Basic

(1) Answer each part True or False, and briefly justify your answer.

- $2n = O(n)$
- $\log_{10} n = O(\log_2 n)$
- $n^2 = O(n)$
- $n^2 = O(n \log^2 n)$
- $n \log n = O(n^2)$
- $3^n = O(2^n)$
- $n! = O(n^n)$

Solution

- $2n = O(n)$. True, $(2n)/n = 2$.
- $\log_{10} n = O(\log_2 n)$. True, because

$$\frac{\log_{10} n}{\log_2 n} = \frac{\ln n / \ln 10}{\ln n / \ln 2} = \frac{\ln 10}{\ln 2}.$$

  Recall that $\log_b x = \ln x / \ln b$ where $\ln x$ is the *natural logarithm* (Base $e = 2.718281828459\ldots$).
- $n^2 = O(n)$. False, $n^2/n = n \to \infty$ as $n \to \infty$.
- $n^2 = O(n \log^2 n)$. False, $n^2/(n \log^2 n) = n/(\log n)^2 \to \infty$ as $n \to \infty$.
- $n \log n = O(n^2)$. True, $(n \log n)/n^2 = (\log n)/n \to 0$ as $n \to \infty$.
- $3^n = O(2^n)$. False, $3^n/2^n = (3/2)^n \to \infty$ as $n \to \infty$ because $3/2 > 1$.
- $n! = O(n^n)$. True, because as $n \to \infty$ we get

$$\frac{n!}{n^n} = \frac{n \cdot (n-1) \cdots 2 \cdot 1}{n \cdot n \cdots n \cdot n} = \underbrace{\frac{n}{n}}_{=1} \cdot \underbrace{\frac{n-1}{n}}_{<1} \cdots \underbrace{\frac{2}{n}}_{<1} \cdot \underbrace{\frac{1}{n}}_{<1} \to 0.$$

(2) Given $f(n) = O(n^2)$ and $g(n) = O(n^3)$, what is the order of $f(n) + g(n)$, $f(n)g(n)$ and $f(g(n))$.

Solution

$f(n) + g(n) = \max\{O(n^2), O(n^3)\} = O(n^3)$

$f(n)g(n) = O(n^2) \times O(n^3) = O(n^{2+3}) = O(n^5)$

$f(g(n)) = O\left(O(n^3)^2\right) = O(n^{3\times 2}) = O(n^6)$

(3) Design an algorithm that, given a list of numbers, discovers if any number has occurred more than twice. (No need to write pseudocode – just the main idea.)

What is its cost? (Use O-notation).

Hint: There is an algorithm that costs $O(n^3)$ and a better one that only costs $O(n \log n)$.

Solution

- First, sort the list using a fast algorithm costing $O(n \log n)$. We then read the sequence from start to end keeping track of any repeated elements and their number of repetitions, which costs $O(n)$.

  So the total cost is $O(n \log n) + O(n) = O(n \log n)$, which is polynomial.

  For example, $[4, 6, 1, 4, 3, 8, 7, 4]$ when sorted gives $[1, 3, 4, 4, 4, 6, 7, 8]$. We can then easily deduce that there is only one element that is repeated more than twice, namely: 4.

- The $O(n^3)$ solution involves nested loops to compare all possible triplets to see if they are equal.

  For clarity here is pseudocode for this:

  Input: A list of numbers $A = [x_1, \dots, x_n]$.
  Output: The set of numbers that are repeated more than twice in A.

```
1:  repeated ← ∅
2:  for i ← 1, ..., n do
3:      for j ← i + 1, ..., n do
4:          for k ← j + 1, ..., n do
5:              if x_i = x_j = x_k then
6:                  Add x_i to repeated
7:              end if
8:          end for
9:      end for
10: end forreturn repeated
```

  The loops at lines 2, 3, and 4 each repeat for a maximum of n times. The check at line 5 costs $O(1)$, and the operation at line 6 can be done in time $O(1)$. (Ask yourself: *How?*)

  The total maximum time is therefore $n \times n \times n \times O(1) = O(n^3)$.

NB.

- The time for *sequential* parts is the *sum* of the individual times.
- The time for *nested* parts is the *product* of the individual times.

(4) A **triangle** in an undirected graph is a 3-clique. Define the language

$$TRIANGLE = \{\langle G\rangle \mid G \text{ contains a triangle}\}$$

Show that $TRIANGLE \in \mathbf{P}$.

Solution

Input: A graph $G = (V, E)$.
Output: True if G contains a triangle, and False otherwise.

```
1: for each triplet a, b, c from V do
2:     if (a, b), (b, c) and (c, a) are valid edges from E then
3:         return True
4:     end if
5: end for
6: return False
```

The loop goes over $\binom{n}{3} = \frac{n!}{3!(n-3)!} = \frac{1}{6}n(n-1)(n-2) = O(n^3)$ possibilities, and the check in line 2 costs $O(1)$.

So the total cost is $O(n^3) \times O(1) = O(n^3)$.

> **PS.** The number of choosing k elements from n elements is
>
> $$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{1}{k!} \cdot n \cdot (n-1) \cdots (n-k+1).$$
>
> We read this as "n choose k." You can learn more about it at https://en.wikipedia.org/wiki/Combination

(5) A **Hamiltonian path** in a directed graph is a path that goes through each vertex exactly once.

$$HAMPATH = \{\langle G, s, t\rangle \mid \text{Directed graph } G \text{ has a Hamiltonian path from } s \text{ to } t\}.$$

Show that $HAMPATH \in \mathbf{NP}$.

Solution

A possible *certificate* to use is a feasible Hamiltonian path from s to t.

