Written Homework 5

Assigned:  Mon Nov 13 2017  
Due:  Wed Nov 29 2017

Instructions:

• The assignment has to be uploaded to blackboard by the due date. NO assignment will be accepted after 11:59pm on that day.

• We expect that you will study with friends and often work out problem solutions together, but you must write up your own solutions, in your own words. Cheating will not be tolerated. Professors, TAs, and peer tutors will be available to answer questions but will not do your homework for you. One of our course goals is to teach you how to think on your own.

• We require that all homework submissions be neat and easily readable. We recommend using a word processor like Microsoft Word or \LaTeX{} for your submissions. If you scan your homework, you may lose points if the scan is not legible.

• To get full credit, show INTERMEDIATE steps leading to your answers, throughout.

• You can give answers that are probabilities as either fully simplified fractions or decimal answers to at least two nonzero digits.

Problem 1 [12 pts (3,3,3,3)]: Sequences
For each of the following lists of integers, (1) indicate whether the sequence is “arithmetic,” “geometric,” “quadratic,” or none of these and (2) give a simple formula that generates the terms of this sequence, where your list elements begin at \( n = 1 \). For example, the sequence

\[
3, 5, 7, 9, 11, \ldots
\]

is arithmetic and generated by the formula

\[
a_n = 2n + 1
\]

starting at \( n = 1 \).

i.  –5, –1, 3, 7, 11, 15, \ldots

ii.  –2, 6, –18, 54, –162, 486, \ldots

iii.  0, 5, 16, 33, 56, 85, \ldots

iv.  6, 12, 24, 48, 96, 192, \ldots
Problem 2 [16 pts (4,4,4,4)] Sums
Evaluate the following sums. You must apply the methods given in class and in the text; i.e., you cannot simply add the numbers with a calculator or write a program. Show your work, and your final answer should be a single integer.

i. \(-9 - 4 + 1 + 6 + 11 + \cdots + 66\)

ii. \(2 + 6 + 18 + 54 + \cdots + 1458\)

Derive formulas in terms of \(n\) for the following sums. You must show your work, and your final formula should only contain \(n\) and integers (but not \(k\)).

iii. \(\sum_{k=4}^{n} (3k - 1)\)

iv. \(\sum_{k=3}^{n} 4 \cdot 5^{k+2}\)

Problem 3 [18 pts (6,4,8)]: Comparisons of Functions
In class, we discussed quadratic sorting algorithms (INSERTION-SORT and SELECTION-SORT) and an \(n \log n\) sorting algorithm (MERGE-SORT). Dozens, if not hundreds, of other sorting algorithms have been developed. Another well-known sorting algorithm is SHELL-SORT whose asymptotic running time is on the order of \(n \log^2 n\), when implemented appropriately.

In the problems that follow, you will compare these three algorithms for sorting. Ignoring lower order terms and constant factors, let \(T_1(n)\), \(T_2(n)\), and \(T_3(n)\) be the “effort” required by INSERTION-SORT, SHELL-SORT, and MERGE-SORT, respectively, to sort a list of length \(n\). We have

\[
T_1(n) = n^2 \\
T_2(n) = n \log^2 n \\
T_3(n) = n \log n
\]

where \(\log n\) is \(\log_2(n)\).

i. Suppose that you were given a budget of 100,000 units of “effort.” For each of the three algorithms, determine the largest list length such that the sorting effort required is guaranteed to be at most 100,000.

ii. How many times larger is the list that MERGE-SORT can handle, as compared to the lists that INSERTION-SORT and SHELL-SORT can handle? How many times larger is the list that SHELL-SORT can handle, as compared to the list that INSERTION-SORT can handle?

iii. Suppose you are running the three algorithms on three different computers. The computer running INSERTION-SORT is 5 times faster than the one running SHELL-SORT, and the computer running SHELL-SORT is 20 times faster than the one running MERGE-SORT. How large must the list be before the computer running MERGE-SORT begins to outperform the one running SHELL-SORT? How large must the list be before the computer running SHELL-SORT begins to outperform the one running INSERTION-SORT?
**Problem 4** [12 pts (3,3,3,3)]: Search Algorithms

In **binary search**, we split the list in half, perform one comparison to determine if our target element is in the first or second half, and repeat on the appropriate half as necessary until only one element remains. Since there are at most $\log_2 n$ halving operations, we use at most $1 \cdot \log_2 n = \log_2 n$ comparisons in the worst-case.

