* Revisit questions from Midterm and Quizzes and exploration
  + Question 4:
    - Big-Ω: For large values of n the running time of f(n) is at least b⋅g(n)
      * Ω(f(n)) = if the lim is infinity (Omega)
    - Big-Θ: For large values n the running time of f(n) is at least a⋅g(n)
      * Θ(f(n)) = if the lim is const(C) (Theta)
    - Big-O: For large values of n the running time of f(n) is at most b⋅g(n)
      * O(f(n)) = if the lim is trends to 0 (Big-O)
  + Question 5-10:
    - The correct inequality for the asymptotic order of growth:
      * n! > 2n
      * (n log n) < 2n
      * n2 > (n log n)
      * 5n > n5
      * (log n) < √n
      * This means summation on i where i ranges from 1 to n = nk, where k=2
      * Diagram

        Description automatically generated
  + Question 11:
    - True/False:
      * ∈ O(n4) : True
      * ∈ O(n3) : True
      * ∈ Θ(n4) : False
      * ∈ Ω(n2) : True
      * A picture containing application

        Description automatically generated
  + Question 12:
    - Given an array A of size n, we want to access the ith element in the array, 0<i<n. What will be the time complexity of this operation?
    - Ans: O(1) because you iterate through one by one until you reach i.
  + Question 13:
    - Given an array A of size n, we want to find if an element k belongs to this array. What will be time complexity of this search operation?
    - Assume that we don't know anything about the order of elements in the array.
    - Ans: O(n), since we do not know the order we operate with however big the array is
  + Question 14:
    - Given a sorted array A of size n, we want to find if an element k belongs to this array. What will be the **best time complexity**to perform this search operation?
    - Note: best **time complexity**and not the best time
    - Ans: O(log n), since it is sorted we don’t work with complexity of O(n), O(1) is too slow….
  + Question 15:
    - Graphical user interface, application, email

      Description automatically generated
    - Need to read algorithm more carefully‼
  + Question 16:
    - Write the loop invariant for the following code:
      * item = -INF (minus infinite)
      * for (i = 0 to n-1)
      * if (A[i] > item)
      * item = A[i]
    - Ans: the loop invariant condition is that 'item' is always maximum among the first i elements of array A.
    - Loop invariants: whatever the holder in the loop is holding throughout the entire loop.
  + Question 17:
    - 1. def mystery(A[0..n-1, 0..n-1]):  
      2. for(i = 0 to n-2):  
      3. for(j = i+1 to n-1):  
      4. if A[ i,j ] != A[ j,i ]  
      5. return False  
      6. return True
    - What does the algorithm compute? The algorithm returns “true” if its input matrix is symmetric and “false” if it is not.
    - What is its basic operation, write the **line number**of code (1, 2, 3, 4, 5, or 6) that is executed maximum number of times? Line 4
    - What is the time complexity of this code? O(n2)
  + Question 18:
    - Consider the subset sum problem.
    - Problem: Given an array of numbers find if there is a subset that adds to a given number. Return True if there exists such subset, else return False. The subset of numbers need not be continuous in the array. We don't know anything about the order of the elements in the array.
    - Identify which of the following strategies can be used to solve this problem.
      * Dynamic Programming: Can be used
        + Dynamic programming – the most optimal solution always/substructure. Overlapping subproblems.
      * Backtracking:    ["", ""] can
        + Back tracking – gives all possible solutions.
      * Brute force Approach:    ["", ""] can
      * Divide and Conquer:    ["", ""] cannot
        + Divide and conquer: quick sort, merge sort, binary search (n log n). Break a problem into not overlapping subproblems. Reduces time complexity usually due to recursion.
  + Question 19:
    - Which of the following is/are property/properties of a dynamic programming problem?
    - Ans: Both optimal substructure and overlapping subproblems.
  + Question 20:
    - In dynamic programming, the technique of storing the previously calculated values is called \_\_\_\_\_\_\_\_\_\_
    - Ans: Memoization
  + Question 21:
    - Which of the following techniques can be called as intelligent exhaustive search?
    - Ans: Backtracking
* Identify time complexity of pseudocode
* Can you identify what does a pseudocode do / write the pseudocode for the following:
  + topological sort
  + Prim’s algorithm
  + Kruskal’s algorithm
  + Dijkstra algorithm
  + Greedy algorithms.. Etc.
* Can you tell the steps to prove a problem to be NP-complete?
* Space for “Additional Notes”: Make notes of any assumptions that you had to make for any question.