Northeastern University
CS5800 – Algorithms
Spring 2022, Jonathan Mwaura

Course Synthesis #1

Name: _____

Problem	Points
Problem #1 - Short Answer Questions	/10
Problem #2 - Guess My Word	/10
Problem #3 - Sorting Algorithms	/10
Problem #4 - Merge Sort Comparisons	/10
Problem #5 - Graph Coloring	/10
Total	/50

Instructions

- This Synthesis[1] will be marked out of 50. All Questions are Compulsory.
- Think of this Course Synthesis as a week-long individual take-home exam where you may consult your notes, the course textbook, anything on the Canvas Page, and any websites linked from the Canvas Page. However, you may NOT consult your classmates or look at any other online resources unless explicitly approved by me beforehand. Please post your questions on the Canvas Discussion Forum if you would like any clarifications or hints.
- To make it easier for the TAs, submit individual .pdf files for each of the problems and make sure your write the **Problem Number** on top of each page.

[1] Acknowledgement: These problem sets have been adapted from problem sets initially created and prepared by Prof Richard Hoshino, Northeastern University, Vancouver

(10 pts.) Problem #1 - Short Answer Questions

In this question, you only need to submit your final ANSWERS to these 6 questions. No additional work is required or requested. No justification is required – all I want is your final answer.

Submit a single .pdf file (ideally one page long) on which you will provide your answers to the 6 questions below.

(1) (1 pts.)Of the four options below, exactly one lists the various running times from fastest to slowest.

A. $O(1)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(2^n)$, $O(n!)$, $O(n^{1000})$
B. $O(1)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n!)$, $O(2^n)$, $O(n^{1000})$
C. $O(1)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^{1000})$, $O(n!)$, $O(2^n)$
D. $O(1)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^{1000})$, $O(2^n)$, $O(n!)$

Determine which list is correct. Answer either A or B or C or D.

(2) (1 pts.) $f(n) = 5n^3 + 10n^2 + 15n + 1000$. Consider the following statements:

i. $f(n) = O(n^1)$
ii. $f(n) = O(n^2)$
iii. $f(n) = O(n^3)$
iv. $f(n) = O(n^4)$

Determine how many of these four statements are TRUE.

(3) (2 pts.) Let $T(n)$ be defined by the recurrence relation $T(1) = 1$, and $T(n) = 32T(n/2) + n^k$ for all $n > 1$.

Determine the integer k for which $T(n) = \Theta(n^5 \log n)$.

(4) (2 pts.) Let A be an array of numbers. In the maximum sub-array problem, your goal is to determine the sub-array $A[x \ldots y]$ of consecutive terms for which the sum of the entries is as large as possible.

For example, if $A = [-2, -3, 4, -1, -2, 1, 5, -3]$, the maximum sub-array is $[4, -1, 2, 1, 5]$, and the largest possible sum is $S = 4 - 1 - 2 + 1 + 5 = 7$.

Suppose $A = [1, 2, -4, 8, 16, -32, 64, 128, -256, 512, 1024, -2048]$.

Determine S, the largest possible sum of a sub-array of A.

(5) (2 pts.)Suppose we want to use the Heapsort algorithm to sort a large list of numbers.

Our first step is to convert the input list to a heap, and then run BUILD-MAX-HEAP, which applies MAX-HEAPIFY on all the nodes in the heap, starting at the bottom and moving towards the top.

For example, if $A = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]$, then BUILD-MAX-HEAP(A) makes a total of 7 swaps, and returns the array $[16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$.

The swaps are made in this order: $(14, 2), (10, 3), (16, 1), (7, 1), (16, 4), (14, 4), (8, 4)$.

Suppose $A = [2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15]$.

Determine the total number of swaps made by BUILD-MAX-HEAP(A).

(6) (2 pts.) You have a knapsack that can hold 10 pounds, which you can fill with any of these items.

Object	A	B	C	D	E
Weight (in pounds)	1	2	3	4	5
Value (in dollars)	$5	$15	$24	$30	$35

In the Fractional Knapsack Problem, you are allowed to take f of each object, where f is some real number between 0 and 1. Your goal is to pick the objects that maximize the total value of your knapsack, with the condition that the chosen objects weigh at most 10 pounds.

