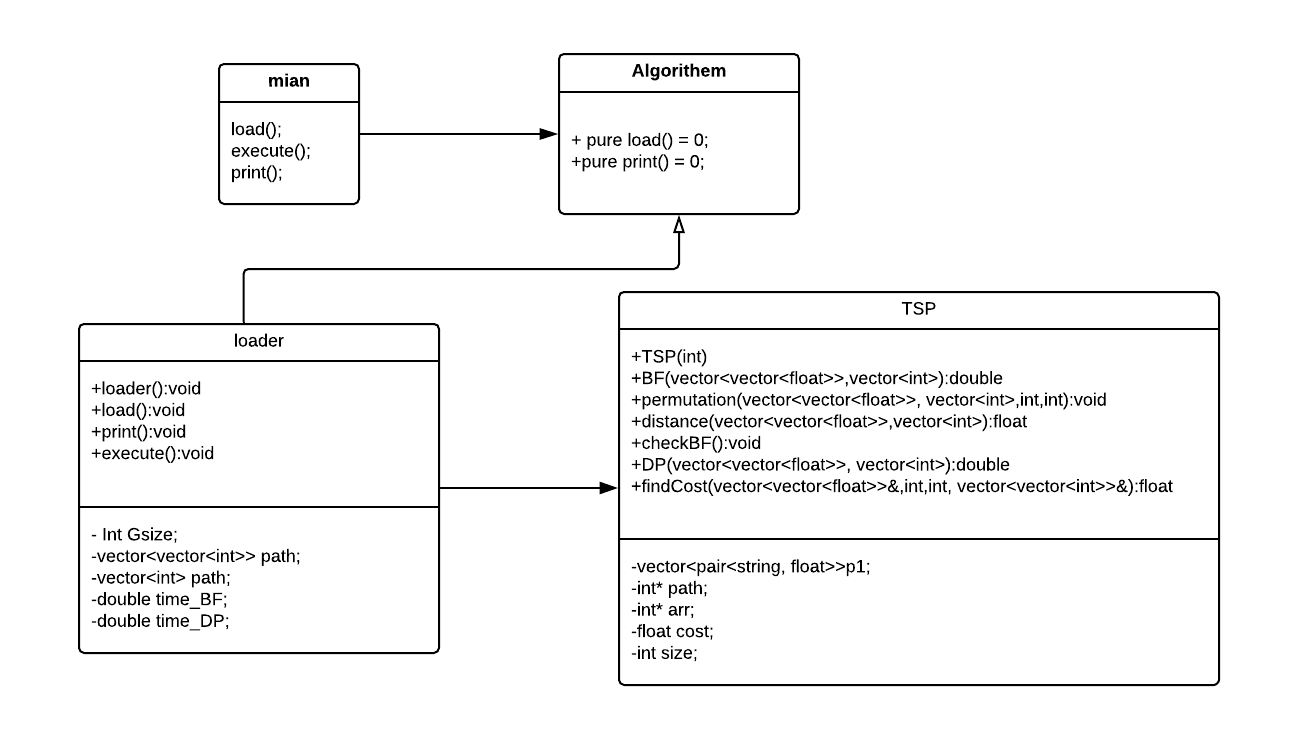
Report3

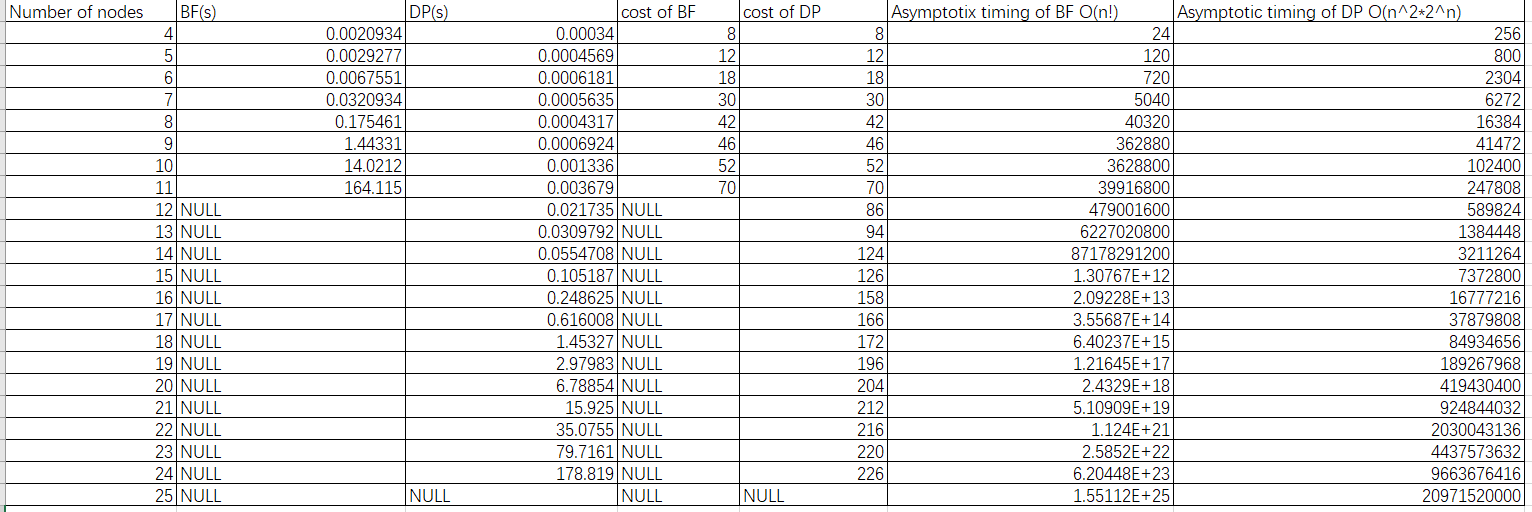
1. Project Architecture Design:

