Assignment 1

CS 452/652/752 Advanced Algorithms and Applications

Out: September 4 2018; due: September 11 2018

1 Important information

1. Total available points is 112.
2. Upto 112 points can be scored. Grading to be done for 100 points, 12 points are for extra credits.
3. Question 1 is worth 4 points and is **compulsory**.
4. Question 2 3, 4, 5, 6, and 7 are each worth 12 points, One can decide to not solve one of these.
5. Questions 8, 9 and 10 are programming questions, preferred language of coding is **python**.
6. Questions 8, 9 and 10 are **compulsory** to solve, totalling 36 points.

2 Submission instructions

1. All questions need to be handed in digitally through the canvas system.
2. For questions 1 to 7, one is expected to submit typed solution (preferably using latex).
3. For programming question, one needs to have three folders for the three problem sets.
4. Folder name should follow the template **Question_8_name_blazerID**.
5. Each folder should have the code in one file and set of test cases (tried for the program) in other file. Everyone is also required to have a `readMe` text file for each of the questions, detailing the instructions to run the code.
6. Any IO system can be used, example, file IO or console IO.
7. Comment the code extensively.

Question 1 *[4 points]*

Express the following function in terms of Big-oh notation:

1. $(n^3)/1000 - 100 * n^2 + 50$
2. $n^a + n^b$ ($a > b$ and $b > 0$)

Question 2 *[12 points]*

Consider the following modification to the `MergeSort` algorithm: divide the input array into thirds (rather than halves), recursively sort each third, and finally combine the results using a three-way Merge subroutine. What is the running time of this algorithm as a function of the length `n` of the input array, ignoring the constant factors and lowest order terms ? [Hint: the `Merge` subroutine can still be implemented so that the number of operations is only linear in the sum of the input array length.]

Question 3 *[12 points]*

Suppose you are given `k` sorted arrays, each with `n` elements, and you want to combine them into a single array of `kn` elements. Our approach is to use the linear time Merge subroutine [`O(n)` runtime] repeatedly, first merging the first two arrays, then merging the result with the third array, then with the fourth array, and so on until you merge in the `k-th` and the final input array. What is the running time taken by this successive merging algorithm, as a function of `k` and `n`, ignoring constant factors and lower-order terms.

Question 4 *[12 points]*

Consider again the problem if merging `k` sorted length-`n` arrays into a single sorted length-`kn` array. Consider the algorithm that first divides the `k` arrays into `k/2` pairs of arrays, and use the merge subroutine to combine each pair, resulting in `k/2` sorted length-`2n` arrays, The algorithm repeats this step until there is only one length-`kn` sorted array. What is the running time taken by this successive merging algorithm, as a function of `k` and `n`, ignoring constant factors and lower-order terms.

Question 5 *[12 points]*

Arrange the following functions in the order of increasing growth rate, with g(n) following $f(n)$ in your list if and only if $f(n) = O(g(n))$.

1. $\sqrt{n}$
2. 10^n
3. $n^{1.5}$
4. $2^{\sqrt{log_2 n}}$
5. $n^{5/3}$

Question 6 *[12 points]*

Recall the partition subroutine employed by the `Quicksort` algorithm. You are told that the following array has just been partitioned around some pivot element: 3, 1, 2, 4, 5, 8, 7, 6, 9

Which of the elements could have been the pivot element ? (List all that apply; there could be more than one possibility).

Question 7 *[12 points]*

Insertion sort can be expressed as a recursive procedure as follows. In order 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 (`T(n)` as a function of input size `n`) for the running time of this recursive version of insertion sort.

Question 8 *[12 points]*

For input strings **s1** and **s2** , write a function to determine if **s1** is an anagram of **s2**.

Example 1: Input: s1 = "palindrome", s2 = "dromepalin" Output: true
Example 2: Input: s1 = "cat", s2 = "mat" Output: false

Question 9 *[12 points]*

Determine for a given string, if it is a palindrome. Consider only alphanumeric characters (ignore others) and ignore cases. Note, empty string is a valid palindrome.

Example 1: Input: "A tan, : nata" Output: true
Example 2: Input: "race, a car" Output: false

Question 10 *[12 points]*

An array **A** is *always increasing* if for all $i <= j$, $A[i] <= A[j]$. An array **A** is *always decreasing* if for all $i <= j$, $A[i] >= A[j]$. An array is *monotonic* if it is either always increasing or always decreasing. Return true if and only if the given array A is *monotonic*.

Example 1: Input: [5,6,6,8] Output: true
Example 2: Input: [5,8,7] Output: false
Example 3: Input: [1,1,1] Output: true