CS201: Data Structures

Name: Tarun Singla	**Entry Number:** 2019CSB1126
Submitted to: Dr. Puneet Goyal	

In this report, I have analyzed the time complexity of Johnson's algorithm for All Pair Shortest Path considering different data structure based heap implementation.

Johnson's Algorithm

Johnson's algorithm is used to find all pair shortest paths for a given graph using the Bellman-ford algorithm and the Dijkstra algorithm.

Intuitively, all pair shortest paths can be found by applying the Dijkstra algorithm from each node. But since the Dijkstra algorithm is only applicable for graphs with positive edges, we can not use it directly. So, we first convert the graph in such a way that all the edge weights become non-negative, and the path weight of any path between any two vertices changes by the same amount. Now, we can apply the Dijkstra algorithm from each vertex to find all pair shortest paths.

To convert the graph, we first add a node to the graph and connect it to all nodes with edge weight equal to zero. Then we apply the Bellman-ford algorithm from this vertex and find the shortest paths for all other vertices.
Now, using these shortest path values, we update each edge in the original graph as follows:

$$w(u,\ v)\ =\ w(u,\ v)\ +\ h[u]\ -\ h[v]$$

where $w(u,\ v)$ is the edge weight of the edge from u to v, and $h[u]$ and $h[v]$ are the shortest paths found from the Bellman-ford algorithm.

The change in path weight of any path between two vertices u and v will always be $h[u]\ -\ h[v]$ because all the other intermediate $h[]'s$ will cancel each other.

Also, since $h[v]\ \leq\ h[u]\ +\ w(u,\ v)$, therefore, the updated edge weight will always be non-negative.

To get the real shortest path between any two vertices, we do as follows:

$$s = s + h[v] - h[u]$$

where s is the shortest path from u to v found using the Dijkstra algorithm.

Time Complexity

Johnson's algorithm involves applying the Bellman-ford algorithm once and that of the Dijkstra algorithm V times where V is the number of vertices. The Bellman-ford algorithm's time complexity is $O(VE)$, where E is the number of edges.

The time complexity of the Dijkstra algorithm depends on the type of heap used. In general, Dijkstra's algorithm involves V insert operations, V extract-min operations, and at most E decrease-key operations (as per aggregate analysis). Here, I will be analyzing this based on four types of heap implementation:

1. Array-Based
2. Binary Heap
3. Binomial Heap
4. Fibonacci Heap

The various time complexities associated are:

	Insert	Extract-min	Decrease-key
Array-Based	$O(1)$	$O(n)$	$O(1)$
Binary Heap	$O(\log n)$	$O(\log n)$	$O(\log n)$
Binomial Heap	$O(\log n)$	$O(\log n)$	$O(\log n)$
Fibonacci Heap*	$O(1)$	$O(\log n)$	$O(1)$

(where n is the number of nodes, *amortized)

Therefore, for Dijkstra, the time complexities considering all the three operations are as follows:

1. Array-Based: $O(V^2)$
2. Binary Heap: $O(E \log V)$
3. Binomial Heap: $O(E \log V)$
4. Fibonacci Heap: $O(V \log V + E)$

where E and V are numbers of edges and vertices, respectively.

The final time complexity of Johnson's algorithm considering various implementations of heap data structure in the Dijkstra's algorithm is as follows:

Implementation	Time Complexity
Array-Based	$O(V^3)$
Binary Heap	$O(E\ V\ log\ V)$
Binomial Heap	$O(E\ V\ log\ V)$
Fibonacci Heap	$O(E\ V\ +\ V^2\ log\ V)$

Therefore, theoretically, Fibonacci heap based implementation should perform the best, followed by Binomial heap and Binary Heap, and Array-based implementation should perform the worst.

Also, as the graph becomes denser and denser, the time complexity of Johnson's algorithm becomes $O(V^3)$ (or worse), which is equal to that of the Floyd-Warshall algorithm used for finding all pair shortest paths.

Practical Analysis

The results of the practical analysis do not follow directly with the theoretical analysis. It is observed that the Array-based implementation takes the most time, which is also observed in theoretical analysis. Binomial heap based implementation takes lesser time than this, and Fibonacci heap based implementation takes even lesser, which follows directly from the theoretical analysis. The difference that we got from that of the theoretical analysis is that the Binary heap based implementation takes the least time.

