CS2302 - Data Structures

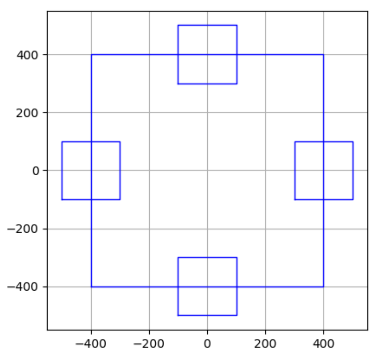
Spring 2020

Final Exam

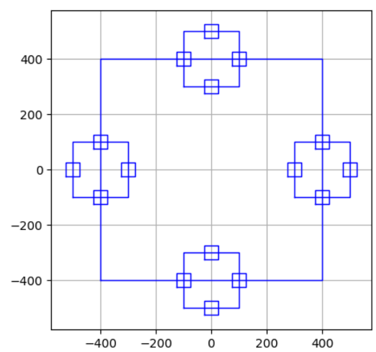
'In my life, I have met many good people who fail a Data Structures exam, however, I have never met a good person who cheats in a Data Structures exam' - Mahatma Gandhi

General note: For all your functions, try to write the most efficient solution possible. Even if your function works, points will be taken off if its running time is suboptimal.

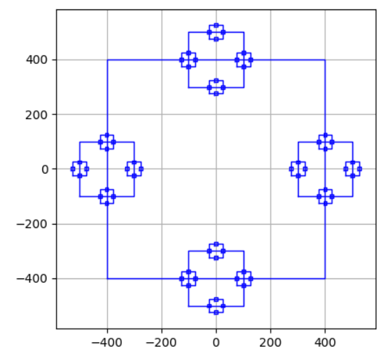
1. (12 points) Write the function *draw_squares(ax, n, x0, y0, s)* that draws figure like the ones below, where n is the depth of recursive calls, $x0, y0$ is the center of the figure, and s is the length of a side of the square (see starter code for sample runs).



draw_squares(ax,2,0,0,800)



draw_squares(ax,3,0,0,800)



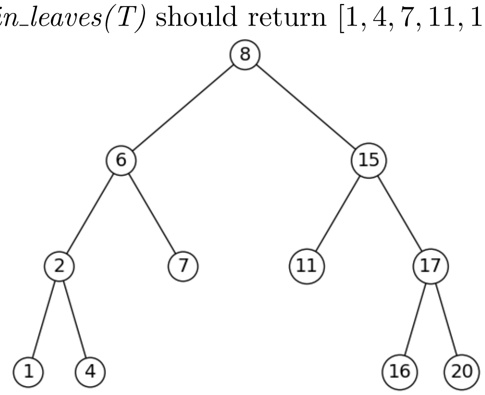
draw_squares(ax,4,0,0,800)

2. (12 points) Write the **recursive** function *smaller(L,i)* that receives a (native) Python list L and an integer i and returns a Python list containing all items in L are smaller than i, in the reverse order than they appear in L and without modifying L.

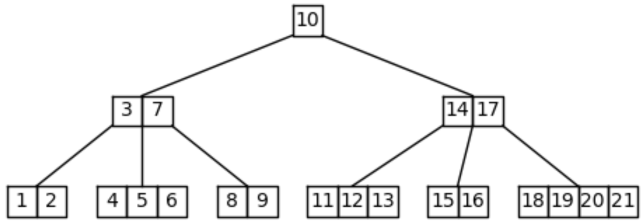
3. (12 points) The cumulative sum of a list A is a list C of the same length as L where $C[i]$ contains $A[0] + A[1] + ... + A[i]$. Thus $C[0] = A[0]$, $C[1] = A[0] + A[1] = C[0] + A[1]$, and, in general $C[i] = C[i-1] + A[i]$. For example, if $A = [2, 3, 1, 4]$, then $C = [2, 5, 6, 10]$. Write the function *cumulative_sum(L)* that receives a reference to a List object L (as defined in singly_linked_list.py) and builds and returns a List object containing the cumulative sum of L.

4. (12 points) Write the function *sorted_row(A)* that receives a 2D numpy array A and returns a list containing the indices of the rows in A that are sorted in ascending order.

5. (12 points) Write the function *in_leaves(T)* that receives a reference to the root of a binary search tree T and returns a list containing the items that are stored in leaf nodes in the tree. For example, if T is the root of the tree in the figure, *in_leaves(T)* should return $[1, 4, 7, 11, 16, 20]$.

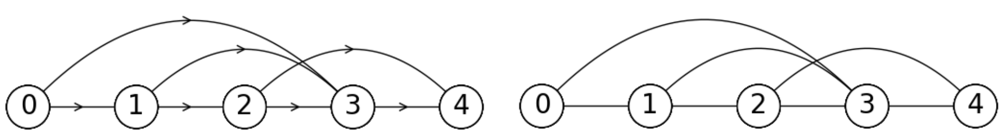


6. (12 points) Write the function *internal(T)* that receives a reference to the root of a B-tree T and returns a list containing the items that are stored in internal (non-leaf) nodes in the tree. For example, if T is the root of the tree in the figure, *internal(T)* should return $[3, 7, 10, 14, 17]$.

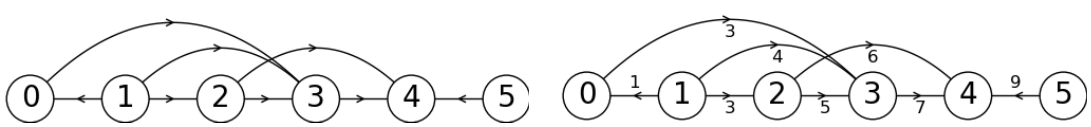


7. (12 points) A (much) simpler version of *subsetsum* consists of, given a list of integers S and a goal k, determining if there are **two** elements of S that add up to k. This problem can be solved in $O(n)$ using a hash table, as done by the function *find_sum_pair(S,k)*. The function provided returns *True* if the pair of numbers exists and *False* otherwise. Modify it to return a list containing the two numbers, if they exist, and *None* otherwise. For example, if $S = [1, 3, 6]$, *find_sum_pair(S,7)* should return $[1, 6]$ and *find_sum_pair(S,10)* should return *None*.

8. (12 points) Write the function *make_undirected(G)* that receives a directed graph G represented as an adjacency matrix and converts G to an undirected graph. For example, if G is the graph on the left, after executing *make_undirected(G)*, G should be the graph on the right.



9. (12 points) Write the function *make_weighted(G)* that receives an unweighted graph G represented as an adjacency list and converts G to a weighted graph, where the weight of an edge is the sum of the indices of the vertices it connects. For example, if G is the graph on the left, after executing *make_weighted(G)*, G should be the graph on the right.



10. (12 points) The function *subsetsum_v2(S,g,rem)* is an attempt to optimize subsetsum by stopping backtracking when the goal is greater or equal to *rem*, the sum of elements remaining in S (if the goal is equal to the sum, return *True*, since the solution is to take all elements, if it is greater there is no solution). If S is a long list, *subsetsum(S,sum(S)+1)* will never finish, while *subsetsum_v2(S,sum(S)+1),sum(S)* will return the right answer without even making a recursive call. Unfortunately, there is a bug in the function. Fix it so it works properly.