is easy to understand and easy to implement as it is close to intuition.

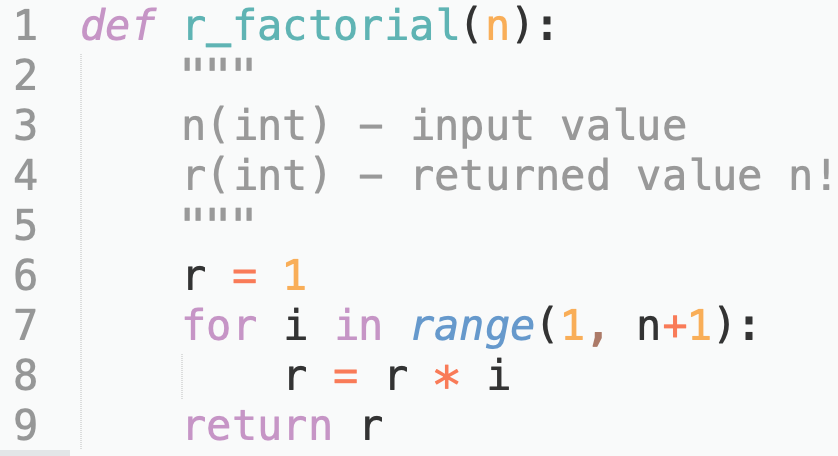
# Examples of problem solving with brute-force technique

## Example 1

### Factorial of a positive integer is defined as follows:

1. Design a brute-force algorithm (presented in Python) to solve the problem
2. Analyze the complexity of the proposed algorithm.

Python code



Analysis

1. Basic operation is multiplication on line 8

2. The worst case is the average case too, because the algorithm runs the same in all situations.

3. The total number of basic operations is:

4. So the complexity of the algorithm is

## Example 2

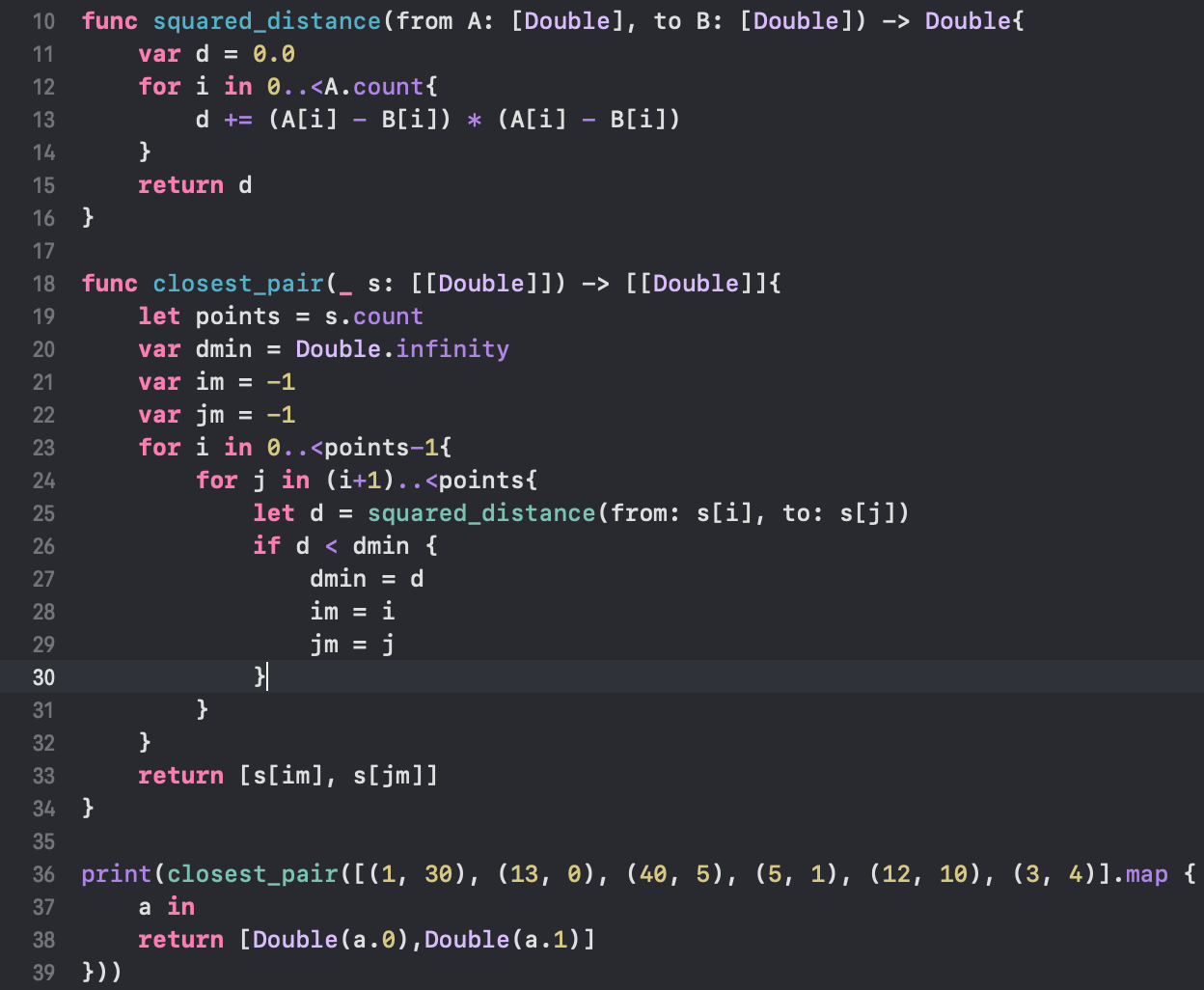
In the space with dimension, the Euclidean distance between two points and is determined by the formula:

