# A) Problem Statement

The Goal of the problem? Literally, provide an algorithm to determine the survivor in a Josephus game. Figuratively, to teach us about runtime complexity.

# B) Algorithm Design

The implementations are identical for both a list and vector. Both walk to the elimination location, remove that person, then add it to a queue of those eliminated. The elimination location advances by the number of passes, and is modded to wrap the indices to the size of the circle. The only difference between the two implementations is the internal structure of adding and removing. The list adds to the tail of the linked list, whereas the vector writes to the ‘end’ of the array. Similarly, the list removes a node from the list, and is done, whereas the vector removes the item, then fills the hole in the array.

# C) Experimental Setup

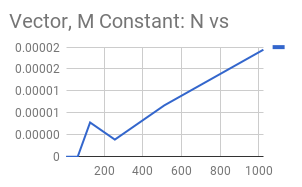
My experiment was done in Visual Studio 2017, running on windows 10, with a 2.7 Ghz with 16 gigs of RAM. The experiment was performed twice for both cases, the “better” sample was chosen for each.

For each experiment, the game is played to completion. The execution function is wrapped in a timing function which returns both the person and the time elapsed. The time elapsed is added to a running total of time, and the elimination function is incremented.

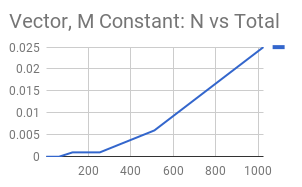
The running total, in addition to M, N, the eliminations, and a calculated average, is plugged into a plot data structure to cache the results for later. This cache is then written to a file, named after the constant variable.

# D) Experiment

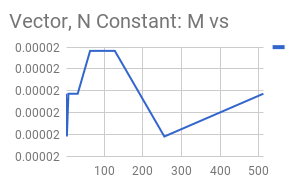
Looking at Vectors, N vs Average helps show our average case, which appears to be N, linear. There does seem to be an outlier somewhere around 128, but 256, 512, and 1024 seem to follow a linear pattern.



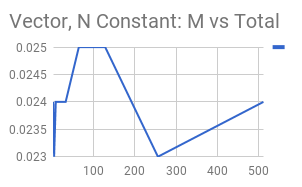
N vs Total runtime however, shows that the algorithm appears to be N2, quadratic. Assuming 128 is an outlier, the interpolation between our smaller numbers and 256 coincides with the pattern we’d expect in a quadratic function. I expect this to be worst case, considering a vector could potentially remove in N time. For example, approach kth position, remove, then advance to N while moving the next item into position.



M vs Average runtime, while the graph looks like a mess, looking at the scale, this appears to be C, Constant. These fluctuations all round out to 0.00002 seconds.



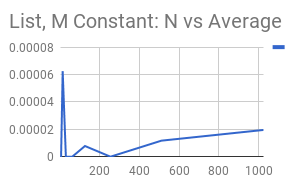
M vs Total runtime, again, while the graph looks like a mess, looking at the scale, this appears to be C, Constant. These fluctuations all round out to 0.024 seconds.



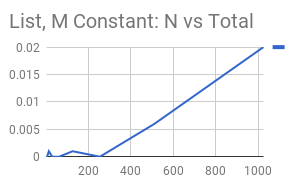
Overall, the Vector looks like it is dependent on N, the number of people, which is to be expected. There is little variance when N is constant and M fluctuates. Looking at the M constant graphs, I would conclude the vector’s implementation is O(N2) and Ω(N).

Looking at List, N vs Average helps show our average case, which appears to be Log(N) or N0.5.

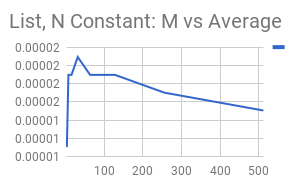
There is a large outlier near 0, and another outlier at 256, but the scale seems to asymptote before 0.00004. This would imply Log(N), however, due to our limited dataset, this may not be the case.



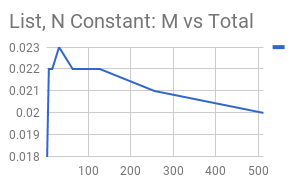
N vs Total runtime however, shows that the algorithm appears to be N, linear. I expect this to be tight case, considering a list removes in place, after I have already traversed to the point to remove.



M vs Average runtime, while the graph looks like a mess, looking at the scale, this appears to be C, Constant. These fluctuations all round out to 0.00002 seconds.



M vs Total runtime, again, while the graph looks like a mess, looking at the scale, this appears to be C, Constant. These fluctuations all round out to 0.02 seconds.



Overall, the List looks like it is dependent on N, the number of people, which is to be expected. There is little variance when N is constant and M fluctuates. Looking at the M constant graphs, I would conclude the list’s implementation is O(N) and Ω(Log(N)).

# E) Discussion

Looking at the two implementations, list looks like the clear winner. Which is to be expected as vector must perform additional overhead. It does not appear to matter whether we are varying M or N.

Comparing their variance on M, List performs slightly faster, but it looks rather negligible. Save for extremely large M, there isn’t a major performance difference between the two.

Comparing their variance on N, List performs a degree faster, averaging something looking like log N, beating vector’s N. On worst case, list looks like N, vector appears to be N2.

## 