CS566 Homework-2

Due – Feb 12, 6:00 PM, at the class beginning

Student's Name: _Kevin Lopez Sepulveda

Upload copy of your homework and any supporting materials (source code and test cases) online to Assignment2 in the CS566_A2 course site

No extensions or late submissions for anything other than major emergency.

(1) [15 pts]

For each of the following pairs of functions, either f(n) is in O(g(n)), f(n) is in Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct and briefly explain why.

(a) f(n) = log(n^2); g(n) = log(n) + 5

   a. f(n) = 2log(n)
   b. g(n)=log(n)+5
   c. they run at similar times because they are both log(n)
   d. f(n) = Θ(g(n)) because theta is similair

(b) f(n) =√n; g(n) = log(n^2)

   a. f(n)=n^1/2
   b. g(n)= 2log(n)
   c. f(n) grows faster than log(n)
   a. f(n)= Ω(g(n)),

(c) f(n) = n; g(n) = nlog2 n

   a. g(n) grows faster than f(n)as n increases because log2(n)>1 for large n.
   b. f(n) = O(g(n))

(d) f(n) = 10; g(n) = log10(n?)

(e) f(n) is constant

(f) g(n) grows with n?(Assume log 10 n)

(g) f(n) = O(g(n))

(h) if log 10 is just constant then it f(n) = Θ(g(n))

(f) f(n) = 2^n; g(n) = 10n^2

Exponential functions grow faster

f(n)= Ω(g(n)),

(2) [20 pts] use Master Theorem, Master Theorem Extension or Substitution method

(a) T(*n*) = 2T(*n*/3) + *n***3
   a. A = 2, b = 3, f(n) = n^3
   b. N^log3(2) is less dominant than n^3
   c. T(n)=Θ(n^3)

(b) T(*n*) = T((9/10) *n*) + sqrt(n)
   a. a=1,b=910,f(n)=n^1/2
   b. n^0 is less dominant than n^1/2
   c. T(n)=Θ(n^1/2)

(c) T(*n*) = 16T(*n*/4) + *n***2
   a. A=16, b = 4, f(n)= n^2
   b. They are equal. N^log4(16)
   c. T(n)=Θ(n^2log(n)

(d) T(n) = T(n-3) + n
   a. T(n−6)+(n+(n−3))… n^2/6
   b. T(n)=Θ(n^2)

(e) T(*n*) = 3T(*n*-1) + 1, T(1)=1
   a. 3^2T(n-1) + 1 … 3^n
   b. T(n)=Θ(3^n)
   c.

(3) [25 pts.] How many inversions (swaps) requires sorting the following lists?
A pair of elements (p_i, p_j) is called an inversion in a list permutation p if $i > j$ and $p_i < p_j$.
Show your work in details, name a sorting algorithm in use

(a) (5,2,3,4,1)

a. To count the inversions in the list (5,2,3,4,1) we can utilize the merge sort algorithm, which effectively counts inversions while sorting the array. Initially, we split the array into two halves: (5,2) and (3,4,1). In the left half (5,2), we further split it into (5) and (2). When merging these two single elements, we find that (5,2) contains one inversion because 5>2, resulting in a merged sorted array of (2,5) with 1 inversion. Next, we focus on the right half (3,4,1), which we split into (3) and (4,1). The sub-array (4,1) can be split into (4) and (1)which also results in one inversion when merging, as 4>1. This gives us a sorted array of (1,4) with 1 inversion. Now, merging (3) with (1,4) reveals another inversion, (3,1) The merged result is (1,3,4) with a total of 2 inversions from this step. Finally, we merge the left and right halves (2,5) and (1,3,4). Here, 2>1and 5>1 count as 2 additional inversions, and 2>3 and 2>4 add 2 more. The total inversions for the entire array come to 7, as counted during the merge operations.

(b) (1,2,3,4,5)

a. The array (1,2,3,4,5) is already sorted in ascending order, which means there are no inversions present. An inversion occurs when a larger number precedes a smaller one, and since all elements in this array are in order, there are 0 inversions. Using the merge sort algorithm confirms this as we find that no pairs (pi,pj) exist such that i<j and pi>pj. Thus, the total count of inversions in this case is simply 0.

(c) (5,1,2,3,4)

a. To count the inversions in the list (5,1,2,3,4), we apply the same merge sort method. First, we split the array into two halves: (5,1) and (2,3,4). The left half (5,1) can be split into (5) and (1). During the merge, we find that (5,1) contains one inversion since 5>, resulting in a sorted array of (1,5) with 1 inversion. Next, we handle the right half (2,3,4) which is already sorted and thus contains 0 inversions. Now we merge (1,5) with (2,3,4). During this merge, we observe that 5>2, 5>3 and 5>4, adding 3 more inversions. Therefore, the total number of inversions for the array (5,1,2,3,4) sums to 4 (1 from the left half and 3 from the merge with the right half). In conclusion, the total counts of inversions for the list is 4 inversions

(4) [25 pts.] Let us consider the easiest sorting algorithms – Maxsort.
It works as follows: Find the largest key, say max, in the unsorted section of array (initially the whole array) and then interchange max with the element in the last position in the unsorted section. Now max is considered part of the sorted section consisting of larger keys at the end of the array. It is no longer in the unsorted section. Repeat this until the whole array is sorted.

a) Write an algorithm for Maxsort assuming an array E contains n elements to be sorted, with indexes 0, 1,…,n-1.

```
def maxsort(E):
    n = len(E)
    for i in range(n - 1, 0, -1):
```

```
    maxIndex = 0
    for j in range(1, i + 1):
        if E[j] > E[maxIndex]:
            maxIndex = j
    E[maxIndex], E[i] = E[i], E[maxIndex]
  return E
```

b) How many comparisons of keys does Maxsort do in the worst case and on average?

- In the first pass, we compare n−1n-1n−1 elements to find the maximum.
- In the second pass, we compare n−2n-2n−2 elements, and so on.
- Total comparisons:

(n−1)+(n−2)+(n−3)+...+1=n(n−1)2(n-1) + (n-2) + (n-3) + ... + 1 = n(n-1)/2

Submit source code, test cases, results, and the answers to part (b).

(5) [15 pts]

Come up with algorithm to convert 2025 using Roman numerals. Write a code for it.
What about 3998 and 5999. Please provide the Roman representation of all input numbers.