Problem 1-1 (Order of growth) 12 pts

(a) For each of the following functions \( f(n) \), find a “canonical” function \( g(n) \) such that \( f(n) = \Theta(g(n)) \). Briefly justify your answers (and I mean briefly).

\[ 2^n + 3n^{17}, \quad 2^{n+1}, \quad \log(n^{23} + 3n^2 + 5), \quad 2^{2n}, \quad \sqrt{n^7 + 3n^4 + 11}, \quad \frac{n^5 - n}{10000}, \quad \log^3 n + 5, \quad 4n \log n \]

(b) Based on your answers in part (a), sort the functions above in asymptotically increasing order. Are there any two functions with the same order of growth? If yes, which ones?

Problem 1-2 (Adding Binary Integers) 5 points

You are given two binary integers \( a \) and \( b \), where \( a \) has \( n \) bits, and \( b \) has \( m \) bits, where \( n \geq m \). These integers are stored in two arrays \( A[1\ldots n] \) and \( B[1\ldots m] \), in reverse order. For example, if \( a = 111000 \) and \( b = 1110 \), then \( A[1] = A[2] = A[3] = 0, A[4] = A[5] = A[6] = 1, B[1] = 0, B[2] = B[3] = B[4] = 1. \) Your goal is to produce an array \( C[1\ldots n+1] \) which stores the sum \( c \) of \( a \) and \( b \). For example, \( 111000 + 1110 = 1000110 \), meaning that \( C[1\ldots 7] = 0110001 \).

Write the pseudocode to produce \( C \). Why was it convenient to store the arrays “backwards”?

Problem 1-3 (Strange Code Segment) 12 points

Consider the following code segment.

```
DIVISORS(n):
    For i = 1 to n
        \( A[i] := 0 \)
    For i = 1 to n
        j := i
        While j \leq n
            j := j + i
```

\(^1\)I.e., function of the form \( a^n n^b \log^c n \) for \( a \geq 1 \) and \( b, c \geq 0 \).
(a)* As a function of j, describe in words the value that each array element \( A[j] \) stores at end end of the procedure.

(Hint: Look at the name of the procedure for a hint.)

(b) Compute the exact running time \( T(n) \) of the above procedure in terms of a single summation \( \sum_{i=1}^{n} f(i) \), where \( f(i) \) is the appropriate function you need to determine for this part. Do not attempt to compute the summation yet. Do not worry about “floors” and “ceilings”.

(c) Try to estimate the asymptotic value of \( T(n) = \sum_{i=1}^{n} f(i) \) using the “bounding terms” method. Do you seem to get a useful result? I.e., does this method seem to give you a “nice” function \( g(n) \) for which you can conclude \( T(n) = \Theta(g(n)) \)?

(d) Now, approximate the value \( T(n) \) by \( \int_{x=1}^{n} f(x) \, dx \). Compute this integral to derive the function \( g(n) \) such that \( T(n) = g(n) + o(g(n)) \). Notice, constants matter in the definition of \( g(n) \), since I’m not merely looking for \( T(n) = \Theta(g(n)) \)!

Problem 1-4 (Insertion Sort) 12 points

(a) Observe that the while loop of Insertion sort uses a linear search to scan (backward) through the sorted subarray \( A[1 \ldots j - 1] \). Can we use a binary search instead to improve the overall worst-case running time of insertion sort to \( \Theta(n \log n) \)?

(b) In class we learned how to implement insertion sort by comparing the key element to the largest element in the sorted portion of the array, and moving that element to the right, if it was larger than the key, and then comparing the key to successively smaller elements until the right position is found.

Implement a variation of insertion sort, in which you instead compare the key to the smallest element in the sorted portion of the array and then iterate by comparing to successively larger elements.

How does this algorithm compare in terms of efficiency to the traditional insertion sort?

\(^2\)You do not have to formally argue that the integral gives you such an accurate approximation.

PS1, Page 2