To verify we:

1) Verify that the path contains all the vertices of G.
2) Verify that the edges are valid, i.e. they really exist in G.
3) If both pass *accept*; otherwise *reject*.

For the cost, we have:

- Step 1 costs $O(n)$ where n is the number of vertices in G.
- Step 2 also costs $O(n)$ because there are $n-1$ edges in the given path.
- Step 3 costs $O(1)$.

So the total cost is $O(n) + O(n) + O(1) = O(n)$, which is polynomial.

(6) We say that two graphs G and H are **isomorphic** if the vertices of one of them can reordered to make it identical to the other (i.e. their adjacency matrices become the same).

Define the language

$$ISO = \{\langle G, H\rangle \mid G \text{ and } H \text{ are isomorphic graphs}\}$$

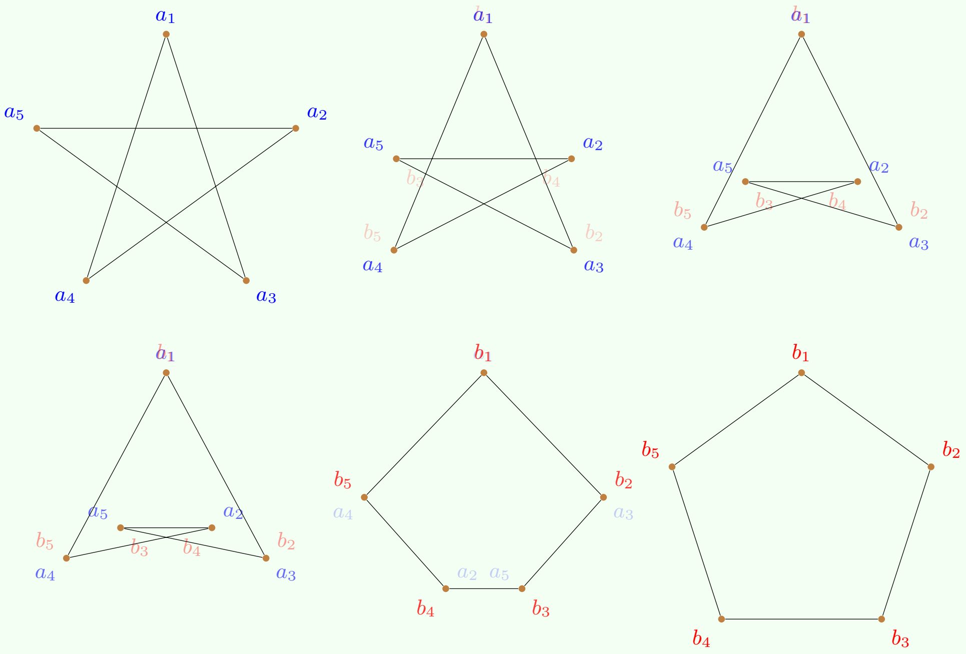
Show that $ISO \in \mathbf{NP}$.

Solution

Let $G = (E, V)$ and $H = (G', V')$ be the two graphs given by their sets of vertices (E and E') and edges (V and V').

- As a first check, the two graphs must have the same number of edges and vertices, i.e.
$$|E| = |E'| \qquad \text{and} \qquad |V| = |V'|,$$
otherwise they would clearly not be isomorphic.
- If G and H are indeed isomorphic then a possible *certificate* can be given as a *renaming map* that tells us how to match/pair the vertices of the two graphs.

For example, the top-left star graph below can be morphed into the bottom-right pentagon graph as follows:



The renaming map of the vertices is given by:

$$\begin{array}{ccc} a_1 & \mapsto & b_1 \\ a_2 & \mapsto & b_4 \\ a_3 & \mapsto & b_2 \\ a_4 & \mapsto & b_5 \\ a_5 & \mapsto & b_3 \end{array}$$

- Given such a certificate, we then have to check that the edges for each pair of vertices match. That is to say: if the vertices $a_i, a_j \in V$ have an edge between them, then their corresponding vertices $b_k, b_\ell \in V'$ must also have an edge between them.

  You may think that we ought to also check that if a_i and a_j are not connected then b_k and b_ℓ are not either, but it is not needed because: $|E| = |E'|$ implies that the previous check is sufficient. (Convince yourself!)

  > We could have written the above more technically as follows.
  >
  > Let v be a given *bijective map* between the vertex sets V and V'. Then we require the following map between E and E' to also be bijective:
  >
  > $$\underbrace{(a_i, a_j)}_{\text{Edge from } E} \longmapsto \underbrace{(b_k, b_\ell)}_{\text{Edge from } E'} = \Big(\underbrace{v(a_i)}_{b_k}, \underbrace{v(a_j)}_{b_\ell}\Big).$$

- Let us now estimate the cost of these checks, assuming that the various properties of the graphs are efficiently implemented.
  - The first check about the sizes can be done in time $O(1)$.
  - To check the validity of the edges:
    (1) we iterate over each edge $(a_i, a_j) \in E$,
    (2) we map it to $(b_k, b_\ell) = (v(a_i), v(a_j))$,
    (3) and then check that this is indeed in E'.

    This costs:

    $$|E| \times \Bigg(\underbrace{2 \times O(1)}_{\text{for } v(a_i), v(a_j)} + \underbrace{O(\log |E'|)}_{\text{Binary search in } E'} \Bigg) = O(|E| \cdot \log |E|),$$

    because $|E| = |E'|$. (You may find it useful to study the table at: https://en.wikipedia.org/wiki/Big_O_notation#Orders_of_common_functions)

  We conclude that the total cost is $O(1) + O(|E| \cdot \log |E|) = O(|E| \log |E|)$, which is polynomial as required.