# **Design Document**

# Runtime Analysis of Every Method Used:

Function: display\_trainers

Analysis: O(n2)+c

Function: show\_all

Analysis: O(n) + O(n) => O(n)+c

Function: display\_recruit\_list

Analysis: O(n2) + O(n2) + c => O(n2)+c

**Algorithm Chosen**

Adjacency Matrix is chosen as the algorithm implementation strategy as the lookup needs to be done fast and assign weights for each edge.

### Adjacency Matrix

* Uses O(n^2) memory
* It is fast to look up and check for presence or absence of a specific edge  
  between any two nodes O(1)
* It is slow to iterate over all edges
* It is slow to add/delete a node; a complex operation O(n^2)
* It is fast to add a new edge O(1)

## **Alternative Approach:**

### Adjacency List

* Memory usage depends more on the number of edges (and less on the number of nodes),  
  which might save a lot of memory if the adjacency matrix is sparse
* Finding the presence or absence of specific edge between any two nodes  
  is slightly slower than with the matrix O(k); where k is the number of neighbour nodes
* It is fast to iterate over all edges because you can access any node neighbour directly
* It is fast to add/delete a node; easier than the matrix representation
* It fast to add a new edge O(1)