Name & UNI: Adrian Cardenas ac4982 Date: 12/07/2022

Point values are assigned for each question. Points earned: \_\_\_\_ / 100, = \_\_\_\_ %

1. Joe Pancake has just invented what he thinks is a great sorting algorithm. Consider the following code:

/\*\*

\* Returns index of the maximum element in array[0..n-1].

\*/

**int** **find\_max**(**int** array[], **const** **int** length) {

**int** max\_index, i;

**for** (max\_index = 0, i = 0; i < length; ++i) {

**if** (array[i] > array[max\_index]) {

max\_index = i;

}

}

**return** max\_index;

}

/\*\*

\* Reverses array[0..limit].

\*/

**void** **flip**(**int** array[], **int** limit) {

**int** temp, start = 0;

**while** (start < limit) {

temp = array[start];

array[start] = array[limit];

array[limit] = temp;

start++;

limit--;

}

}

**void** **pancake\_sort**(**int** array[], **const** **int** length) {

**for** (**int** curr\_size = length; curr\_size > 1; --curr\_size) {

**int** max\_index = find\_max(array, curr\_size);

**if** (max\_index != curr\_size-1) {

flip(array, max\_index);

flip(array, curr\_size-1);

}

}

}

Show the array [2, 4, 1, 5, 3] after the for loop in pancake\_sort executes once.

(5 points, one for each element in the correct position)

For the execution of the for loop: [1, 2, 3, 4, 5]

What is the best-case complexity of the pancake\_sort algorithm above including helper functions? (1 point for symbol, 1 for function)

The best case time complexity is Theta(n)

What is the worst-case complexity of the pancake\_sort algorithm above including helper functions? (1 point for symbol, 1 for function)

The worst case time complexity is Theta(n^2)

Which of the elementary sorting algorithms discussed in class is *closest to (but not exactly the same as)* what Joe has written? Use the complexities to guide your response. Circle your answer. (1 point)

bubble / selection / insertion

1. You are given a collection of bolts of different widths and corresponding nuts. You are allowed to try a nut and bolt together, from which you can determine whether the nut is larger than the bolt, smaller than the bolt, or matches the bolt exactly. However, there is no way to compare two nuts together or two bolts together. The problem is to match each bolt to its nut.

**Note:** You *cannot* get credit for parts b and c if your answer to part a is incorrect.

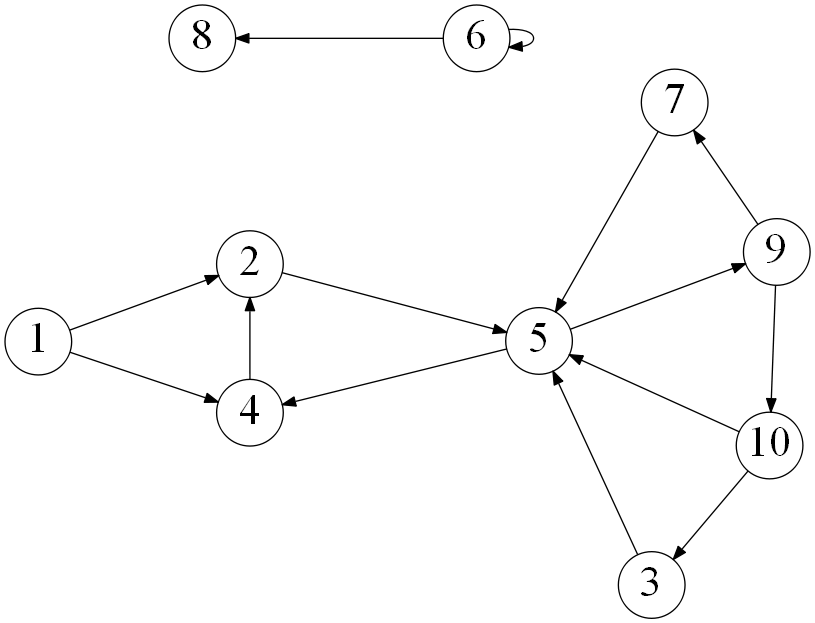
1. Design an algorithm for this problem with *efficient* average-case complexity faster than quadratic run time. (6 points)
2. Choose any nut and check each bolt to find the one that fits (bolt n).
3. Split the pool of bolts into those larger than that nut and those smaller than that nut.
4. Use the bolt n to check each nut to find the one that fits (nut n)
5. Split the pool of nuts into those larger than that bolt and those smaller than that bolt
6. Now that nuts and bolts have been indirectly compared, you can continue to use the method above recursively on the pool of nuts and bolts that are both larger and smaller than the original nut-bolt pair
7. Continue until all nuts have their corresponding bolts.
8. What is the average-case complexity of your algorithm? (1 point for symbol, 1 for function)

Theta(nlogn)

1. What algorithm did we study this semester that is most similar to the algorithm you’ve designed? (2 points)

Quicksort

1. List the order in which the vertices are visited with a breadth-first search starting with vertex 1. If there are multiple vertices adjacent to a given vertex, visit the adjacent vertex with the lowest value first. (10 points)



