CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

Problem 1 (8 points)

For each of the following statements, either prove that it is true, or give a counterexample demonstrating that it is false:

1. **If $f(n) = \Theta(g(n))$ then $2^{f(n)} = O(3^{g(n)})$**
   Sol:

   False

   We know that If $f(n) = \Theta(g(n))$, only if $f(n)=O(g(n))$ and $f(n)= \Omega(g(n))$

   For example:
   $f(n)= 2n$
   $g(n)= n$

   $2^{f(n)} = 2^{2n}$
   $3^{g(n)} = 3^{n}$

   But 2^{2n} is not <= 3^{n}.

2. **If $f1(n) = O(g1(n))$ and $f2(n) = O(g2(n))$ then $f1(n) + f2(n) = O(g1(n) + g2(n))$.**

   Sol:

   True

   We know that If $f1(n) = O(g1(n))$ and $f2(n) = O(g2(n))$ then $f1(n) + f2(n) = O(g1(n) + g2(n)$.

   Proof:

   $F1(n) <= c1g1(n)$ and $F2(n) <= c2g2(n)$ for large n. Thus,

   $$f1(n) + f2(n) <= c1*g1(n)+c2*g2(n)$$

   $$<= (c1+c2) * (g1(n)+g2(n))$$

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

3. **If $f_1(n) = O(g_1(n))$ and $f_2(n) = o(g_2(n))$ then $f_1(n) + f_2(n) = o(g_1(n) + g_2(n))$.**

Sol:

False

For example:
$F1(n) = n^2$
$F2(n) = n$

$G1(n) = n^2$
$G2(n) = n^2$

$f_1(n) + f_2(n)$ is not equal $o(g_1(n) + g_2(n))$

as the below argument is not true:

$n^2 + n < (2n^2)$

According to the previous problem proof, If f1(n) = O(g1(n)) and f2(n) = O(g2(n)) then f1(n) + f2(n) = O(g1(n) + g2(n).

4. **$f(n) = \Theta(f(n/2))$**
Sol:

False

For example:
$F(n) = 2^n$

$F(n/2) = 2^{n/2}$

$2^n > 2^{n/2}$

$2^n \neq \Theta(2^{n/2})$

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

Problem 2:

Solution:

Ascending Order

Equivalence class A: $1/n$, $1/1000$, 10^{100}

Equivalence class B: $\log \log n$

Equivalence class C: $\sqrt{\log n}$

Equivalence class D: $\log n$, $\log_{10} n$, $\log n^2$

Equivalence class E: $\log^2 n$

Equivalence class F: $\sqrt{n}$

Equivalence class G: $n^{2/3}$

Equivalence class H: $\log n$, $2^{\log n}$, $\log 2^n$, $n + \log n$

Equivalence class I: $n \log n$

Equivalence class J: $n^{3/2}$

Equivalence class K: n^2, $n + n^2/10^{20}$

Equivalence class L: 2^n, 2^{n+1}

Equivalence class M: $n * 2^n$

Equivalence class N: 3^n

Equivalence class O: 2^{2n}, $4n$

Equivalence class P: $n!$

Equivalence class Q: $(n+1)!$

Equivalence class R: n^n

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

Problem 3:

You are given A[0...n + 1], a sorted array that contains all of the non-negative integers in the range [0, n], inclusive, as well as one duplicate value. Design an o(n) algorithm that determines which number is present twice in the array. Note the requirement is o(n), not O(n). For this problem, you should ignore the cost associated with reading the input. You should also not do any processing of the input as it is read into the array. Design your algorithm assuming the data is already in the array as described above.

Solution:

Algorithm Analysis:

In this program, we have used the binary search algorithm to find the single duplicate value. We have implemented this algorithm using Recursion. In the recursion step, we tend remove almost half of index positions by checking if the list value at the middle index of the list is less than or greater than/equal to the mid index value. This checking and the recursion step works correctly as the range of numbers is [0, n]. Once we reach the base step, where in we check if there are two consecutive duplicate numbers present, we stop and display the result.

Complexity Analysis:

To get input from user it takes O(n) time which is ignored. As this program is based on the Binary search Algorithm with some little tweaks based on the requirements, the overall time complexity is O (log n).

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

Problem 4:

Solution:

Algorithm Analysis:

In this algorithm, we pair elements according to their preferences. A Match is said to be unstable if there is other element which want to get paired up so for that preference is checked between current partner and new partner if new partner has higher preference, then new partner will become his new pair and old partner would be set free. The correctness of algorithm can be argued as while pairing elements of 2nd group according to first group element's preference we can start from any first group element and end up with same output of pairs being formed. For ex: while pairing students according with first group element's preference we can start pairing from any first group element say 2 or 4 but we will end up pairing same irrespective who is paired first. In the algorithm implemented we have paired the first group element and 2nd group element according to first group element's preference and vice versa. Then we check if the pairings formed according to first group element's preference and 2nd group element's preference are the same and accordingly print the result.

