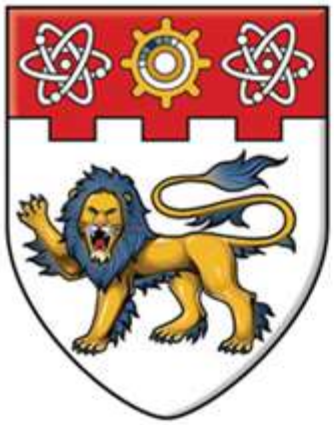


NANYANG TECHNOLOGICAL UNIVERSITY

CZ2001 Algorithm Individual Assignment

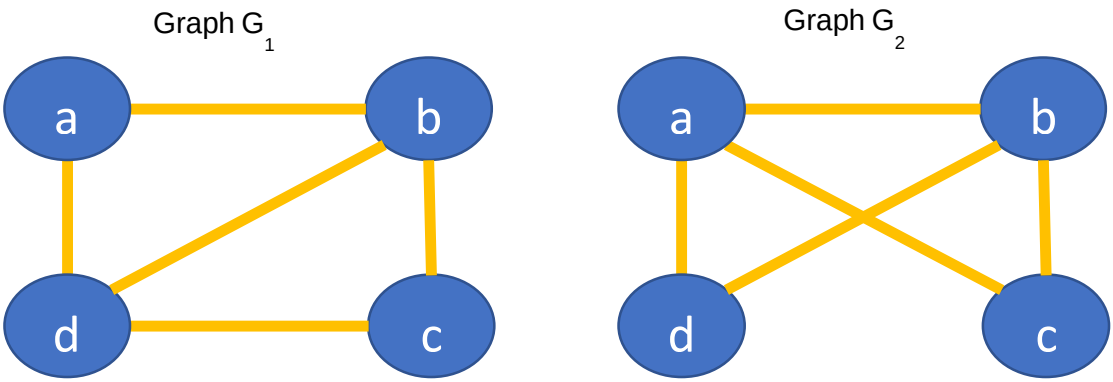
Name: Lee Cheng Han

Matriculation Number: U1920206L

Tutorial Group: SE2

Email: CLEE069@e.ntu.edu.sg

Question 1



The above problem is an NP class problem as it cannot be solved in polynomial time.

This problem is a decision problem as there are only two possible answers, one being the graph are the same while the other being that the graphs are different. To solve this decision problem, we need to consider all possible result and terminate after we found that both Graphs are different.

To find out whether both Graph G_1 and Graph G_2 are the same, we must use the concept of Graph Isomorphism (Researched by L´aszl´o Babai). Graph Isomorphism states that a graph can exists in different forms having the same edge connectivity. It is the problem of deciding whether there is a bijection between theirs sets of vertices $V(G_1)$ and $V(G_2)$ that takes edges of Graph G_1 to edges of Graph G_2 and non-edges of Graph G_1 to non-edges in Graph G_2.

Another way to put it is that there exists a function 'f' from vertices of G_1 to vertices of G_2 [f: $V(G_1) \Rightarrow V(G_2)$], such that there are two criteria must be satisfied. The first criteria being that f is a bijection and the second is that f preserves adjacency of vertices. For instance, if the edge is $\{a, b\} \in G_1$, then the edge $\{f(a), f(b)\} \in G_2$ and $G_1 \equiv G_2$.

For the first criteria, a bijection between theirs sets of vertices $V(G_1)$ and $V(G_2)$ simply means that there must be a same number of vertices and edges in Graph G_1 and Graph G_2. $|V(G_1)| = 4$ and $|E(G_1)| = 5$ for Graph G_1 while $|V(G_2)| = 4$ and $|E(G_2)| = 5$ for Graph G_2. The first criteria is met as number of components are same.

For the second criteria, edge connectivity is not retained in both Graphs. Despite both graphs having 2 cycles of length 3 ({a, b, d, a} and {c, d, b, c} for Graph G_1 and {b, c, a, b} for Graph G_2), it does not mean that edge connectivity is retained. {a, b, c, d} in Graph G_1 suggests that it has a cycle of length 4 (a-b-c-d-a) which is not present in the second graph. The second criteria is not met. Hence, we conclude that both Graphs are not the same.

To prove that this problem cannot be solved in polynomial time, it is important to find the time complexity. It has a time complexity of $O(n!*n^2)$ by trying all n! mappings of the vertices of Graph G_1. Since it is not solvable in polynomial time, this is an NP class problem.

Site: https://www.tutorialspoint.com/graph_theory/graph_theory_isomorphism.htm

Question 2a

A suitable approximate algorithm for the travelling salesman problem is the Christofides algorithm. It is one of the more famous algorithms that have multiple steps. It is significantly better than any of the exact solution approaches with a time complexity of $O(V^4)$, V being the number of cities(Vertices) the salesman must travel.