Determine the maximum total value of your knapsack.

(10 pts.) Problem #2 - Guess My Word

This question is inspired by the online Guess My Word challenge, whose URL is:

https://hryanjones.com/guess-my-word/

Each time you enter a guess, the program will tell you whether the secret word is alphabetically *before* your guess, alphabetically *after* your guess, or *exactly* matches your guess.

Each secret word is randomly chosen from a dictionary with exactly $267,751$ words.

(a) Post a screenshot of you winning this game.

You receive full credit if you require at most 20 guesses *or* guess the word within 2 minutes. If you require more than 20 guesses *and* require more than 2 minutes, you will receive partial credit. (You can play this game as often as you'd like! Please submit a screenshot of your best result.)

(b) Suppose the secret word is randomly chosen from a dictionary with exactly $2^k - 1$ words, where k is a positive integer. Describe an algorithm that guarantees that you can identify the secret word in at most k guesses. Clearly justify how and why your algorithm works.

(c) Let $T(n)$ be the maximum number of guesses required to correctly identify a secret word that is randomly chosen from a dictionary with exactly n words.

Determine a recurrence relation for $T(n)$, explain why the recurrence relation is true, and then apply the Master Theorem to show that $T(n) = \Theta(\log n)$.

(d) Suppose I give you \$15 to play the online Guess My Word game. Every time you make a guess, you give me \$1. If you agree to play this game with me, do you expect to *win* money or *lose* money? Clearly justify your answer. (Assume that each of the $267,751$ words is equally likely to be chosen.)

(10 pts.) Problem #3 - Sorting Algorithms

Sorting algorithms are incredibly important and used by us on a daily basis. In this course we have investigated and analyzed numerous sorting algorithms.

In each of the following questions, you are given an input array A with n elements, where n is a very large positive integer. You will then compare two sorting algorithms and justify which of the two algorithms is faster in sorting this array A.

In your responses, you must clearly explain which of the two algorithms runs faster. Make sure you rigorously justify your answer, carefully proving whether each algorithm is $O(n)$, $O(n \log n)$, or $O(n^2)$.

(a) Let $A = [5, 5, 5, 5, 5, \ldots, 5]$ be an array where all of the elements are equal to 5.

Determine whether **Selection Sort** or **Insertion Sort** sorts this array faster.

(b) Let $A = [n, n-1, n-2, \ldots, 3, 2, 1]$ be an array where the first n positive integers are listed in decreasing order.

Determine whether **Heapsort** or **Quicksort** sorts this array faster.

For this question, assume the Quicksort pivot is always the right-most element.

(c) Let A be an array, where each of the n elements is a randomly chosen integer between 1 and n. For example, if $n = 12$, this array could be $A = [3, 5, 1, 10, 5, 7, 9, 12, 2, 8, 8, 6]$.

Determine whether **Bubble Sort** or **Bucket Sort** sorts this array faster.

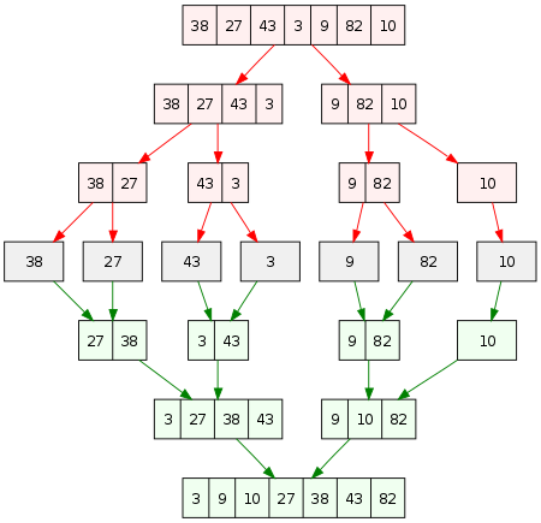
(d) Let A be an array, where each of the n elements is a randomly chosen digit between 0 and 9. For example, if $n = 12$, this array could be $A = [3, 5, 1, 0, 5, 7, 9, 2, 2, 8, 8, 6]$.