1 -> 2 -> 4 -> 5 -> 9 -> 7 -> 10 -> 3 -> 6 -> 8

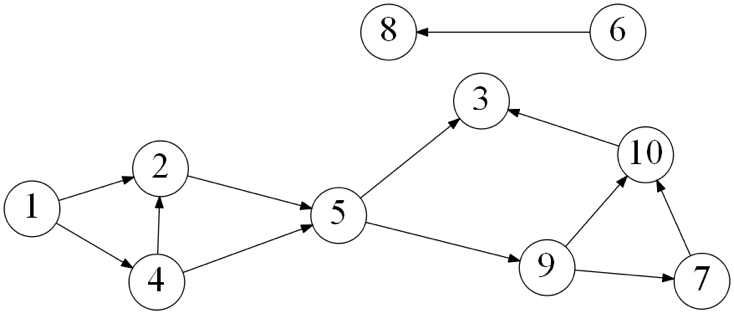
1. Using the same graph as in the previous question, list the order in which the vertices are visited with a depth-first search starting with vertex 1. If there are multiple vertices adjacent to a given vertex, visit the adjacent vertex with the lowest value first. (10 points)

1 -> 2 -> 5 -> 4 -> 9 -> 7 -> 10 -> 3 -> 6 -> 8

1. On undirected graphs, does either of the two traversals, DFS or BFS, always find a cycle faster (i.e. in less operations) than the other? If yes, indicate which of them is better and explain why it is the case; if not, draw two graphs supporting your answer and explain the graphs. (10 points)

BFS would find a cycle much faster than DFS in most cases. Rather than traverse all adjacent vertices consecutively for the current vertex (until a cycle is reached), like BFS, DFS traverses a single adjacent vertex (with the lowest number) until a cycle is reached. This means that there will be a longer path traversing deeper into the graph to reach a cycle with DFS (as you are going for “depth”). If we take the graph above as an example (without directions), the first cycle is reached when at least 4 traversals. However, with BFS, the first cycle is reached after at least 2 traversals.

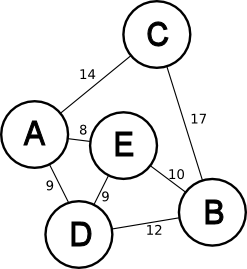
1. Consider the following graph:



List the order in which the vertices are visited with a topological sort, as shown in class. Always process the vertex with the lowest number first if several vertices have indegree 0. (10 points)

1 -> 4 -> 2 -> 5 -> 6 -> 8 -> 9 -> 7 -> 10 -> 3

1. Consider the following graph:

****

Apply Kruskal’s algorithm to find the minimum spanning tree. Edges are sorted first by length, and in the event of a tie, by name, where the two letters are in alphabetical order. Use makeset(x), find(x), and union(x, y) to determine if there are cycles.

1. Circle the edges that are part of the minimum spanning tree. (4 points)

AC, AD, AE, BC, BD, BE, DE

1. What is the weight of the minimum spanning tree? (1 point)

8 + 9 + 10 + 14 = 41

1. Draw the tree that results from applying the union-find algorithm for cycle detection. When drawing the tree, put vertices with lower letters in the left subtrees so that all the vertices in a level are sorted alphabetically from left to right. (5 points)



1. Using the same graph as in the previous question, use Prim’s algorithm to find the minimum spanning tree starting with vertex A. Fill in the table below, as shown in lecture. (10 points)

|  |  |
| --- | --- |
| Tree vertices | Remaining vertices |
| A(-,-) | B(-,inf) , C(A, 14) , D(A,9) , E(A,8) |
| E(A,8) | B(E,10) , C(-,inf) , D(A,9) |
| D(A,9) | B(E,10) , C(-,inf) |
| B(E,10) | C(A,14) |
| C(A,14) |  |

1. In a weighted graph, assume that the shortest path from a source 'S' to a destination 'T' is correctly calculated using Dijkstra’s shortest path algorithm. If we increase weight of every edge by 1, does the shortest path always remain the same? If so, prove it. If not, give a counterexample. (10 points)

The shortest path does not remain the same if every edge were to increase by 1. Let’s assume there was a path including 4 edges (3 vertices between S and T) from ‘S’ to ‘T’ (e.g., SU -> UV -> VW -> WT), each with a weight of 1. Let’s also assume that S and T are connected by a single edge with a weight of 5. If every edge’s weight were to increase by 1, then the shortest path from S to T would become the single ST edge (now 6), as opposed to its original shortest path with multiple edges(originally a weight of 4, now a weight of 8.

1. Suppose you have coins in a row with the following monetary value in cents: 7 1 3 2 1 7 5

You want to take coins such that you maximize the value of collection subject to the constraint that no two adjacent coins may be taken. Use dynamic programming to fill in the table. (7 points)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **C** | **0** | **7** | **1** | **3** | **2** | **1** | **7** | **5** |
| **F** | **0** | **7** | **7** | **10** | **10** | **11** | **17** | **17** |
| **S** | **0** | **0** | **1** | **1** | **3** | **3** | **4** | **6** |

What is the maximum amount of money you can take? (1 point)

The maximum is 17 cents

What coins did you take? List the indices of the coins taken from lowest to highest, assuming that first 7-cent coin is in index 1. (2 points)

The coins taken were the first 7-cent coin at index 1, the 3-cent coin at index 3, and the 7-cent count at index 6.