50005 CSE Lab 2

Computational Complexity of Banker's Algorithm

The main complexity of the banker's algorithm in this exercise lies within the `checksafe()` function, which is called during a request of resources through `requestResources()` by some process.

The first section initializes the temporary arrays that the algorithm will use to check the new state of system, which are `temp_avail[]`, `temp_need[][]`, `temp_allocation[][]`, `work[]`, and finally `finish[]` which is used to verify that each individual process is will not cause a deadlock in this new state. Given n customers and m resources, the resulting complexity of this series of iterative for-loops is is:

$$\mathcal{O}(2nm + 2n + m)$$

assuming the assignment operation is $\mathcal{O}(1)$ each for N elements, which is $\mathcal{O}(N)$ overall.

In the final section that carries out the actual safety check, a while loop is used to iterate a single for-loop of n elements over a maximum possible of n times (if all processes are considered safe in the next state i.e `checkSafe()` returns `true`). This for-loop also contains a comparison operation which compares two different arrays of m elements each, which is implemented with a nested for-loop. If the condition passes, then another array assignment operation is carried out. Thus, the *overall computational complexity of this implementation of the banker's algorithm is:

$$\mathcal{O}(2nm + 2n + m + 2n^2m) = \Theta(n^2m)$$

**Formula ignores some simple $\mathcal{O}(1)$ assignment and comparison operations.*