Suppose there are points in the space.

Propose a brute-force algorithm to find the **closest pair** of points.

1. Design a brute-force algorithm (presented in Swift) to solve the problem
2. Analyze the complexity of the proposed algorithm.

A program to find the **closest pair** of points



The output of the program: [[5.0, 1.0], [3.0, 4.0]]

Analysis

Function: squared\_distance:

1. Input size is the number of dimensions of points (d)

2. Basic operation is multiplication on line 13

3. There is no worst case

4.

Function closest\_pair:

1. Input size is a tuple of , where n- the number of points, d – number of dimensions.

2. The most time-consuming part is the call to squared\_distance() on line 25.

3. There is no worst case

4. Count:

when , ops

when , ops

…

when , ops

# Exercises

For each of the problems in this section:

1. Design a brute-force algorithm (presented in Java) to solve the problem.
2. Analyze the complexity of the proposed algorithm.

Warm-up problems

### Exponential power

Exponential power of a positive integer is defined as follows:

### Combination

Combination is determined by the following formula

(example)

### Matrix multiplication

Multiply two square matrices and of size n

More challenging problems

### Nearest pair (closest pair)

In the Space with dimension , the Euclidean distance between two points and is determined by the formula:

Suppose there are points in the space. You are required to find the **closest pair**.

### Convex hull

A shape S is convex if for any points P, Q in the shape, every point in the line joining P and Q is also in S. That is, for all λ with 0 ≤ λ ≤ 1: λP + (1 − λ)Q ∈ S.

Convex Hull of a set of points (at least three). Smallest convex shape S which contains the points.

That is, for all convex S′, which contain the points, S ⊆ S′.

Theorem: For any finite set of points, Convex Hull is a convex polygon, and its vertices are included in the set of points given

Hence, we just need to find the extreme pairs of points. The polygon formed using the line segments joining these pair of points will give the convex hull.

Extreme: All other points are on the same side of the line joint the pair of points.

For ease, we assume no triplets of points are colinear (at least not in the boundary of the convex hull).