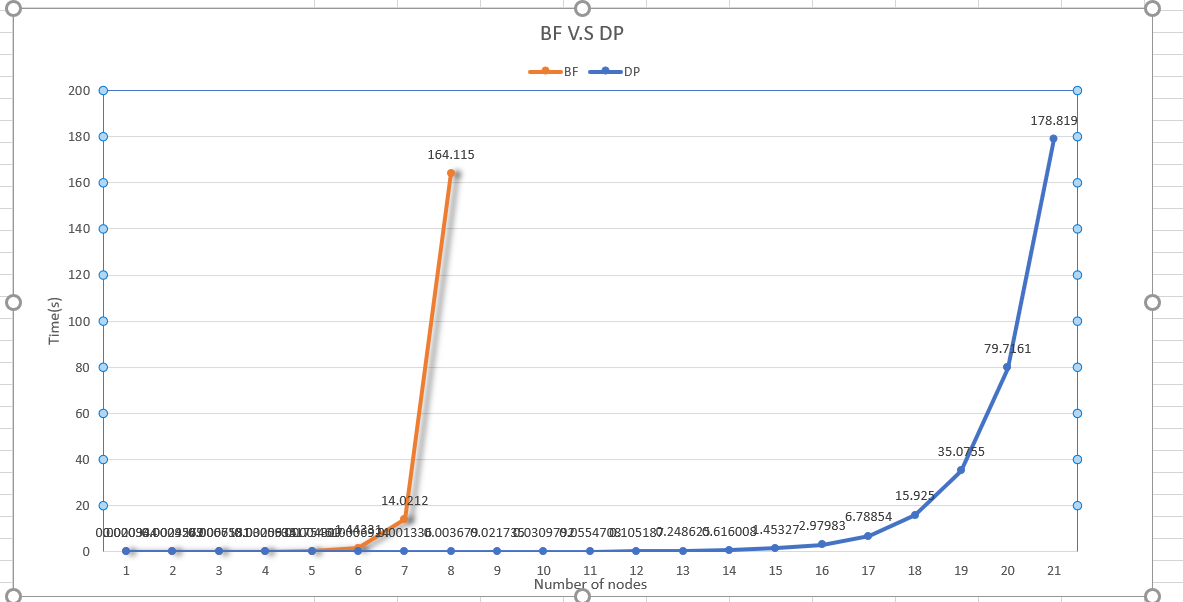


From the above UML diagram, I use strategy pattern to model my code like Lab1. In main.cpp, I only need to create a loader object and call all the necessary functions in an elegant way. It avoids me to put amount of codes in the main which let me understand clearly and easy to reimplement later. I also create two pure virtual function in Algorithm.h ,using pure virtual is a way to remind me overwrite the “load” and “print” function in the loader (which will inheritance from the Algorithm class) and in the case, I can put output system and loader system in a single interface. Loader is the class that I will direct create a loader object in main file and it like a connection connect the functional class and the main class. If I need to add any new functions in TSP class, I can simply call the function in the execute() in loader. The loader, at the same time, including the TSP class, which can call the TSP function in loader and all the function in TSP can share the same load and print function in loader because all the function in TSP need to print their results and read the input file. In conclusion, the strategy pattern can cut my whole program into small pieces, each piece has own utilization difference from others. It avoids me to put ton of codes into same class which is not easy to reread and reimplement.