Although the time complexity of Binary heap based implementation and Binomial heap based implementation is similar, even then, the Binary heap based implementation takes lesser time. It could be because of the Binomial heap's complexity compared to the Binary heap and the higher number of operations involved in it.

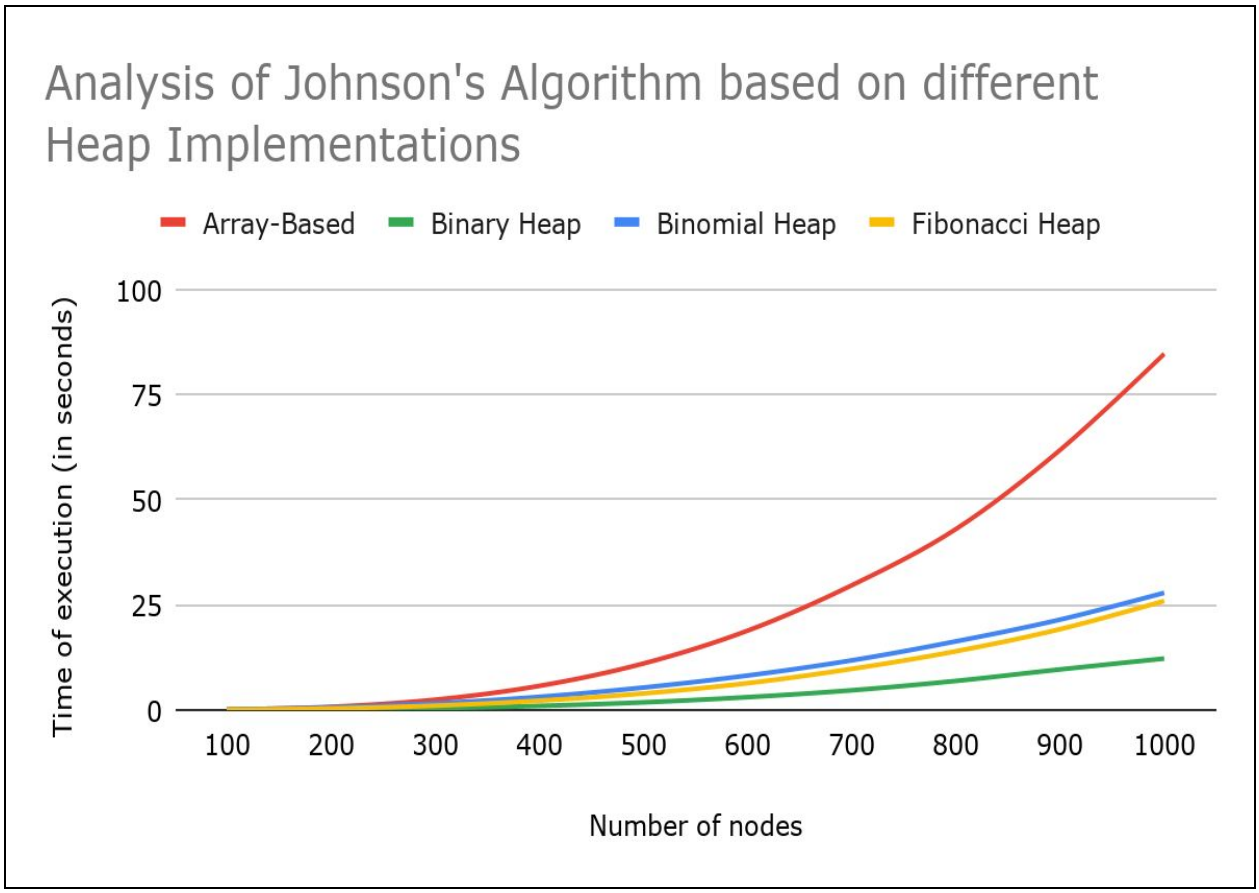
Fibonacci heap based implementation, which was supposed to take the least time as per theoretical analysis, also took more time than the Binary heap based implementation.