Complexity Analysis:

The time complexity of Algorithm:

To get I/p from user it takes O(n) time.

To assign free elements it takes O(n) time.

For pairing elements, it takes $O(n^2)$ time.

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

Problem 5:

Implement MergeSort, InsertionSort and BucketSort.

Solution:

1. Merge Sort:

Algorithm Analysis:

This algorithm was taught in CSCI 603: Computational Problem Solving (Fall 2021) and utilizes the same algorithm.

Credits: Professor Maria Cepeda.

Basic Idea: Recursively splits the input list into two halves and goes on splitting until the all the newly generated lists are sorted.

Follows a Divide and Conquer approach.

Divide: Divides the list until it's sorted (has only one element).

Conquer: Combine all sorted lists together to form a single sorted list.

Base Case: A list is sorted if the length of the list is equal to 1.

Recursive Case: The list consists of more than one element.

Once the two halves are sorted, we perform the merge step. Merge step combines the two sorted lists into one sorted list.

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

Time Complexity Analysis:

Time Complexity: O(n * log n)

T(n) =

MergeSort(X, n):

Base Condition if n(length of array A) == 1, then return X (Sorted Array) - O(1)

Calculate the middleIndex = len(A) // 2 using formula - O(1)

Use the middleIndex to split the array into two halves

A = X[0: middle - 1] - O(n)

B = X [middle: n] - O(n)

As = MergeSort[A, middle] - T(n/2)

Bs = MergeSort[B, n- middle] - T(n/2)

return Merge(As, Bs) - O(n)

T(n) = O(n) + 2 * T(n/2)

T(n) = O(n * log n)

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

2. Insertion Sort:

Algorithm Analysis:

Basic Idea: Compare key with values at previousIndex.

For n-1 iteration:

Do (In every iterations)

As long as previousIndex is greater than or equal to Zero
AND
the value of key is less than the value at the previousIndex:

Copy the value (larger value in this case) at previousIndex to location previousIndex + 1.
Decrement the previousIndex by 1.

Check for while loop condition again, and keep moving larger values by one position.

Once the loop reaches and index = 0 or an index where key value is greater than
the previous value, copy the key value at that location

Repeat the above steps for n-1 iterations.

After completion of all (n-1) iterations, the input array will be sorted in ascending
order.

Time Complexity Analysis:

Time Complexity: O (n^2)

O (n): To iterate over the array of length n

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

O (n): To shift the key to the appropriate location in the array.

3. Bucket Sort:

Algorithm Analysis:

Basic Idea: Divide the input (uniformly distributed) values into array of size of input where each input array location is a linked list.

Each array location can be called as a bucket which holds data in a particular range, say for input values in range [0, 1) and 8 input values, there will be 8 locations in the array where, index_0 holds data in range 0 - 0.125 index_1 holds data in range 0.125 - 0.250 and so on.

Time Complexity Analysis:

Time Complexity: Expected Run time: O(n)

Since the input elements n are uniformly distributed, as per mathematical analysis, the input will be distributed even in the given range [0, k). There we will roughly have equal number of input values in each bucket, where we have created n buckets for n input values.

So for truly uniform data, we will get roughly one input value in each bucket we created.

Now, independent of the sorting algorithm we use, we are guaranteed we will have a constant number of input value(s) in each bucket which can be sorted in constant time.

Best Case: O(n)

Each linked list takes ends up with a single element and insertion sort takes constant time per list.

CSCI 665: Foundations of Algorithms
Homework 1

Submitted by – Aniket Narendra Patil (ap8504)
Aditya Ajit Tirakannavar (at2650)

Total cost of insertion sort is Θ(n)

Worst Case: $O(n^2)$

All elements end up in a single list and insertion sort takes $O(n^2)$ to sort all elements.

Running Time Analysis:

Time Analysis for different sorting algorithms (secs)						
Input Value	Merger Sort		Insertion Sort		Bucket Sort	
	Uniform Distribution	Gaussian Distribution	Uniform Distribution	Gaussian Distribution	Uniform Distribution	Gaussian Distribution
n = 100	1.42	1.14	0.79	1	0.94	1.03
n = 1000	1.51	1.5	1.33	1.43	1.23	1.52
n = 10000	1.54	1.68	3.82	4.14	1.72	3.45
n = 100000	2.63	2.86	180.23	194.95	2.76	8.37