Due Friday 16^{th} of June at 6pm Sydney time (week 3)

In this assignment we review some basic algorithms and data structures, and we apply the divide-and-conquer paradigm. There are *three problems* each worth 20 marks, for a total of 60 marks. Partial credit will be awarded for progress towards a solution. We'll award one mark for a response of "one sympathy mark please" for a whole question, but not for parts of a question.

Any requests for clarification of the assignment questions should be submitted using the Ed forum. We will maintain a FAQ thread for this assignment.

For each question requiring you to design an algorithm, you *must* justify the correctness of your algorithm. If a time bound is specified in the question, you also *must* argue that your algorithm meets this time bound. The required time bound always applies to the *worst case* unless otherwise specified.

You must submit your response to each question as a separate PDF document on Moodle. You can submit as many times as you like. Only the last submission will be marked.

Your solutions must be typed, *not* handwritten. We recommend that you use LaTeX, since:

- as a UNSW student, you have a free Professional account on Overleaf, and
- we will release a LaTeX template for each assignment question.

Other typesetting systems that support mathematical notation (such as Microsoft Word) are also acceptable.

Your assignment submissions must be your own work.

- You may make reference to published course material (e.g. lecture slides, tutorial solutions) without providing a formal citation. The same applies to material from COMP2521/9024.
- You may make reference to either of the recommended textbooks with a citation in any format.
- You may reproduce general material from external sources in your own words, along with a citation in any format. 'General' here excludes material directly concerning the assignment question. For example, you can use material which gives more detail on certain properties of a data structure, but you cannot use material which directly answers the particular question asked in the assignment.
- You may discuss the assignment problems privately with other students. If you do so, you must acknowledge the other students by name and zID in a citation.
- However, you must write your submissions entirely by yourself.
  - Do not share your written work with anyone except COMP3121/9101 staff, and do not store it in a publicly accessible repository.
  - The only exception here is UNSW Smarthinking, which is the university's official writing support service.

Please review the UNSW policy on plagiarism. Academic misconduct carries severe penalties.

Please read the Frequently Asked Questions document, which contains extensive information about these assignments, including:

- how to get help with assignment problems, and what level of help course staff can give you
- extensions, Special Consideration and late submissions
- an overview of our marking procedures and marking guidelines
- how to appeal your mark, should you wish to do so.

Question 1

You are given an array A of n distinct positive integers and a positive integer x.

1.1 **[8 marks]** Design an algorithm which decides if there exist two distinct indices $1 \leq i, j \leq n$ such that $2A[i] - 3A[j] = x$. In the worst-case, your algorithm must run in $O(n \log n)$ time.

1.2 **[4 marks]** Solve the same problem as in 1.1 but with an algorithm which runs in the **expected time** of $O(n)$.

1.3 **[8 marks]** Design an algorithm which counts how many distinct pairs of indices (i, j) where $1 \leq i < j \leq n$ satisfy both:

- $A[i] > A[j]$; and
- $A[i] + A[j] = x$.

In the worst-case, your algorithm must run in $O(n(\log n)^2)$ time.

Question 2

2.1 **[6 marks]** Blake and Red each have a collection of n distinct Pokemon cards. Blake creates an array B containing the serial numbers of their cards, and Red creates an array R with the serial numbers of his. Blake wishes to know how many of their cards are *not* also owned by Red (that is, the size of $B \setminus R$).

For example, if Blake has cards with serial numbers $B = [7, 2, 1, 5]$ and Red has cards numbered $R = [2, 8, 7, 3]$, then there are 2 cards owned by Blake but not Red.

Design an algorithm which runs in $O(n \log n)$ time and determines how many cards are owned by Blake but not Red.

2.2 **[4 marks]** Gerald has a class of k students, each of which has a collection of distinct Pokemon cards. One day, he asks each member of his class to bring their collection, sorted by serial number. The ith student has c_i cards, with their **sorted** serial numbers in an array $N_i[1 \ldots c_i]$. The total number of (not necessarily distinct) cards owned by the class is $S = \sum_{i=1}^{k} c_i$.

He wants to find all the distinct cards owned by the class (i.e. $\bigcup_{i=1}^{k} N_i$), but needs to do so in $O(S \log k)$ to finish the lesson in time. Gerald devises the following algorithm to do so:

> We create an array N^* to store the serial numbers of the unique cards owned by the class. We first put all the numbers from N_1 into N^*.
> Then, for each i from 2 to k, we merge N_i with N^*, only including one copy of duplicate cards. Since N_i and N^* are both sorted, we are guaranteed to see duplicate cards one after the other while merging. We then replace N^* with the merged array.
> The final array N^* contains the merged arrays $N_1, N_2, \ldots, N_k$ without duplicate serial numbers, i.e. all the distinct cards owned by the class.

This algorithm is correct, but unfortunately does not meet Gerald's $O(S \log k)$ time complexity requirement. Justify why the worst-case time complexity of Gerald's algorithm is slower than $O(S \log k)$.

Note that an example with a fixed size (i.e., fixed S or k) is **not** sufficient!

2.3 **[10 marks]** Design an algorithm that finds the distinct cards owned by Gerald's class, and runs in $O(S \log k)$ time.

Question 3

3.1 Read about the asymptotic notation in the review material and determine if $f(n) = O(g(n))$ or $g(n) = O(f(n))$ or both (i.e., $f(n) = \Theta(g(n))$) or neither of the two, for the following pairs of functions.

[A] **[5 marks]**

$$f(n) = \log_2(n); \quad g(n) = \sqrt[5]{n}$$

[B] **[5 marks]**

$$f(n) = n^n; \quad g(n) = 2^{n \log_2(n^2)}$$

[C] **[5 marks]**

$$f(n) = n^{1+\cos(\pi n)}; \quad g(n) = n$$

You might find L'Hôpital's rule useful: if $f(x), g(x) \to \infty$ as $x \to \infty$ and they are differentiable, then

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}.$$

3.2 **[5 marks]** Given n strings, each of length at most m, a divide and conquer approach can be used to find the longest common prefix amongst the strings. For example, the longest common prefix amongst `apple`, `apply` and `apart` is `ap`. Song has provided an algorithm to do so below:

1. Recursively determine the longest common prefixes among the first $n/2$ words and the last $n/2$ words.
2. Merge the results to determine the longest common prefix between the two halves by scanning through character by character.
3. Base case: for $n = 1$, the longest common prefix is the entire string.

However, Song isn't sure about the time complexity of the above algorithm. In Big-Theta notation, explain and justify what the time complexity of the above algorithm is.