1. For a question to be polynomial-time, it has to have a polynomial time complexity. For instance, finding the nth Fibonacci term recursively is not polynomial because the time complexity is factorial. Meanwhile, finding the nth Fibonacci term iteratively is polynomial because the time complexity is O(n).
2. A polynomial-time algorithm is an algorithm that can find an answer to a problem with a polynomial amount of basic operations. The set of decision problems called P are problems for which a polynomial solution has been found, hence they are in P.
3. Theta: Theta will define the behavior of an algorithm time-complexity. The algorithms graph will basically match Theta’s.

Omega: This will define an algorithm lower bound. The lower bound is like the upper bound, there will exist an N where the algorithm will not drop below the Omega curve.

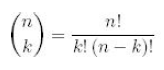
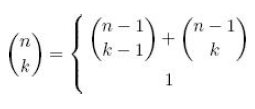
Big O: This defines the upper limit of an algorithm. The algorithm will have an input size N that will always be below the curve of its Big O order.

1. I would prefer an algorithm of order n2. N2 is of polynomial time, this is much better than 2n which is exponential. The exponential algorithm would be way to resource intensive.
2. Yes, you can store the previously visited node in a balance tree to make the algorithm access them faster on sequential calls.
3. Binary search has a O(lg n) and sequential search has a O(n). I would use binary search because it is more time efficient. It is more efficient because it will scale better with a larger input. Sequential may be better if the key is at the beginning of an array but the worst case time complexity is much higher than O(lg n).
4. Merge sort will divide and conquer the array then merge each element and sub array back into one large sorted array. Quick sort will also divide and conquer around a pivot and sort the array as it divides.
5. Establish a recursive property that gives the solution to an instance of the problem.

Solve an instance of the problem in a bottom-up fashion by solving smaller instances first.

1. - The recursive property would be the N-1 fashion of the factorial portions of the binomial coefficient algorithm

* This algorithm will make pascal’s triangle; therefore, we can use preceding values to solve future instances of the problem

1. Step 1: D(0) will be used to calculate D(1) and so on. This shows the recursion.

Step2: the algorithm will complete from bottom up: D(0) – D. Each pass will build from the previous solution until the answer is reached.

EX: D(1)[2][4] = min(D(0)[2][4], D(0)[2][1] + D(0)[1][4])

1. A greedy algorithm is one that will take the best possible solution current solution with out the consideration of past moves and it cannot reconsider its choices. For example, Prims algorithm is greedy because it will not backtrack and it always chooses the current best edge.
2. A greedy algorithm for providing change would be to provide the largest valued coins first, then subtract what could be given from the largest coin. Then you would consider the next tier of coin, for instance the dime. Give dimes if possible and subtract 10 cents for every dime. You can repeat this for every coin denomination.
3. Prims algorithm is clearly polynomial because the time complexity is O(n2). The repeating loop inside of the algorithm only must complete n-1 times.
4. No, if there is not a unique MST prims algorithm may produce a different MST.
5. Backtracking is helpful for problems which require a sequence of elements to be chosen for a solution, for example, the n-Queens problem.
6. The third node to be pruned for the N-Queens problem would be node (3,1) this would prevent 4 unnecessary nodes from being visited.
7. Computational complexity is the hierarchy of an algorithm’s overall complexity, including other complexities such as space, and time complexity. Time complexity is just a part of the computational complexity.
8. It would make sense that you could find an algorithm with a time complexity that is the lower bound of the computational complexity but because the computational complexity contains more variables, I don’t think it is possible.
9. Yes, you can solve a problem for searching an array for a key in polynomial time. For example, the book provides algorithms like binary search and sequential search which are tractable.
10. Finding a polynomial time algorithm for and NP problem would provide answers to all current NP problems. The answer can be modified to solve all problems. Its currently the Computer Science “Million Dollar question” as we discussed in class.
11. SEE PICTURE
12. SEE PICTURE
13. SEE PICTURE
15. A problem may be NP and NP complete if the reduction can be completed in polynomial time and there is not a proof that a polynomial time does not exist.