Basic Algorithms

Homework 1

Due: February 8th, 3 PM EST

Instructions

Please answer each **Problem** on a separate page. Submissions must be uploaded to your account on Gradescope by the due date and time above.
Please note that no late submission will be accepted for this homework.

Note

You have to wait until the lecture on February 2nd to solve Problems 3 and 4.

Problems To Submit

Problem 1 (13 points)

Use induction to prove the following for every positive integer n:

$$1 \times 1! + 2 \times 2! + \cdots + n \times n! = (n+1)! - 1.$$

Note that your answer should follow the following format:

1. Base case: Check that the statement holds for the base case $n = 1$.
2. Induction step:
   - Assumption: Assume that the statement holds for $n = k$.
   - Conclusion: Use the assumption to show that the statement holds for $n = k + 1$.

Recall: We have $n! = 1 \times 2 \times \cdots \times n$. Example: $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$.

Problem 2 (13 points)

Consider the following recursion with $a_1 = 3$, $a_2 = 5$, and for every integer $n \geq 2$ we have:

$$a_{n+1} = 3a_n - 2a_{n-1}.$$

Use strong induction to prove that $a_n = 2^n + 1$.
Note that your answer should follow the following format:

1. Base cases: Check that the statement holds for the base cases $n = 1, 2$.
2. Induction step:
   - Assumption: Assume that the statement holds for $n = 1, \ldots, k$.
   - Conclusion: Use the assumption to show that the statement holds for $n = k + 1$.

Problem 3 (12 points)

Rank the following functions in order of their asymptotic growth. That is, find an order $f_1, f_2, f_3, \ldots, f_6$, such that $f_1 = \mathcal{O}(f_2)$, $f_2 = \mathcal{O}(f_3)$, and so on. You do not need to provide an explanation of your ranking.

(a) 2^n

(b) $n^{1.5} \log_2 n$

(c) $n^2 - 1$

(d) $n!$

(e) $2^{\log_2 n}$

(f) 3^n

Problem 4 (6+6+6+6 points)

Prove or disprove: For each of the following, if it is true, then provide a proof, otherwise provide a counterexample and justify why the counterexample works.

(a) If $f = \mathcal{O}(g)$ then $g = \mathcal{O}(f)$.

(b) If $f = \Omega(h)$ and $g = \Omega(h)$, then $f + g = \Omega(h)$.

(c) If $f = \mathcal{O}(g)$ and $g = \mathcal{O}(h)$ then $f = \mathcal{O}(h)$.

(d) If $f = \Omega(g)$ and $h = \Omega(g)$ then $f = \Omega(h)$.

Problem 5 (7+7+7 points)

Let's learn another sorting algorithm, SELECTION_SORT! In each iteration, SELECTION_SORT finds the largest element among the remaining elements and place it at the end of them.
Example:

$$\begin{aligned} A &= [5, 3, 7, 10, 1] \\ &\Rightarrow [5, 3, 7, 1, 10] \\ &\Rightarrow [5, 3, 1, 7, 10] \\ &\Rightarrow [3, 1, 5, 7, 10] \\ &\Rightarrow [1, 3, 5, 7, 10] \end{aligned}$$

(a) Write the pseudo-code for SELECTION_SORT.

(b) Find the time complexity of SELECTION_SORT in the best and worst case scenarios.

(c) Now consider this variant of SELECTION_SORT: In each iteration, the algorithm finds the two largest elements among the remaining elements and place them at the end.
Is this variant more time-efficient compared to the original SELECTION_SORT? Justify your answer.

Problem 6 (7+10 points)

Consider the following functions which both take as arguments three n-element arrays A, B, and C:

```
COMPARE-1(A, B, C)
    For i = 1 to n
        For j = 1 to n
            If A[i] + C[i] ≥ B[j] Return FALSE
    Return TRUE
```

```
COMPARE-2(A, B, C)
    aux := A[1] + C[1]
    For i = 2 to n
        If A[i] + C[i] > aux Then aux := A[i] + C[i]
    For j = 1 to n
        If aux ≥ B[j] Return FALSE
    Return TRUE
```

(a) When do these two functions return TRUE?

(b) What is the worst-case running time for each function?

Bonus Problems

The following problems are optional. They will NOT be graded and they will NOT appear on any of the exams. Work on them only if you are interested. They will help you to develop problem solving skills for your future endeavors.
You are highly encouraged to think on them and discuss them with your peers on a separate thread on the course page on Campuswire.

Bonus Problem 1

We know that each of the first n positive integers, except one of them, appears in the array A exactly once. Thus, A has $n-1$ elements in total. Our goal is to develop an algorithm to find the missing number.

(a) Develop an algorithm which runs in linear time (in n) and requires linear amount of additional space (in n).

(b) Improving previous part, now develop an algorithm which runs in linear time (in n) and requires constant amount of additional space.

Bonus Problem 2

Consider the array B of size n which consists of n arbitrary positive integers (not necessarily the first n positive integers). Our goal is to find the least positive integer which is missing from B.

(a) Develop an algorithm which runs in quadratic time (in n) and requires constant amount of additional space.
Hint: Sort the array!

(b) Develop an algorithm which runs in linear time (in n) and requires linear amount of additional space (in n).

(c) Improving previous part, now develop an algorithm which runs in linear time (in n) and requires constant amount of additional space.
Hint: Use the array itself as the additional space!

(d) Can you do part (c) if the array B may also have negative elements?