CISC-235 Data Structures
Assignment 1
Happy Mid-Autumn Festival!

September 17, 2019

1 Big-O, Big-Ω, and Big-Θ Analysis and Proofs: 40 points

1) True or False and then proof: n^{13} is not $O(n^{10})$.
Hint: first determine if this is a true or false statement and then proof that it is true/false.

2) let $T(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$, i.e., T(n) represents the general form of polynomials, where $a_0, a_1, \ldots a_k$ are real numbers with $a_k \neq 0$. Does T(n) have a Big-Θ estimation? If it has, what is the Big-Θ of T(n)? Proof your claim.

3) Proof that if h is O(g) and g is O(f), then h is O(f)

2 Time Complexity Analysis of Bubble Sort: 30 points

Write you own code (pseudo or runnable) to implement bubble sort, count the number of operations required based on your own code, e.g., determine the T(n) of your program. Note that you need to consider the best, average and worst case. You must give the details how you calculate the number of operations. Next, analyze the worst case time complexity of bubble sort, i.e., performing Big-O and Big-Ω estimation on your T(n). No proof needed.

Hint: Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

3 Runtime Analysis of Programs: 80 points

Let us analyse the runtime complexity of two algorithms, i.e., linear search (Algorithm A) and binary search (Algorithm B) using experimental method.

Algorithm A: Store the list S in an array or list, in whatever order the values are given. Search the list from beginning to end. If $S[i] == x$ for any value of i, stop searching and return True (i.e., found), otherwise return False.

Algorithm B: Store the list S in an array or indexed list (such as the Python list). **Sort the list**. Use binary search to determine if x is in S. Return True or False as appropriate.

When using Algorithm A, searching for a value that is in the list will require an average of $n/2$ comparisons, while in the worse case, searching for a value that is not in the list will require n comparisons. When using Algorithm B, searching for any value will require an average of about $logn$ comparisons. However, sorting the list will take $O(nlogn)$ time.

If we are doing a very small number of searches, Algorithm A is preferable. However if we are doing many searches of the same list, Algorithm B is preferable since the time required to sort the list once is more than offset by the reduced time for the searches. This is what complexity theory tells us.

Your task is to conduct experiments to explore the relationship between the size of the list and the number of searches required to make Algorithm B preferable to Algorithm A. See the detailed requirement below:

1) Implement two algorithms using Python. When implementing Algorithm B, you must write your own sort function and your own binary search function. You may use any sort algorithm that has complexity in $O(nlogn)$.

2) For n = 500, 1000, 5000, and 10000, conduct the following experiment:

   - Use a pseudo-random number generator to create a list S containing n integers.

   - For values of k ranging from 10 upwards:

   - Choose k target values, make sure half of which are in S and half are not in S,

   - Use Algorithm A to search the list S for the k target values. Use Algorithm B to search the list S for the k target values.

   - Determine the approximate smallest value of k for which Algorithm B becomes faster than Algorithm A – note that this may be greater than n. Call this value F(n). Create a graph showing your values of F(n) as a function of n.

Hints:

1) To easily create a list of search values, half of which are in S and half of which are not: - when generating the list S, use only even integer values -

randomly choose some elements of S as the first half of the search list - randomly choose odd integer values as the second half of the search list

2) Use the timeit module of Python to record running time.

How to Submit

You need to prepare a folder which contains your answer and code. For question 1-3, write your answers in a text file and submit in the pdf format. You need to submit your code for question 3. Zip the folder and upload it to OnQ. You must comment your code. If you have any question regarding this assignment, you can post a question on Piazza or send me an email, or come during office hours. The deadline will be specified on Onq.