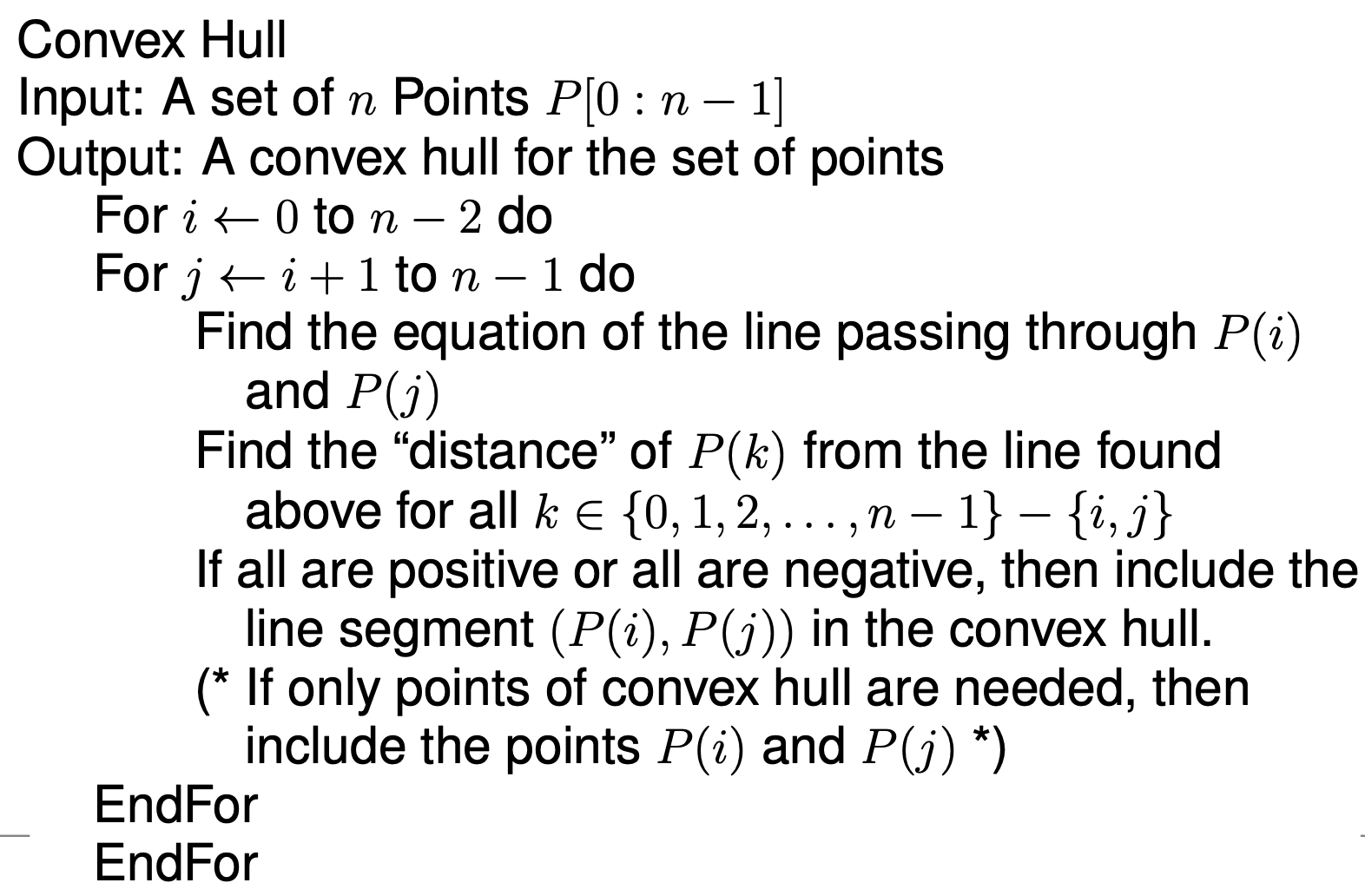
Line passing through

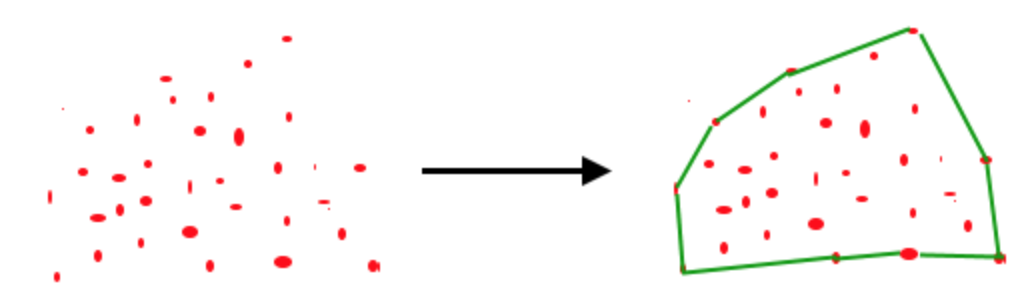
Rearrange to put it in form .

Distance of a point from a line is given by

Count the number of times line equation is evaluated, and distance is calculated.

Hints:





(Image from geeksforgeeks.org)

A testcase from geeksforgeeks.org

points[0]=new Point(0, 3);

points[1]=new Point(2, 3);

points[2]=new Point(1, 1);

points[3]=new Point(2, 1);

points[4]=new Point(3, 0);

points[5]=new Point(0, 0);

points[6]=new Point(3, 3);

convex hull is

(0, 3)

(0, 0)

(3, 0)

(3, 3)

### Travelling Salesman Problem

Input a weighted graph (undirected). Find a simple circuit which goes through all the vertices and has minimum weight.

Hints:

Brute Force/Exhaustive approach:

For each possible order of the vertices (there are of them!):

Find the weight of the circuit formed when the vertices are traversed in that order. Then find the one with minimal weight.

### Knapsack Problem

A set of items, each having weight and value , and a knapsack size .

Find a subset , such that and is maximised.

Hints:

Exhaustive approach: Consider all possible subsets of (there are of them!).

For each subset as above, check if , and if so this is a feasible set.

Among all feasible sets, choose the one which maximizes .

