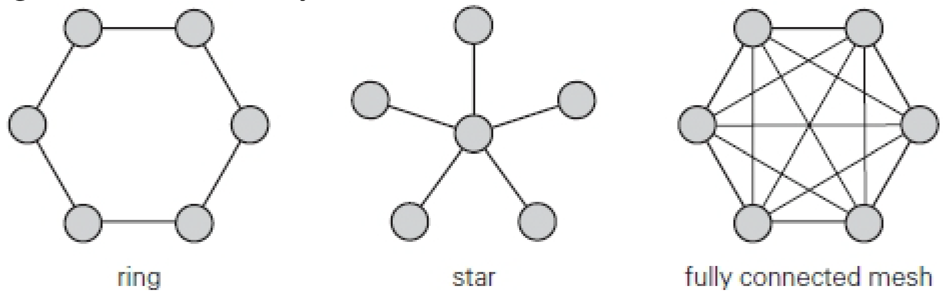
CSCE 321, Dr. Goneid
Exercises (2) Brute Force Algorithms

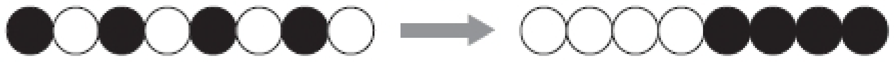
1. A network topology specifies how computers, printers, and other devices are connected over a network. The figure below illustrates three common topologies of networks: the ring, the star, and the fully connected mesh.



You are given a boolean matrix A[0..n− 1, 0..n − 1], where n > 3, which is supposed to be the adjacency matrix of a graph modeling a network with one of these topologies. Your task is to determine which of these three topologies, if any, the matrix represents. Design a brute-force algorithm for this task and find out its complexity.

---

2. **Alternating disks:** You have a row of 2n disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those that interchange the positions of two neighboring disks.



Design an algorithm for solving this puzzle and determine the number of moves it takes.

<u>Solution:</u>
Represent black by "1" and white by "0" then build an array A of 2n alternating "1" and "0" representing the alternating disks. We will use <u>Bubble Sort</u> with a swap flag as in the slides.

<u>Algorithm (with swap flag):</u>
```
for i = 2n-1 downto 1
{
   swapped = false
   for j = 0 to i-1
        if ($A_j > A_{j+1}$) {swap($A_j$ , $A_{j+1}$); swapped = true}
  if (not swapped) return
}
```
Find the number of moves T(n) by experimenting with an example (e.g. take n = 4 so that A = [10101010], or n = 3 so that A = [101010])
You will notice that the number of disk moves (swaps) in each of the (i) loop iterations will be n, then n-1 then n-2, etc with one move at the end.
Hence, $T(n) = n + (n-1) + ... + 1 = n (n+1)/2 = O(n^2)$

---

3. Consider the problem of searching a string of text T[0 .. n-1] for a substring that matches a pattern P[0..m-1]. The Brute Force algorithm is:

```
ALGORITHM  BFStringMatch(T[0..n − 1], P[0..m − 1])
for i ← 0 to n-m do
    j ← 0
    while j < m and P[j] = T[i + j]
        j ← j + 1
    if j = m return i
return -1
```

Show that the worst case number of comparisons is *O(mn)*

---

4. The Closest Pair Problem in 1-D:

- Consider a set of n points on one line. Their distances from the origin are stored in a random fashion in an array X[0..n-1]. A Brute Force algorithm would be:

```
$D_{min} = \infty$
for i = 0 to n-2
    for j = i+1 to n-1
        if ($D_{ij} < D_{min}$) { $D_{min} = D_{ij}$ ;  K = i; L = j;}
return K , L, $D_{min}$
```

  Find the number of comparisons made by this algorithm as a function of n.

- Another Brute Force algorithm would first sort the X-array, so that the closest pair would come one after the other. Then a scan through the sorted array would find the closest pair.
  Write such algorithm and find the number of comparisons made by this algorithm as a function of n, taking the number of comparisons done by the sorting algorithm as *O(n log n)*

---

5. The Hamming distance between two strings of equal length is defined as the number of positions at which the corresponding symbols are different. It is named after Richard Hamming (1915–1998), a prominent American scientist.
Suppose that the points in the problem of closest pairs are strings and hence the distance between two of such strings is now measured by the **Hamming Distance.** Write an algorithm to find the closest pair of strings in an array of strings and find the number of comparisons done by this algorithm.

Solution:

Consider two strings A, B, both of the same length ***m*** characters. A function to compute the Hamming Distance between them would be:

```
int HD (string A , string B, int m)
{
    int D = 0;
    for i = 0 to m-1
        if ( $A_i \neq B_i$) D = D + 1;
    return D;
}
```

The number of comparisons made by this algorithm is ***T(m) = m***
Given an array of strings S[0..n-1] and if we use the first algorithm for the closest pair as in the previous problem, then

```
$D_{min} = \infty$
for i = 0 to n-2
    for j = i+1 to n-1
        $D_{ij} = HD(S[i], S[j])$    // m comparisons
        if ($D_{ij} < D_{min}$) { $D_{min} = D_{ij}$ ;  K = i; L = j;}       // one comparison
return K , L, $D_{min}$
```

Hence,

$$T(n) = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}(m+1) = (m+1)\sum_{i=0}^{n-2}(n-i-1) = (m+1)\sum_{i=1}^{n-1} i = (m+1)n(n-1)/2 = O(mn^2)$$

---

6. Given a fast computer that makes 10 billion operations per second. Estimate the time required to compute the solution of each of the following problems using Brute Force algorithms:

- Traveling Salesman Problem (TSP) with 20 cities.
- The 0/1 Knapsack problem with 20 items.
- The Assignment problem with 20 persons and 20 jobs
