**COMP3270**

**Fall 2014**

**Homework 1 Solutions**

**Exercise 2.2-1)** Express the function n3/1000 – 100n2 - 100n + 3 in terms of big-theta notation

*Solution:* Throw away constants and lower-order terms for a polynomial, yielding

Θ(n3)

**Exercise 2.2-3)** In the linear search, how many elements must be searched on average, assuming the element is equally likely to be any element in the array? How about in the worst case? What are the average and worst-case running times of linear search in big-theta notation? Justify your answer.

*Solution:*  Assuming every value is equally likely to be the element being searched for, the average number of elements to be searched would be

(1+2+3+… + n-1)/(n-1) = (n\*(n-1))/(2\*(n-1) = n/2

The worst case would be to search them all, or n-1 elements

Both of these are Θ(n) after throwing away the coefficient. That result could also be easily proved by the definition of big-theta.

**Exercise 2.3-1)** Illustrate the operation of merge sort on the array A = <3,41,52,26,38,57,9,49>

<3,9,26,38,41,49,52,57>

<3,26,41,52> <9,38,49,57>

<3,41> <26,52> <38,57> <9,49>

<3> <41> <52> <26> <38> <57> <9> <49>

*Solution*:

**Exercise 2.3-4)** We can write insertion sort as a recursive function. To Sort A[1..n] we recursively sort A[1..n-1] and then insert A[n] into the sorted array A[1..n-1]. Write a recurrence for the worst case running time of this recursive version

*Solution:*

*T(n) = T(n-1) + n-1*

**Problem 2.2-d)** What is the worst-case running time of bubblesort? How does it compare to the running time of insertion sort?

*Solution:* In the worst case, each comparison in line 3 (p.40) for bubblesort will cause a swap. However, that only takes a constant amount of time, and each pair must be compared to see if a swap is necessary. This means that the running time will be

**Problem 2.3-a)** What is the Θ(n) running time for the code fragment of Horner’s rule:

y=0

for i=n downto 0

y = ai+x\*y

*Solution*: The first line takes 1 step. The loop executes n+1 times. The assignment statement takes 3 steps (multiplication, addition, assignment), in any event a constant amount of time. Thus, the entire code fragment take *c1 + (n+1)\*c2* time for some constants c1 and c2, which is Θ(n).

b) Write pseudocode for the naïve polynomial evaluation that computes each term from scratch. What is the running time? How does it compare to Horner’s rule?

*Solution*: The naïve evaluation computes term by term:

*P(x) = a0+a1x+a2x2+a3x3+ … + anxn*

The pseudocode looks like

y=0

for i= 0 to n

term = ai

for j = 1 to i

term = term\*x

y = y + term

The inner loop takes a constant amount of time to execute, and executes i times. The outer loop iterates n+1 times and takes a constant amount of time plus the time for the inner loop. Thus, the entire algorithm takes time

=