Pseudocode for Christofides Algorithm
Christofides (G(V,E))
MST(G(V,E))
Matching(MST())
MST MultiGraph(Matching,MST)

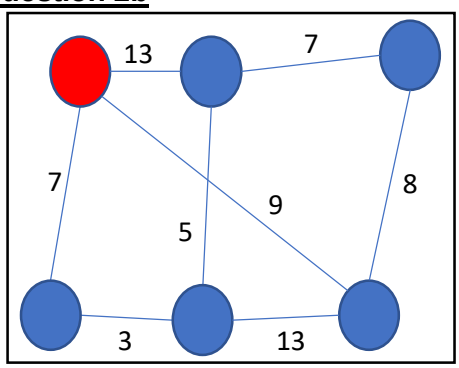
The first step is to implement a minimal spanning tree. Based on what we learnt in lecture, Prim's Algorithm can be used to find minimal spanning tree. This is done by selecting vertex from the fringe vertex. The next step is to obtain odd degree vertices. This is done by counting the number of edges linked to a vertex and if the number of edges connected to the vertex is odd-numbered, that vertex will be known as an odd-degree vertex. The following step is to create a minimum perfect matching of the odd-degree vertex found in the previous step. At the end of this step, all the odd-degree vertex will be connected via an edge. The second last step consists of forming a Euler tour which combines the multigraph created from the previous step and minimal spanning tree. The purpose of this is to change all odd degree nodes to even degree for the next step. Finally, a Hamiltonian path is created to remove additional cities visited by the salesman. Cities visited will not be visited again by the traveller as it walks on the Eulerian circuit.

The Christofides algorithm time complexity has a time complexity of $O(V^4)$. The running time of the Edmonds' algorithm which is used to create minimum perfect matching of the odd degrees, the costliest step, is bounded by $O(V^4)$ times a polylogarithmic factor. Prim's algorithm which is used to determine the minimal spanning tree has a time complexity of O(E log V), E being the number of edges which is more efficient than Edmond's algorithm, hence the time complexity of the Christofides algorithm is $O(V^4)$.

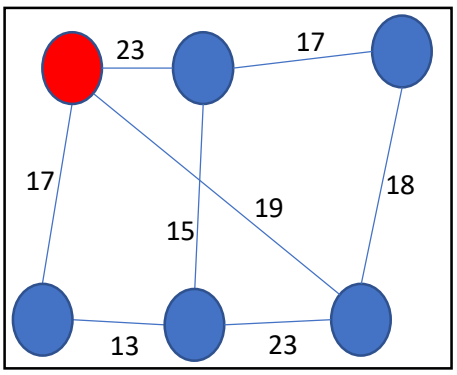
Despite being one of the best algorithms in solving the travelling salesman problem, it does not come without its limitation. The main drawback to Christofides' algorithm is the fact that as the algorithm would take longer with increasing vertices. The number of comparisons will increase exponentially when there are more cities.

Site: https://cse442-17f.github.io/Traveling-Salesman-Algorithms/

Question 2b



Input that give best solution



Input that give not-the-best solution

Question 3a

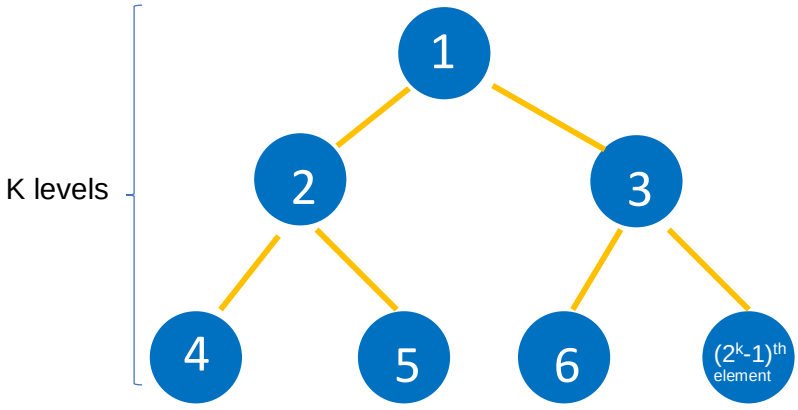
When running on an he best case for mergesort is $\frac{n}{2}log_2 n$ while the best case for insertion sort is $n-1$.

If n > 8, mergesort is used. When n = 8, $\frac{8}{2}\log_2 8 = 4 * 3 = 12$. Mergesort does not happen when n = 8 so $\frac{12n}{8}$ must not be factor into the key comparison calculation.

Since insertion sort only occurs when n is less than or equals to 8, the insertion sort key comparison can be calculated by $n - \frac{n}{8} = \frac{7}{8}n$.

Total final number of key comparisons is $\frac{n}{2}\log_2 n + \frac{7}{8}n - \frac{12}{8}n = \frac{n}{2}\log_2 n - \frac{5n}{8}$

Question 3b



Since array is already in ascending order, the above diagram will be an example of array B in the form of a tree. It is also stated in the question that there are 2^k-1 distinct elements. This means that there are k levels in the tree. ConstructHeap algorithm is applied to B till it is a maximizing heap.