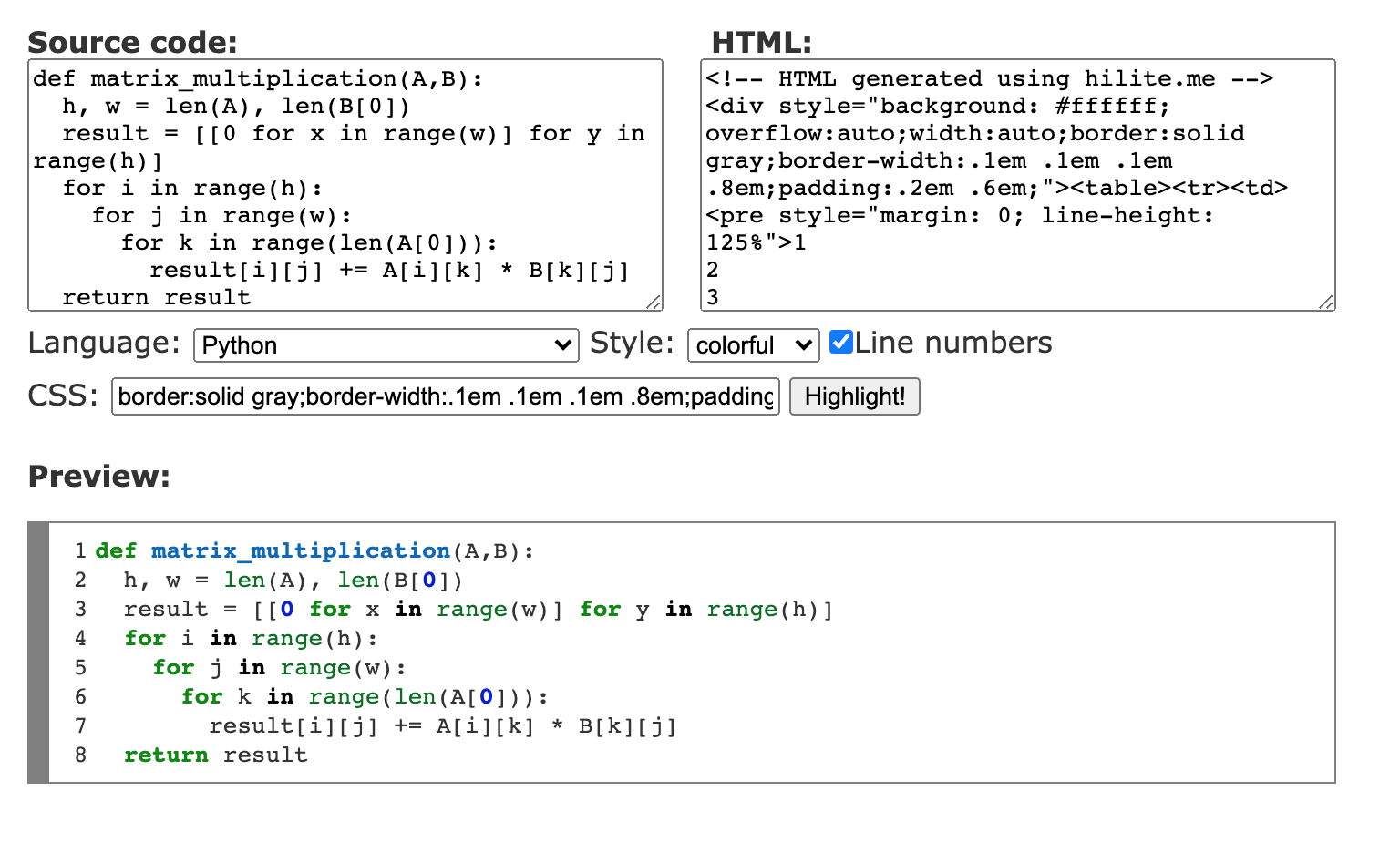
Technical help:

How to add line numbers to code

|  |
| --- |
| def matrix\_multiplication(A,B):  h, w = len(A), len(B[0])  result = [[0 for x in range(w)] for y in range(h)]  for i in range(h):  for j in range(w):  for k in range(len(A[0])):  result[i][j] += A[i][k] \* B[k][j]  return result |

<http://hilite.me/>

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | **def** **matrix\_multiplication**(A,B):  h, w = len(A), len(B[**0**])  result = [[**0** **for** x **in** range(w)] **for** y **in** range(h)]  **for** i **in** range(h):  **for** j **in** range(w):  **for** k **in** range(len(A[**0**])):  result[i][j] += A[i][k] \* B[k][j]  **return** result | |



Appendix:

, where const

, by definition