1. Data Summary:

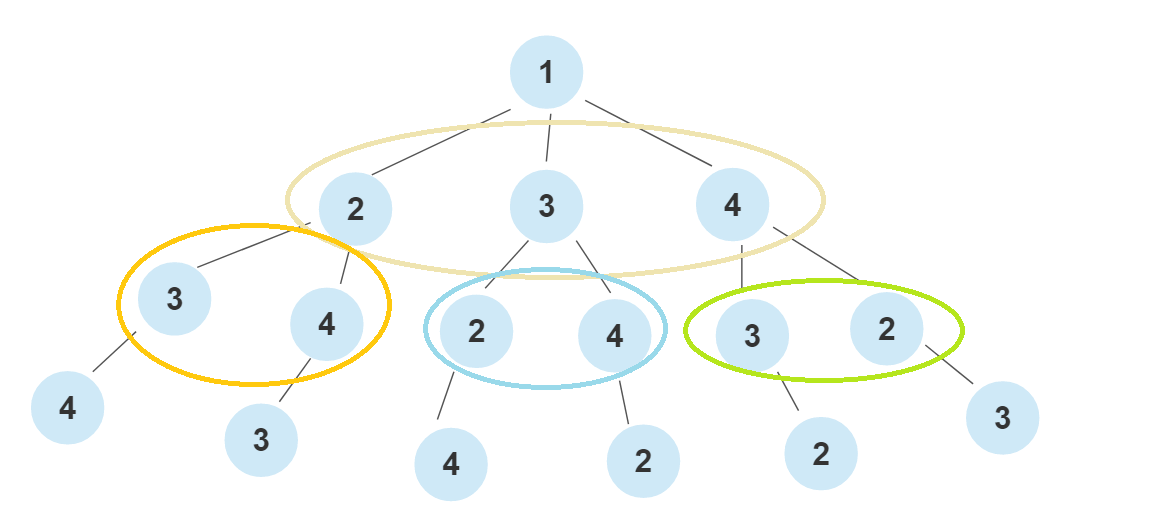
**!!(BF: naive brute force method) && (DP: dynamic programming)**





In the BF v.s DP graph, the orange line is BF timing curve and the blue is the DP timing curve. As we can see in the graph the BF and DP timing curve are overlap to each other before 5’s nodes, and the BF curve have slight increase in node 6 and 7. However, after 7 nodes, the curve increase sharply and finally when the nodes become 11, my computer crashed and return the location error message which mean there are not more place to run the BF method. For the DP timing curve, this curve remains a little change before 15’s nodes and have a high increment after nodes number become 19 and end at the 24 nodes. The difference of two curve is because those two algorithms have two different time complexity.

In the Code execution v.s time complexity graph, the yellow line represents the O(n!) and the blue line represents the O(n^2\*2^n). Those two curves show the exact same increasing trend as the curves in BF v.s DP graph which is the O(n!) curve will increase faster than O(n^2\*2^n) curve and at the some point have a sharply increment(Because the data in time complexity increase too fast, the other two curve cannot represent their trend in the same graph). The time spend in running the BF method is O(n!) due to premutation idea. Because this method requires to calculate all possible premutation of given nodes(n) which is (n-1)! in total paths, thus, the time complexity is O(n!). For DP method is O(n^2\*2^n) is because DP is a bottom up method which divide a big problem into some sub-problems and solve the sub-problems one by one. The time complexity for solving each sub-problem(n-1) will be O(n) which is using to find conbinations/subset. Also, the O(2^n) coming for the following: Suppose the starting node is “1”, and the ending node is also “1”, how many sub-problems(subset) will need to consider? Will have:



In this case, when the nodes n = 4, the DP method need to consider 2^(n-2) subsets/combinations, Therefore, for n nodes, the DP method need to consider 2^(n-2) subsets/combinations which time complexity is O(2^n). and total sub-problem finding is O(n\*2^n). Because DP will use the recursive function and running for time of the size of node times, the whole time using is another n. Therefore, the full-time complexity for DP method is O(n^2\*2^n).

1. DP method I use:

On my own DP method, I write up two functions to complete the DP’s functionality and I also use the Adapter pattern fulfill my design. In the function called “DP”, I make a graph with size n by n and calculate all distance by inputting the positions of nodes. And then I create a double int vector “dp”, initializing all values into -1 and the size is n by 2^n which use to store all possible path’s cost. After that I call my next function “findCost” in a recursive way to find the minimum cost. The “findCost” function will receive four parameters including the graph I make, the start position, a mask which will represent the path state and the dp. Once the mask size equal to the left shift graph size which mean that I already looking up all the combinations and need to return. And then, I have if condition to check whether the cost in dp change or not and need to be return to keep track. I initialize the ans into a really big number because this variable will be changed and finally will store a minimum cost. Moreover, I create a for loop to look up all nodes to find a possible path. In the for loop, there is a if loop to check that current node was visited or not. Because mask is an way to keep track the cost, therefore, when I need to compare the mask and the node, I need to make a 1 left shift city and do the “AND” with mask. For example, if mask equal “1” in binary will be 0001, and if city is “0”, in binary will be 0000 and they will AND up into 0000 which is “0”, it means the city haven’t visited and need to do the recursive to calculate the new cost. Comparing the new cost and the cost I got before and choose a smaller one. Finally, when I done all the recursive I will return the minimum to my DP function. (Reference link: https://codingblocks.com/resources/travelling-salesman/)