Due Friday 29th of September 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 *Asymptotics*

Read about 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. Justify your answers.

[A] [**5 marks**]

$$f(n) = \sqrt[6]{n^2 - n}; \quad g(n) = \log_2\left(n^5\right)$$

[B] [**5 marks**]

$$f(n) = n; \quad g(n) = (\sin(\pi n) + \cos(\pi n))n$$

[C] [**5 marks**]

$$f(n) = 4^{n^2 \log_2(n)}; \quad g(n) = (n!)^n$$

[D] [**5 marks**]

$$f(n) = n^{2-\cos(\pi n)}; \quad g(n) = n\sqrt{n}$$

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

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

Remember that when dealing with asymptotics in computer science, we only care about integer values of n (input sizes are integers).

Question 2 *Convenience Store*

Serge owns and runs a convenience store where he purchases items from m different manufacturers and resells them to his customers. Each manufacturer sells items up to n at a time. When buying in bulk from a manufacturer, Serge receives a discount. To find out the cost per item when purchasing i items at a time from manufacturer k, Serge can ask for a quote from the manufacturer, denoted $Q(i, k)$. The manufacturers will give each quote in constant time, but if Serge asks for more than $\lceil \log_2 n \rceil$ quotes from the same manufacturer, he will be blocked for spam and shunned from the business community forever. It is never more expensive to purchase more items from a single manufacturer (that is, $Q(i, k) \geq Q(j, k)$ for all manufacturers k and all quantities of items $i < j$).

Serge also has an array $S[1..m]$, where $S[k]$ is the price he sells the items from manufacturer k for in his shop.

Serge can sell items for a profit whenever the cost he pays per item is cheaper than the price he sells them for in his shop. Serge is interested in determining the smallest number of items he needs to buy from each manufacturer in order to sell the items for a profit. For each manufacturer $1 \leq k \leq m$, we denote this quantity $M[k]$. That is, $M[k]$ is the smallest number of items that Serge must buy from manufacturer k in order to sell the items for a profit. For all manufacturers, $Q(n, k) < S[k]$, so it is always possible for Serge to make a profit.

2.1 **[6 marks]** Serge is interested in purchasing from a particular manufacturer k. Design an algorithm that determines the smallest number of items Serge needs to purchase from manufacturer k in order to sell them for a profit, without asking for more than $\lceil \log_2 n \rceil$ quotes.

Example: Suppose manufacturer k sells 1 item for \$2, 2 or 3 items for \$1.50 each, and 4 items for \$1.25 each. That is, $Q(1, k) = \$2, Q(2, k) = Q(3, k) = \$1.50$ and $Q(4, k) = \$1.25$. Serge sells items from manufacturer k for \$1.75 each ($S[k] = \$1.75$). The smallest number of items Serge needs to sell to make a profit is 2, so $M[k] = 2$.

$Q(i, k)$ gives the cost per item when i items are purchased from manufacturer k, and $S[k]$ is the price Serge can sell the items for.

2.2 **[14 marks]** Each of Serge's m manufacturers does not produce items of the same quality. In fact, Serge has numbered them in increasing order of quality, so manufacturer 1 produces the lowest quality items and manufacturer m produces the highest quality items. Serge knows for a fact that when purchasing items from a higher quality manufacturer, he needs to purchase the same or fewer items to sell them for a profit. That is, if $k > l$, then $M[k] \leq M[l]$.

Serge would like to compute $M[k]$ for all m manufacturers. Serge has realised that it is possible to use your method from Q2.1 on each of the m manufacturers for a total time complexity of $O(m \log n)$, but has decided that this is not fast enough.

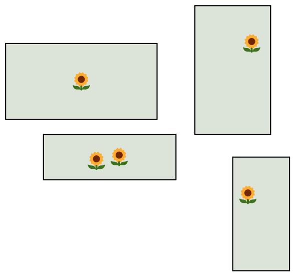
Assuming $m < n$, design an algorithm that runs in $O(n)$ time to determine the smallest number of items Serge needs to buy from each manufacturer in order to sell the items for a profit while asking each manufacturer for at most $\lceil \log_2 n \rceil$ quotes.

Your algorithm should use the fact that $M[k] \leq M[l]$ for all manufacturers $k > l$.

Question 3 *In the Night Garden*

Alice is planting n_1 flowers $f_1, \ldots, f_{n_1}$ among n_2 rectangular gardens $\mathcal{G}_1, \ldots, \mathcal{G}_{n_2}$. Bob's task is to determine which flowers belong to which gardens. Alice informs Bob that no two gardens overlap; therefore, if a flower belongs to a garden, then the flower belongs to *exactly* one garden and a garden can contain multiple flowers. If a flower f_i does not belong to any garden, then Bob returns an *undefined* garden for f_i.

Each garden $\mathcal{G}_i$ is given by a pair of points $\mathcal{G}_{BL}[i]$ and $\mathcal{G}_{TR}[i]$, representing the bottom left and top right corners of the garden respectively. Each flower is represented by a point $F[i]$ representing its location. Let $n = n_1 + n_2$.



A collection of $n_1 = 5$ flowers and $n_2 = 4$ gardens.

More formally, you are given three arrays:

- $\mathcal{G}_{BL} = [(x_1, y_1), \ldots, (x_{n_2}, y_{n_2})]$, where $\mathcal{G}_{BL}[i] = (x_i, y_i)$ represents the bottom left point of garden $\mathcal{G}_i$,
- $\mathcal{G}_{TR} = [(x_1, y_1), \ldots, (x_{n_2}, y_{n_2})]$, where $\mathcal{G}_{TR}[i] = (x_i, y_i)$ represents the top right point of garden $\mathcal{G}_i$, and
- $F = [(x_1, y_1), \ldots, (x_{n_1}, y_{n_1})]$, where $F[i] = (x_i, y_i)$ represents the location of flower f_i.

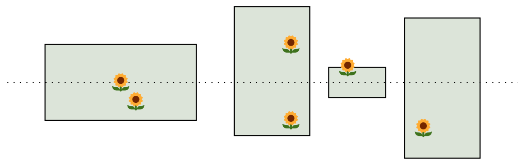
You are not guaranteed that a flower belongs to a garden; it is possible that a flower is planted in none of the gardens. Your goal is to return the garden that a flower f_i belongs to (if any), for *each* f_i.

- For example, in the diagram above, if the flower ordering is sorted by its x-coordinate and the rectangular gardens are sorted by their x-coordinate of $\mathcal{G}_{BL}$, then we return

$$f_1 : \mathcal{G}_1, \quad f_2 : \mathcal{G}_2, \quad f_3 : \mathcal{G}_2, \quad f_4 : \mathcal{G}_4, \quad f_5 : \mathcal{G}_3.$$

  Your output can be in any form (e.g. array, list, set, dictionary), so long as you maintain the correct pair of flowers and gardens.

3.1 [12 marks] We first solve the special case where all of the gardens intersect with a horizontal line. Design an $O(n \log n)$ algorithm to determine which flowers belong to which gardens (if such a garden exists).



A collection of $n_1 = 6$ flowers and $n_2 = 4$ gardens that intersect with a horizontal line.

Hint. What do you know about two adjacent gardens if they have to intersect with a horizontal line?

3.2 [**8 marks**] We now remove the assumption that every garden intersects with a horizontal line. Design an $O(n(\log n)^2)$ algorithm to determine which flowers belong to which gardens (if such a garden exists).

Hint. Divide and conquer, and use the previous part as a subroutine.