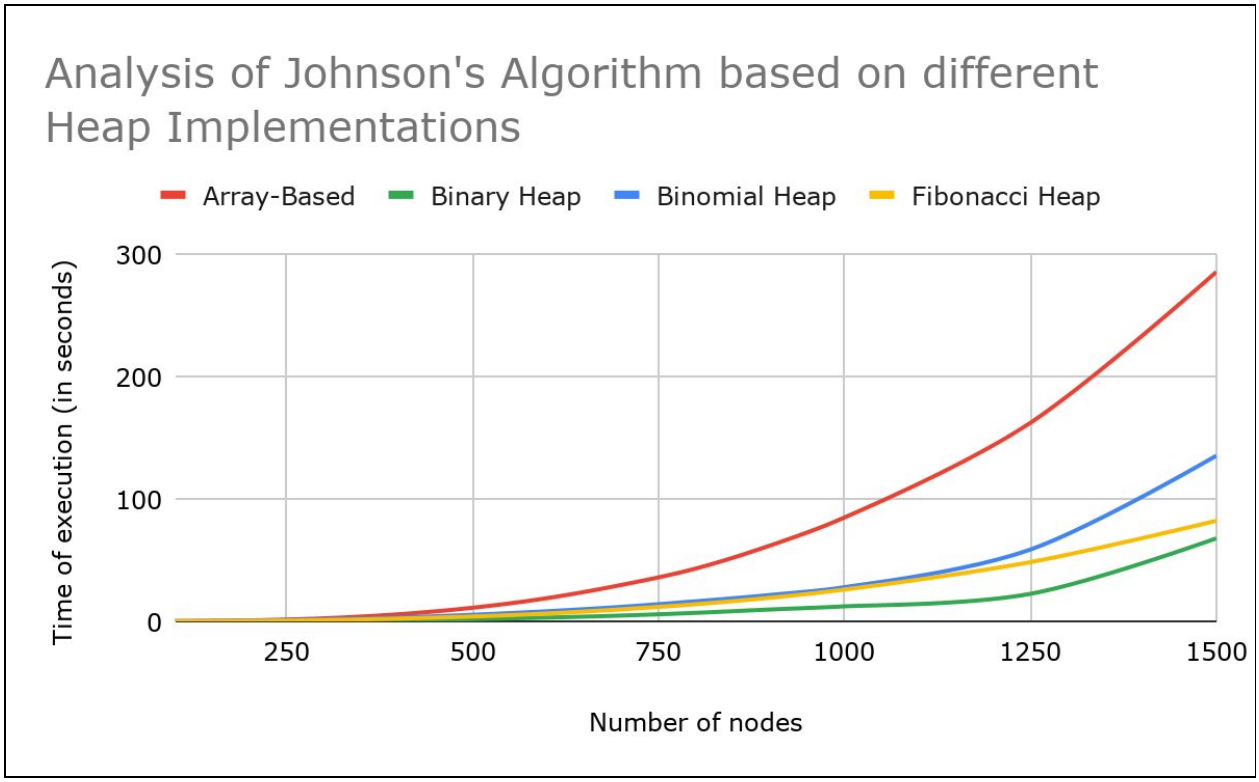
The main reason behind this could be the higher complexity of the implementation, where each node is associated with four pointers. The constant factors in the algorithm took more time than expected. Also, the extract-min operation has a linear running time in the worst case. In general, it only looks good in theory and has been designed for theoretical purposes only.

Graphical Analysis

Here are some graphs/plots showing Johnson's algorithm's execution time using different data structure based heap implementations. The graphs used were random, with the probability of an edge between two nodes varying between 0.4 to 0.8. For a particular number of nodes, at least five graphs were analyzed, and the time of execution was averaged to maintain generality.

Here, it should be noted that the time of execution may vary depending on the machine and the state of the machine used. We are just using these for the purpose of comparison.





Conclusion

We conclude that the Binary heap based implementation takes the least time, followed by Fibonacci heap based implementation, then, Binomial heap based implementation, and lastly array-based implementation. Although the theoretical analysis suggests that Fibonacci heap based implementation should take the least time, but in reality, it is not the best implementation for real-time systems due to the high hidden constant factors. It just suggests so theoretically. To obtain high performance, we should use Binary heap based implementation.