Jacob Alspaw **jaa134** **HW4**

1

a. The smallest possible number of edges in a connected graph with N nodes is N-1 edges. In this graph, there will be one connection resulting from every node, subtracting one. This graph must be acyclic, because, otherwise, there would be an extra edge to produce the cycle. The minimum connections in a cyclic graph are N-1+1=N. N is greater than N-1, therefore this graph will be acyclic.

b. This proof can be solved by inductive assumption. If we assume the largest number of connections comes from a graph that is fully connected, then each node can touch every other node. This is represented by the equation P(N) = N(N - 1)/2, where N is the number of nodes. Through the inductive step, we can see that adding another node to the set still holds true to our equation. This is true for all N greater than or equal to zero. The graph with the most edges must be complete if we look at set theory. If we split a grouping of nodes into two subsets, whose nodes are not connected between each subset, but whose individual graphs are complete, then we can prove the graph with the most edges must be perfectly complete. Assume the cardinality of the first set is "j", and the cardinality of the second set is "k". The maximum number of edges in the first set is j(j – 1)/2 and the maximum number of edges in the second set is k(k-1)/2. The sum of edges in the first set and the second set is less than the edges of a perfectly complete graph. Additionally, a complete graph denotes every node is connected by a unique node; the behavior is every possible connection that can be made, will be made. If this is not the case, then our graph will not be complete or reach the maximum amount of edges.

2

This question all depends on the pivot selection. By selecting the middle input of each array, we assure quicksort runs quickest in both case.

a. Quick Sort: Pre-Sorted: O(nlogn) because $\log_2 n$ levels and then n comparisons each level.

b. Quick Sort: Reverse: O(nlogn + n/2) = O(nlogn) because $\log_2 n$ levels and then n comparisons each level with the addition of swapping for the first run through of the array.

c. Even though both situations are represented by O(nlogn), because there is swapping the first run through of the array for reverse sorted, whereas pre-sorted there is no need, then theoretically pre-sorted arrays will be sorted faster.

3

The solution for this question is as simple as partitioning an array. First we should start by labeling one side of the array blue, and one side of the array red. There will be no boundary defining the middle section between these two colors, yet. If we have indices starting outside of our array, form opposite sides, then we have a means to sort the set in linear time. By moving these indices inward until they cross each other, the indices will need to stop at an out of place object. Once both indices stop, they will swap they're objects and continue moving inward. One the two indices cross positions in the array, the sorting algorithm has been completed in linear time and the boundary is the location in between the two indices.

O(n/2) to move the left index inward.

O(n/2) to move the right index inward.

O(x/2) to swap out of place blocks, where x is the number of out of place blocks

Total comparisons: O(n/2 + n/2 + x/2) = O(n)

4

Initially load in blocks 1, 4, and 7 and merge the numbers contained by the blocks. Once a block has been fully used, the next block in its array will be loaded into the main memory, taking the previous blocks spot. This process continues until all numbers have been merged and all blocks are empty.

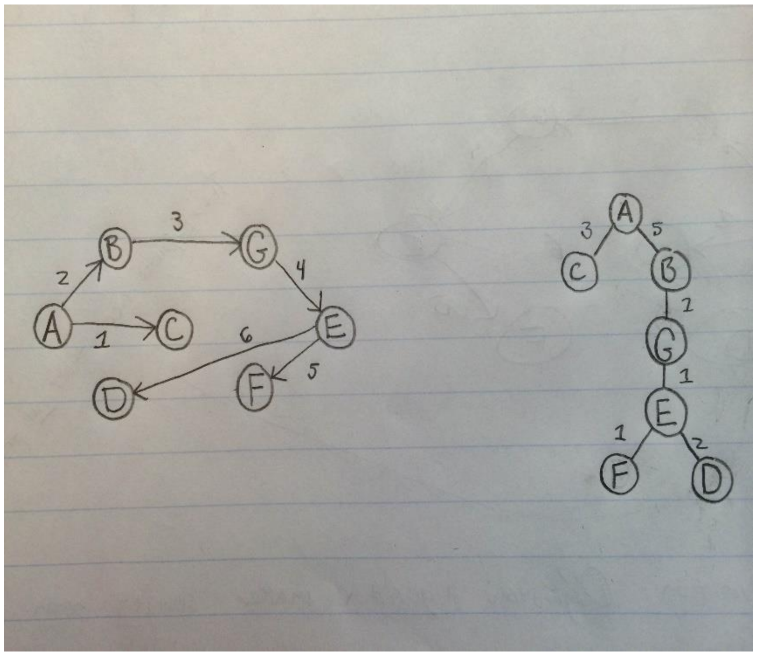
Block 1 will have its numbers merged first. Then block 2 will be loaded in its place. Then block 2 will run out and block 3 will take its place. Block 4 will run out and 5 will be loaded into its place. Block 7 will run out and block 8 will be loaded into its place. Block 3 will run out, but no other block will take its place. Block 8 will run out and block 9 will replace it. Block 5 will run out and block 6 will replace it. Finally, Block 6 will run out, and then block 9 will run out.

Loading order: 1 -> 4 -> 7 -> 2 -> 3 -> 5 -> 8 -> 9 -> 6

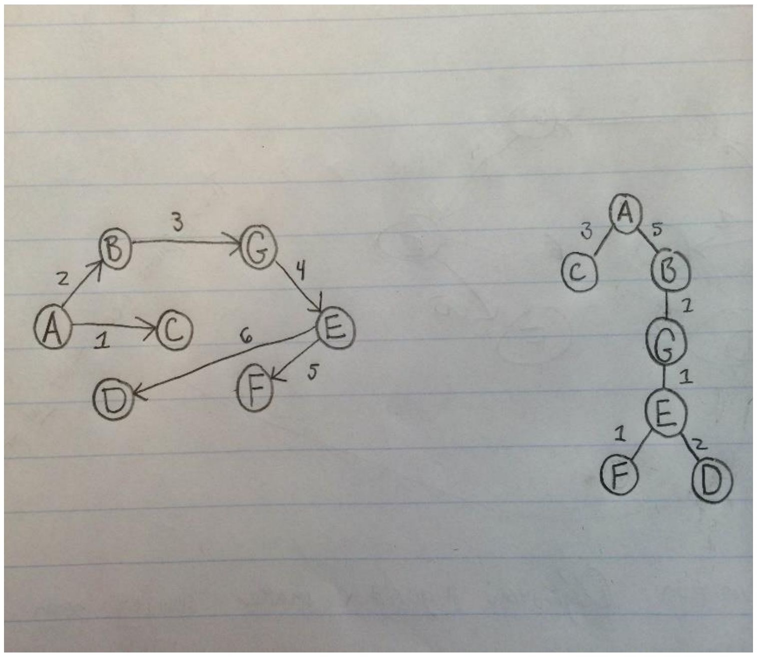
5

The path order is labeled next to each transversal of the graph.

a. Dijkstra's Algorithm:



b. Prim’s Algorithm:



c. -By definition, Dijkstra’s algorithm will produce the shortest spanning tree. From a designated node, “A,” the shortest path from said node to any other node can be found using Dijkstra’s algorithm.

-If we divide the graph into two disjoint subsets {A,B,G,E,D,F} and {C}, then the disjointed edge connecting node A to node C should be part of our minimum spanning tree. However, there exists a connection with less spending. By connecting nodes B and C, we can reduce overall spending by one. Therefore, because there is a more cost effective solution, Prim’s algorithm did not produce the minimum spending tree.