Determine whether **Counting Sort** or **Merge Sort** sorts this array faster.

(10 pts.) Problem #4 - Merge Sort Comparisons

Merge Sort is a powerful divide-and-conquer algorithm that recursively sorts an array by breaking it into two approximately-equal pieces, applying Merge Sort to each, and then merging the two sorted sub-arrays to produce the final sorted array.

This picture provides a visual illustration of Merge Sort, on the unsorted array [38,27,43,3,9,82,10].



When we merge two sorted sub-arrays, we only compare the left-most element of each sub-array, since one of these two elements is guaranteed to be the smallest. We then repeat the process until one of the two sub-arrays is empty. Then there is nothing left to compare, and we will have our desired merged array.

Let's count the number of *comparisons* needed to produce the combined (and sorted!) merged array.

To get the first green level, we require 3 comparisons (38-27, 43-3, 9-82).
To get the second green level, we require 5 comparisons (27-3, 27-43, 38-43, 9-10, 82-10)
To get the third and final green level, we require 6 comparisons (3-9, 27-9, 27-10, 27-82, 38-82, 43-82)

Thus, Merge Sort requires 3+5+6 = 14 total comparisons to sort the array [38,27,43,3,9,82,10].

(a) $A = [6, 5, 8, 7, 3, 4, 2, 1]$. Perform the Merge Sort algorithm on this array, using a visual illustration of each step to generate the sorted array $[1, 2, 3, 4, 5, 6, 7, 8]$. Show that exactly 12 comparisons are needed to sort this input array A.

(b) Let $M(n)$ be the *minimum* number of comparisons needed to sort an array A with exactly n elements. For example, $M(1) = 0$, $M(2) = 1$, and $M(4) = 4$. If n is an even number, clearly explain why $M(n) = 2M(n/2) + n/2$.

(c) If n is a power of 2, prove that $M(n) = \dfrac{n \log n}{2}$, using any method of your choice. Show all your steps.

(d) Let A be a random permutation of $[1, 2, 3, 4, 5, 6, 7, 8]$. Determine the probability that *exactly* 12 comparisons are required by Merge Sort to sort the input array A. Clearly and carefully justify your answer.

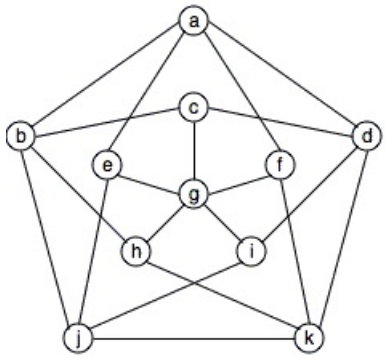
(10 pts.) Problem #5 - Graph Coloring

In this question you will explore Graph Colouring algorithms.

Given a graph G, we say that G is k-colourable if every vertex of G can be assigned one of k colours so that for every pair u, v of adjacent vertices, u and v are assigned different colours.

The *chromatic number* of a graph G, denoted by $\chi(G)$, is the smallest integer k for which graph G is k-colorable. To show that $\chi(G) = k$, you must show that the graph is k-colourable and that the graph is not $(k-1)$-colourable.

(a) Let G be the graph below, with 11 vertices and 20 edges. Clearly explain why $\chi(G) = 4$.



(b) Create a simple greedy algorithm for colouring the vertices of any graph G, ideally using as few colours as possible. Explain how your algorithm works, i.e., the order in which your algorithm chooses the vertices of a given graph, and how a colour is assigned to each vertex.

Apply your algorithm to the graph in (a). How many colours did your algorithm use?

(c) For any graph G, does your algorithm always use exactly $\chi(G)$ colours? If so, explain why. If not, provide a graph G for which your algorithm requires more than $\chi(G)$ colours.

(d) It is **NP**-complete to determine whether an arbitrary graph has chromatic number k, where $k \geq 3$. However, determining whether an arbitrary graph has chromatic number 2 is in **P**.

Given a graph G on n vertices, create an algorithm that will return TRUE if $\chi(G) = 2$ and FALSE if $\chi(G) \neq 2$. Clearly explain how your algorithm works, why it guarantees the correct output, and determine the running time of your algorithm.