COSC3320: Recursion

Instructor: Carlos Ordonez
University of Houston

1 Introduction

This is a written homewok emphasizing theory: correctness and time complexity. You need to produce a PDF in latex.

2 Homework requirements

1. PDF Format: IEEE double column
2. Notation: following textbooks. You must use different symbols for variable assignment and comparison, to avoid ambiguity with math symbols and programming symbols (e.g.$x \doteq 1$, $x \leftarrow 1$) and comparison with equality ($=$).
3. Arrays are indexed $1 \dots n$.
4. Correctness: In some questions you are asked to showq the P, Q, I (precondtion, postcondition and loop invariant).
5. Functions: arrays passed by reference, functions may return several values as a list.
6. You must write the algorithm in math notation. You cannot copy/paste source code into the latex document.
7. It is a good idea to program the algorithms in C++, Java or Python, but making sure arrays are indexed 1 to n (declare them $[n+1]$).
8. Derive tight time complexity $\Theta(g(n))$ bounds when possible. Otherwise, $O(f(n)) = g(n)$.
9. References: I know you will start searching the answers in Google. If you find anything beyond the textbooks, you must disclose the specific reference in your report.

3 Problems

1. write a recursive function to find the minimum element of an array A with n elements (sequential acceptable, recursive with halves acceptable). Assume no duplicates and the array is not sorted. Find $\Theta(f(n))$ and show all the steps to get n_0, c.. Convert to iterative algorithm and show the loop invariant I.
2. Write a Quicksort algorithm that has two subscripts (Hoare's version) for the partition function (iterative and recursive) and that can handle duplicate values. Estimate and contrast number of comparisons and swaps between Hoare's and Lemuto's. Show the loop invariant for the iterative partition function.
3. Slow and fast primality checking: to check if a number n is prime. Clarify $O()$ of input size (n), depending on the programmining language data type (infinite precision integers vs fixed size integers).

4. Show an example input array and an algorithm version where Quicksort has time complexity $\Theta(n^2)$. Show an example where Quicksort has time complexity $O(n\ log(n))$.

5. Write Mergesort in fully recursive functions (no loops for merge function). Does $\Theta(f(n))$ change? Show $T(n) = \Theta(n\ log(n))$ with recurrences. Write P, Q, I for the non-recursive merge function.

6. Write a recursive binary search function that can handle duplicates (if key or other elements are repeated) on array A. It returns the first and last subscript of the key (if found) as a list of 2 elements (they can be returned as an array). Show derivation of $T(n)$ with recurrences, counting number of comparisons. Write P, Q. Notice there is no I because there is no loop.

7. Recursive function to generate all 2^n subsets of a set of n numbers, bottom up or top down. Hint: look at the lattice. What is its $T(n) = \Theta(f(n))$?. How can you justify all possible subsets are generated (not missing any)?

8. You have an array of n=1M elements already sorted. How would you add δ =100k new elements (unsorted) and sort it efficiently again? $O()$? That is, avoiding a sort on the array with 1.1M elements. Discuss all potential ways to sort it, from ineffcient to most efficient. You can assume the array has enough space in main memory. Compute $T(\delta)$ where δ is a fraction of n.

9. Backtracking: Problem 5A Jeff's textbook (subsequence from a string). Show algortithm in pseudocode. Show $T(k, n)$

10. Greedy: Huffman codes for a long string of repeated symbols to compress it. Give an example on a string with length at least 30 and 6 letters with different frequencies.

11. Write a recursive $O(n^2)$ function(s) to sort a doubly linked list. Use next() and prev() to get the next and previous nodes as necessary. Derive $T(n)$. Transform it into two nested loops and show the loop invariant I.

12. Backtracking: Problem 6B Jeff's textbook. Symmetric binary B-trees (Red-black trees). Show a recursive algorithm to insert nodes into the tree, with or without backtracking. Compare O() with AVL trees.

13. It is possible to program MergeSort in a programming language that does not support recursion? Requirements: partition and merging, programming language has functions and arrays, computer has a stack (hint!). Show push() and pop() operations with subscripts.

14. $O(n^2 + n + 2^n + log(n) + n!)$=?, $O(2^n) = O(3^n)$?, $O((log(n))^2) > O(n)$?

15. Explain the three potential algorithms to solve the n-Queens problem, contrasting $O()$ and why they guarantee no queen attacks another queen (from Pandurangan book).

16. Fibonacci numbers: write a function with forward recursion, going from 1 to n, exploiting a table with partial results or 2 temporary variables. Compute $Theta(f(n))$. It this smart recursion (i.e. equivalent to dynamic programming)? (Erickson's book)

17. Based on the definitions, justify why $\Theta(2^n) < \Theta(4^n)$ (Pandurangan book, Ch.2).

18. Is it possible to program binary search using only linked lists? Is it possible to program merge sort using only linked lists?

19. Show a step by step example aligning two sequences of 3 symbols with a Naive, recursive and smart recursion (dynamic prog.) algorithms from Gopal's textbook (Ch.6).

20. Compare algorithms and show examples with figures/tables to solve the n-Queens problem for n=4..10 with backtracking and the faster solution based on $6t + 1, 6t + 4, 6t + 5$ and its generalization to $6t + 2, 6t + 3$ (exercise). Show solutions for $n = 9, 10$. (Pandurangan's book)