**#1 (4 points):** Rank the following functions by order of growth into groups. Place functions f(n) and g(n) into the same group if and only if f(n) ∈ Θ(g(n)). List the group in ascending order of growth.

a) n^2

b) sqrt(n)

c) log(n)

d) 1 + 2 + 3 + … + (n-1) + n

e) n!

f) 1000n

g) n log(n)

h) 1

i) 2^n

**#1** **Answer** (*smallest growth on top*):

h) 1

c) log(n)

b) sqrt(n)

f) 1000n

d) 1 + 2 + 3 + … + (n-1) + n a) n^2 ----> same group

e) n!

i) 2^n

**#2 (3 points):** Briefly explain why Python lists should not be used as keys for a hash table.

**#2 Answer:**

Lists are mutable and thus you can change the elements, add new elements, or delete elements . So, if you change the items of the list after the hash key was generated, then the hash key will be changed, and you will loose track of items that you had stored in the hash table.

**#3 (4 points)** We covered several data structures: list, stacks, queues, heaps, and hash tables. For each problem below choose the best data structure that can be used to solve the problem efficiently.

1. Given a list A of n integers, reverse the order of the elements in A.

**Answer:** I think a stack would be a good structure for this. The way a stack works (LIFO) should work well for reversing the order of something. You could also just simply use another list. I think whatever data structure you use it will take linear time.

1. Given a list A of n integers, report the ten largest integers in A.

**Answer:** I think a maximum heap would work best for this. To get each largest element in order would take log(n) time so for the top 10 elements it would take 10\*log(n) time. A list would not be too bad but it would probably need to be sorted first which would take n\*log(n) time with merge sort. If the list is already sorted then list would most likely be the best data type since it would take constant time to look up the top 10 elements.

1. Given an undirected graph G determine if G is connected.

**Answer:** I think this depends on what format the data is in. I will assume that we have an adjacency list. For an adjacency list you could either use a queue or a stack data structure. Both have the same worst case run time however I believe a stack (depth first search) is generally more memory efficient.

**#4 (5 points):** We covered several algorithm design techniques: incremental, divide-and-conquer, dynamic programming, greedy, and backtracking. For each problem below choose the most efficient technique that can be used to solve the problem efficiently. If more than one choice is available select the one that is simplest to implement. Justify your answer.

1. Given a list of n triangles, find the range (i.e., min and max) of the perimeters of the input triangles.

**Answer:** Incremental design I think. Just loop through all the triangles and keep track of the min and max perimeter. This will run in linear time.

1. Sort a list of n triangles in decreasing order of perimeter.

**Answer:** I think the most efficient sorting algorithm we covered was merge sort which is a divide-and-conquer solution. This will always run in n\*log(n) time no matter the input. Other algorithms could be faster but often have a slower worst case.

1. Given a list of n items and their corresponding integer weights, maximize the total weight of items that fit in a backpack of capacity W by selecting items from the given list.

**Answer:** I am assuming that W is the maximum *weight* the backpack can hold. I believe the most efficient approach to this would be dynamic programming. There is definitely no greedy property for this problem so that will not work. I think you could solve this using divide-and-conquer but I think there would be repeat work involved. Memoization may also be a valid implementation but it is not one of the listed choices for the question.

1. Given a list of n items and their corresponding integer weights, maximize the total number of items, selected from the given list, that fit in a backpack of maximum capacity W.

**Answer:** I am assuming that W is the maximum *weight* the backpack can hold. I think a greedy algorithm will work for this. If you sort the items in descending weight and incrementally take items off the end of the list until you reach the maximum capacity that should work.

**#5 (4 points):** Explain the similarities and differences between Divide-and-Conquer and Dynamic Programming. When would you use Dynamic Programming instead of Divide-and-Conquer?

**Answer:**

Divide and conquer involves splitting a problem into sub-problems, solving each piece separately, them merging the solutions back into one main solution. This almost always involves recursion. Dynamic programming involves building up solutions until you reach the goal solution.

Any time you are repeating work or calculations in an implementation of a divide-and-conquer is a good time to use dynamic programming. This happens somewhat frequently with divide and conquer solutions. In addition if you have an exponential run time for an algorithm this is generally a good hint to use dynamic programming.