Assignment 2: Sorting, Algorithm Analysis, and Graphs

COSC 3020: Algorithms and Data Structures

Lars Kotthoff
larsko@uwyo.edu

09 October 2018

Instructions

Solve the following tasks. You may work in teams of up to two people. For the theoretic part, submit a PDF with your solution, for the practical part, submit the Javascript file(s) to WyoCourses. If you are working in pairs, only one partner needs to submit, but note in a comment to the submission who you've worked with. You have until Friday, 19 October 2018, 23:59h. If you submit files other than PDF and Javascript files, you may get no points for them.

You may *not* use external libraries in your code unless explicitly stated.

I will test your code on Linux with `node.js`. Please test it on the lab machines, where you have the same environment – if your code does not work, you may get no points for this part.

1 Theory vs. Practice (9 points)

- List at least 3 reasons why asymptotic analysis may be misleading with respect to actual performance in practice. (3 points)
- Suppose finding a particular element in a binary search tree with 1,000 elements takes 5 seconds. Given what you know about the asymptotic complexity of search in a binary search tree, how long would you guess finding the same element in a search tree with 10,000 elements takes? Explain your reasoning. (3 points)
- You measure the time with 10,000 elements and it takes 100 seconds! List at least 3 reasons why this could be the case, given that reasoning with the asymptotic complexity suggests a different time. (3 points)

2 Graph Properties (6 points)

- Prove that if two graphs A and B have the same number of nodes and are completely connected, they must be isomorphic. (3 points)
- Prove that if two graphs A and B are isomorphic they do *not* need to be completely connected. (3 points)

You need to give complete, formal proofs – state *all* your assumptions and make sure that you've explained every step of your reasoning.

Hint: A good way to start is by writing down the definitions for everything related to what you want to prove.

3 Merge Sort (10 points)

Implement an iterative (no recursive calls) and in-place version of merge sort. The prototype of the function should be the same as in the lectures:

```
function mergesort(x);
```

You may start from the implementation given in the lectures. Test your code by calling your merge sort function with a few different inputs and verifying that the result is correct.

Hint: To make merge sort in-place, think about what happens during the merge – where are elements moved to and from? To make it iterative, think about the part of the array each recursive call considers.

Analyse the time complexity of your implementation and give a Θ bound for its worst-case runtime.

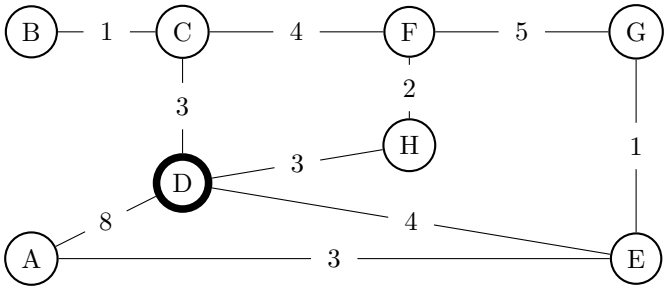
4 Quicksort (5 points)

In the lectures I mentioned strategies for determining a good pivot for quicksort. The implementation on the slides simply picks the leftmost element in the part of the array that we consider as a pivot. I also mentioned a few other ways of picking a good pivot.

Median-of-three is a better way of picking a pivot – inspect the first, middle, and last elements of the part of the array under consideration and choose the median value. Using the probabilities for picking a pivot in a particular part of the array (similar to what we did in the lectures, see slide 34), argue whether this method is more or less (or equally) likely to pick a good pivot compared to simply choosing the first element. Assume that all permutations are equally likely, i.e. the input array is ordered randomly.

Your answer needs to reason about probabilities that you derived for picking a good pivot.

5 Dijkstra's Algorithm (3 points)



Run Dijkstra's algorithm on the above graph to determine the shortest paths from vertex D to all other vertices. The graph is undirected, i.e. all edges can be traversed in either direction. The weight of each edge is the number on it.

Show the distance or distance estimate for each node after each iteration of the algorithm. You may break ties arbitrarily.