Assignment 1: Sorting and Algorithm Analysis

COSC 3020: Algorithms and Data Structures

Lars Kotthoff
larsko@uwyo.edu

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. The name and student ID of *all* partners must be clearly visible on each page of your PDF and in each source code file. Only one partner needs to submit.

I will test your code on Linux with `node.js`, checking whether your code works with empty inputs in addition to functionality. Please test it on the lab machines, where you have the same environment – if your code does not run, you may get no points for this part. If you submit a file other than a PDF or if your code is not in a separate file you may get no points for it. If you submit scanned or photographed handwritten notes, I will take off 10%.

1 Asymptotic Complexity

In the lectures, we said that logarithms with different bases don't affect the asymptotic complexity of an algorithm. Prove that $O(\log_3 n)$ is the same as $O(\log_{13} n)$. Use the mathematical definition of O – do a formal proof, not just the intuition. Start by giving the formal definition of O.

Total 5 points.

2 Runtime Analysis

Analyze the running time of the following recursive procedure as a function of n and find a tight big-O bound on the runtime for the function. You may assume that each operation takes unit time. You do not need to provide a formal proof, but you should show your work: at a minimum, show the recurrence relation you derive for the runtime of the code, and then how you solved the recurrence relation.

```
function mystery(n) {
    if(n <= 1)
        return;
    else {
        mystery(n / 3);
        var count = 0;
        mystery(n / 3);
        for(var i = 0; i < n*n; i++) {
            for(var j = 0; j < n; j++) {
                for(var k = 0; k < n*n; k++) {
                    count = count + 1;
                }
            }
        }
        mystery(n / 3);
    }
}
```

Total 5 points.

3 Sorting – Insertion Sort

Sort the list 4,3,1,0,-1,8 using insertion sort, ascending. Show the list after each outer loop. Do this manually, i.e. step through the algorithm yourself without a computer.

Total 5 points.

4 Sorting – Merge Sort

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

```
function mergesort(list);
```

You may start from the implementation given in the lectures. Test your code to make sure that it works as intended. You may, but do not need to, submit your test code in addition to your implementation of merge sort.

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.

Total 13 points.

5 Sorting – Quicksort

In the lectures I only briefly 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, e.g. randomly.

Median-of-three is also a good 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 (in the same way as we did on 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 must derive probabilities for choosing a good pivot and quantitatively reason with them.

Total 5 points.