Now consider **ternary search** instead. Here we would split the list into thirds, perform (at most) 2 comparisons to determine which third contains our target element, and repeat on the appropriate third as necessary until only one element remains.

i. What is the worst-case number of comparisons performed by ternary search? Explain.

ii. Which algorithm performs fewer comparisons in the worst-case, and by how much? Your answer should be in the form, “Algorithm A performs $x$-times fewer comparisons than Algorithm B,” for appropriate values of A, B, and $x$. **Hint:** The following mathematical fact may come in handy; it allows one to change the base of a logarithm. $\log_a n = \log_b n / \log_b a$.

Let us now generalize to **k-ary search**, where we split the list into $k$ equal size groups and perform (at most) $k - 1$ comparisons to determine the appropriate group on which to repeat.

iii. What is the worst-case number of comparisons performed by $k$-ary search? Explain.

iv. What is the integer value of $k$ which minimizes the number of comparisons in the worst-case? Explain.

**Hint:** Ensure that your expression from part iii is in the form $f(k) \cdot \log_2 n$ for some function $f(k)$. To do this, make use of the fact that $\log_a n = \log_b n / \log_b a$, as you did above.

**Problem 5** [42 pts (8,6,8,6,6,8)]: Induction and Recurrences

1. Solve the following recurrence.

   $$T(n) = 8T(n/2) + n^3$$

2. The Fibonacci numbers 1, 1, 2, 3, 5, 8, ... is a sequence defined by the equation

   $$F_n = F_{n-1} + F_{n-2}$$

   where $F_1 = 1$ and $F_2 = 1$. Consider a currency system with Fibonacci denominations. In other words, unlike the US currency system which has $1, $5, $10, $20, $50, ... denominations, we
would have $1, $2, $3, $5, $8, $13, \ldots$ denominations. In what follows, we will show that the Fibonacci currency system is efficient in that very few bills are needed to make change for any value $d$.

Consider the standard greedy algorithm for making change: To make change for $d$, you would choose the largest denomination less than or equal to $d$, subtract that denomination value from $d$, and then repeat for the remainder. For example, to make change for $19$ in the US currency system, we would choose a $10$ bill, yielding a remainder of $9$. We would then choose a $5$ bill, yielding a remainder of $4$. We would then choose a $1$ bill, yielding a remainder of $3$, and so on. Our change would then be $10, 5, 1, 1, 1, 1$.

Applying the same greedy strategy in the Fibonacci system, our change would be $13, 5, 1$.

**i.** In making change for $d$ using the greedy strategy, let $F_n$ be the largest Fibonacci denomination value less than or equal to $d$. (1) Argue that $F_n \leq d < F_{n+1}$. (2) Prove that after choosing an $F_n$ bill, the remainder $d - F_n$ must satisfy $d - F_n < F_{n-1}$.

**ii.** Prove by induction that for any dollar value $d$, you can make change for $d$ without using any denomination more than once and without the use of adjacent denominations. For example, while you could make change for $14$ using two $5$ bills, one $3$ bill, and one $1$ bill, you would be using a denomination twice (the $5$ bill) and you would be using adjacent denominations (the $5$ and $3$ bills). One could instead use one $13$ and one $1$ bill, which are non-adjacent and without repetition.

*Hint:* Make use of the results from part i.

**iii.** Prove that if change is being made with $n$ denominations $d_1, d_2, \ldots, d_n$ and adjacent denominations are not allowed, then at most $n/2$ unique denominations can be used. For simplicity, you may assume that $n$ is even; the more general result for $n$ odd or even is that at most $\lceil n/2 \rceil$ unique denominations can be used.

*Hint:* Try a proof by contradiction, group the denominations in adjacent pairs, and employ the Pigeonhole Principle.

**iv.** Prove by induction that $\forall n \geq 6, F_n \geq 2^{n/2}$. Note that $2^{n/2} = (\sqrt{2})^n \approx 1.414^n$.

**v.** Given the results of the previous parts, prove (not necessarily by induction) that $\forall d \geq 8$ it is always possible to make change for $d$ in the Fibonacci currency system using at most $\log_2 d$ bills.

*Hint:* Start by considering the largest Fibonacci denomination $F_n$ in any such solution and then use the result from part iv above to bound the size of $n$. You will then need the results from other parts above to finish the proof.