CSCI 314 – Fall 2012 Handout: asgn06
Algorithms
Michael H. Goldwasser
Saint Louis University Wednesday, 24 October 2012

Homework #6: Midterm Corrections
Due Date: Monday, 29 October 2012

Midterm Examination (Takehome Edition)

Name: ____

- In this homework, you are to complete all of the questions from our recent in-class exam.
- The homework is due at Monday, 29 October 2012. You are not otherwise limited to any specific time constraint for working on the homework (that is, you need not limit yourself to 50 minutes).
- You must work entirely on your own for this entire homework.
- You may use the book and any notes, including your graded midterm exam. However, you may not use any online materials.

(For grading use only)

Problem	Points	Grade
1	12	
2	12	
3	24	
4	24	
5	16	
6	12	
Total	100	

Problem #1 (12 points)

Give a formal proof that for any real constants $a > 0$ and $b > 0$,

$$(n + a)^b \text{ is } O(n^b).$$

Specifically, show that there exists constants $c > 0$, $n_0 \geq 0$ such that $(n+a)^b \leq c \cdot n^b$ for all $n \geq n_0$.

Problem #2 (12 points)

Give tight asymptotic bounds for the following recurrences.
(You do *not* need to give any justification for a correct answer)

- $T(n) = 4T(n/2) + n$.
- $T(n) = 4T(n/2) + n^2$.
- $T(n) = 4T(n/2) + n^3$.

Problem #3 (24 points)

Computers will often have a small amount of fast, primary memory, along with a large amount of slow, secondary storage, written on disk. For this reason, we can often predict the performance of an algorithm simply by counting the number of page accesses which take place to/from secondary storage.

We are interested in studying the efficiency of implementing a stack in such a model. We assume that our stack is maintained in secondary storage, and that each page of secondary storage will hold up to p items.

We consider two possible approaches. In the first approach, we are able to keep a copy of one page of the stack in the fast memory. When performing an operation, if the relevant stack page is already in our fast memory, then no page access from secondary storage is required. If, however, the current stack operation requires access to a page which is not currently in our fast memory, we will have to write the currently held page back to the disk, and fetch the new page into fast memory.

[6 points] (a) If only Push operations are allowed, what is the worst-case number of page accesses required for n such operations, expressed as a function of n and p? Briefly justify your answer.

[6 points] (b) If both Push and Pop operations are allowed, what is the worst-case number of page accesses required for n such stack operations, expressed as a function of n and p? Briefly justify your answer.

Please note that this problem continues on the following page.

Our second approach is to store two pages of the stack in our fast memory. Again, when performing a stack operation, if the relevant stack page is already in our fast memory, than no page access from secondary storage is required. If, however, the current stack operation requires access to a page which is not currently in our fast memory, we will need to free up room in fast memory by writing one of the two pages back to the disk. Of the two pages, we will choose to write that page which was least-recently used. After writing that page to memory, we can read in the new page which is needed for the current operation.

[12 points] (c) If both PUSH and POP operations are allowed, what is the worst-case number of page accesses required for n such stack operations, expressed as a function of n and p? **Justify your answer.**

Problem #4 (24 points)

Consider the following problem. We are given as input,

- A collection $X = \{x_1, x_2, \ldots, x_n\}$ of non-negative integers.
- An integer, T.

The goal is to find out whether or not there exists a subset of numbers from X which sums exactly to T. As it happens, dynamic programming can be used to solve this problem. Specifically, we can consider a set of subproblems, indexed by integers j and b, such that:

$$S(j,b) = \begin{cases} \text{TRUE} & \text{if there exists a subset of } \{x_1, \ldots, x_j\} \text{ which sums exactly to } b \\ \text{FALSE} & \text{otherwise} \end{cases}$$

The goal is then to compute $S(n,T)$.

[12 points] (a) Give a recursive formula which can be used to compute value $S(j,b)$. Please make sure you specify any necessary base case conditions.

[12 points] (b)

- How many distinct subproblems will be considered during the overall computation?
- How long will it take to compute a single such subproblem using the recursive formula you gave above?
- In what order could you compute those subproblems, if doing bottom-up dynamic programming.?
- What is the overall running time for solving the original problem?

Problem #5 (16 points)

Consider "Sample Graph 1" given on the last page of this exam.

(a) Draw the tree edges that result from a breadth-first search starting at vertex A.

(b) Draw the ordered, rooted tree(s) that results from a depth-first search, assuming that the outer loop processes vertices in alphabetical order, and that vertex adjacencies are reported in standard alphabetical order.

You are only to draw the tree edges.

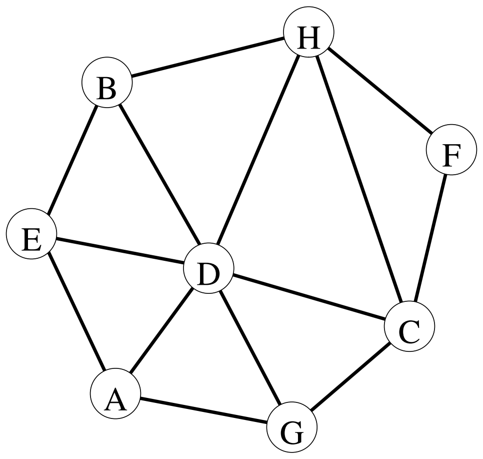
Furthermore, near each vertex v in your diagram, place a label designating discovery and finishing times $(v.d, v.f)$.

Problem #6 (12 points)

Consider "Sample Graph 2" given on the final page of the exam. Give <u>two different</u> topological orderings of the vertices, which are consistent with this graph.

(you are welcome to tear this page loose, so long as no answers are written upon it)

Sample Graph 1



Sample Graph 2