A maximizing heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node. This means that the $(2^k-1)^{th}$ element will be at the first level after the tree has been heapsorted since the array is in ascending order and it will be the node with the biggest value. This is only achieved after k-1 comparisons as it moves from k^{th} level to 1^{st} level. By observation and backward substitution, we have

$$
\begin{aligned}
w(p) &= 2w(p-1) + 2(p-1) \\
&= 2[(2W(p-2) + 2(p-2)] + 2(p-1) \\
&= 2^2 w(p-2) + 2^2(p-2) + 2(p-1) \\
&= 2^2[(2w(p-3) + 2(p-3)] + 2^2(p-2) + 2(p-1) \\
&= 2^3 w(p-3) + 2^3 w(p-3) + 2^2(p-2) + 2(p-1) \\
&= 2^{p-2} w(2) + 2^{p-2}(2) + 2^{p-3}(3) + \cdots + 2(p-1) \\
&= 2^{p-2}(2) + 2^{p-2}(2) + 2^{p-3}(3) + \cdots + 2(p-1) \\
&= 2^p + \sum_{i=0}^{p-3} 2^p(p-i) \\
&= 2^p + 2(p-1) + 4(p-2) + 8(p-3) + \cdots + 2^{p-3}(3)
\end{aligned}
$$

$$= 2^p + 2((p-1) + 2(p-2) + 4(p-3) + \cdots + 2^{p-2}(3))$$
$$= 2^p - 2p + 2 + 4(2^{p-4} - 1) + 2^{p-2}(3)$$
$$= 2^p - 2p - 2 + 2^p$$
$$= 2^{p+1} - 2p - 2$$
$$= 2^{k+1} - 2k - 2 \; where \; k = p$$

Site: https://socratic.org/questions/59d6254b7c01492d0823900d

Question 4

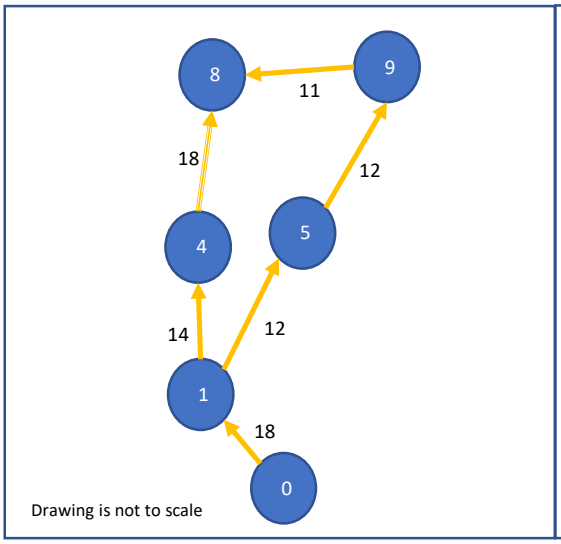


Figure 1 – Graph G

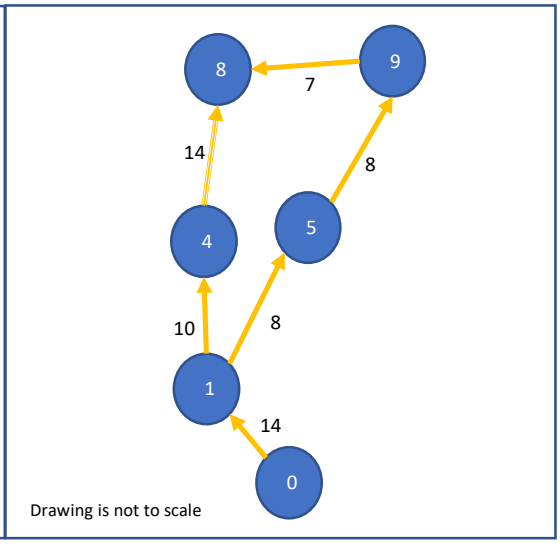


Figure 2 – Graph G'

Figure 1 shows Graph G, which is a weighted and directed graph with all edge weights greater than 10. Figure 2 shows Graph G' for which weight has been reduced by 4.

By applying Dijsktra's algorithm on G' and computing the shorted path, the shortest path computed on G' will not be the shortest path in G. The shortest path may change as there may be a different number of edges in different paths.

For example, the shortest path from node 0 to node 8 in Graph G is Vertex 0 -> Vertex 1 -> Vertex 4 -> Vertex 8 and the total weight is 50. Another path can be seen where Vertex 0 -> Vertex 1 -> Vertex 5 -> Vertex 9 -> Vertex 8 and the total weight is 53. For Graph G', the weight of the new path is 38 and 37 respectively.

Hence, it is not always that the shorted path in Graph G' is also a shortest path in Graph G.