National University of Singapore

School of Computing

CS3230 - Design and Analysis of Algorithms

Final Assessment

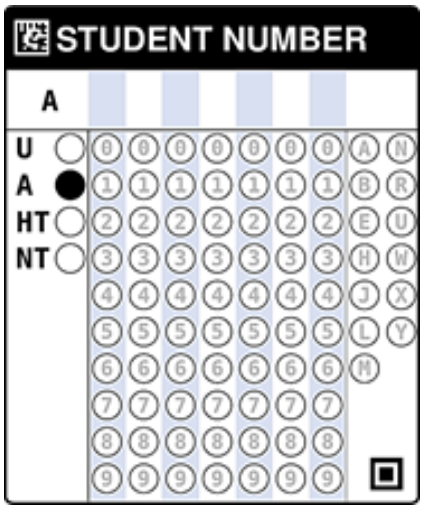
(Semester 2 AY2023/24)

Total Marks: 80 Time Allowed: 120 minutes

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains TWO (2) sections. It comprises THIRTEEN (13) printed pages.
3. This is an **Open Book** Assessment.
4. For Section A, use the bubbles on page 2 (use 2B pencil).
5. For Section B, answer **ALL** questions within the **boxed space**.
   If you leave the box blank, you will get 0.5 mark (**ONLY** for essay questions worth ≥ 2).
   However, if you write at least a single character and it is totally wrong, you will get a 0 mark.
   You can use either a pen or a pencil. Just make sure that you write **legibly**!
6. Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
   Read all the questions first! Some questions might be easier than they appear.
7. You can assume that all **logarithms are in base** 2.

Write your Student Number in the box below using **(2B) pencil**:



A Multiple Choice Questions ($16 \times 2.5 = 40$ marks)

Select the **best unique** answer for each question. Each correct answer is worth 2.5 marks.
Write your MCQ answers in the special MCQ answer box below for automatic grading.
We do not manually check your answer.

Write your MCQ answers in the answer box below using (2B) pencil:

No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
B	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
D	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
E	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

The default answers (as the MCQ section is not supposed to be archived to open up possibilities of reuse in the future): eceb aaab cadd ebda.

Page 3 is redacted.

Page 4 is redacted.

Page 5 is redacted.

Page 6 is redacted.

B Essay Questions (40 marks)

B.1 Average-case Comparisons ($2+3+5=10$ marks)

Consider the following algorithm to find the two largest numbers from an n-length array $A[1..n]$ of n distinct numbers.

Input: Given n distinct numbers, $A[1..n]$ contains a random permutation of them.
Begin

1. If $A[1] > A[2]$,
   Then let $a = A[1]$ and $b = A[2]$,
   Else let $a = A[2]$ and $b = A[1]$

Endif

2. For $i = 3$ to n do:

   2.1. If $b < A[i]$ Then

   2.2. If $a < A[i]$,
   Then let $b = a$ and $a = A[i]$
   Else let $b = A[i]$

   Endif

   Endif

//Comment: a and b store the two largest elements of the subarray $A[1..i]$.
End For

3. Output a, b.

End

a). (2 marks) For a fixed i, in the For loop iteration with index i, at step 2.1: what is the probability that $A[i] > b$?

(Your answer can be in terms of i. **Hint:** Observe the comment in the pseudocode carefully.).

$2/i$
Marking scheme: Blank 0.5 for each of the three parts for B1

B1a: 2 marks for 2/i (or 2/(i+1), 2/(i-1)).
1 mark for 1/i or similar
In some cases 1 mark where had some errors, but reasonable logic (for example a complex formula without simplification was given, but was not completely correct)
Most other get 0 marks

b). (3 marks) What is the expected number of comparisons made by the above algorithm? (Here, we count only the comparisons made between a, b and the array elements in steps numbered 1, 2.1, and 2.2 and not any comparisons done in the control statement of the For statement).

Give the bound in the form $n + O(f(n))$, where $f(n)$ is as tight as possible.

$n + O(\log n)$

B1b: n + O(log n), given 3 marks
n + O(n), given 1 mark (this is a trivial bound, as one can have at most 2 comparisons in each iteration of the loop)
In some cases, partial correctness gets 1 mark (especially if they have reasonable argument in B1c)
Most other get 0 marks

c). (5 marks) Provide justification for your answers to parts (a) and (b).

(a) The probability that $A[i] > b$ is the same as the probability that it is among the two largest elements in $A[1 : i]$. This probability is $2/i$.

(b) The algorithm makes one comparison in step 1, and one comparison between $A[i]$ and b in the If statement of step 2.1 of each iteration of the for loop. This gives $n - 1$ comparisons.

In each iteration of the For loop, the 2nd If statement (step 2.2) makes one comparison between a and $A[i]$ only if $b < A[i]$ is true in the first If statement. This happens with probability $2/i$ in the iteration of the For loop with index i. Thus, the expected number of comparisons in step 2.2 is:

$\sum_{i=3}^{n} 2/i = 2H_n - 3 = \Theta(\log n)$.

Hence, the total number of comparisons is $n + \Theta(\log n)$.

B1c: 2 marks for the correct proof for part (a), and 3 marks for correct proof of part (b)
In some cases 1 marks for (a) or (b) or mix, for reasonable proof idea.
In some cases part (b) gets 2 marks for almost there proof.
Though note that the above credit is only if you are going towards the correct answer

B.2 Maximizing Profit ($1 + 3 + 4 + 2 = 10$ marks)

You are a fisherman. Very early this morning, you caught n fishes. These n fishes weigh $w_1, w_2, ..., w_n$ kilograms respectively (you can assume that all n fish weights are Integers between 1 to 1024 kilograms, not necessarily distinct).

In the fish market, there are m fish sellers described as a sequence of m pairs $(x_1, p_1), (x_2, p_2), ..., (x_m, p_m)$. A fish seller j with (x_j, p_j) pair means that this fish seller j is willing to buy x_j number of fish (x_j is also an Integer, $1 \leq x_j \leq n$ (notice x_j could be as large as n), and not necessarily distinct) at p_j SGD per kilogram (p_j is also an Integer, $1 \leq p_j \leq 1024$, and not necessarily distinct).

Design a greedy algorithm to compute the maximum profit (in SGD) that you can get by selling (some, if not all) your n fishes to (some of) these m fish sellers optimally. Note that there are partial marks if your greedy algorithm is correct only when $x_j = 1$ for all m fish sellers (e.g., see part a).1)).

For example, if you caught $n = 3$ fishes with weights $4, 7, 5$ and there are $m = 2$ fish sellers $(1, 10), (3, 9)$, then the optimal strategy is to sell your second fish with weight 7 kilogram to the first fish seller who only wants to buy 1 fish that day at price 10 SGD per kilogram (you get $7 \times 10 = 70$

SGD) and then sell your two other fishes to the second fish seller who can buy up to 3 fishes but you only have two fishes left (you get $4 \times 9 + 5 \times 9 = 81$ SGD). So your total profit is $70 + 81 = 151$ SGD.

a). ($2 \times 0.5 = 1$ mark) **Judge your understanding:**
Just write two output Integers, one for each test case below.
1). $n = 3$, weights $9, 4, 5$, $m = 4$, fish sellers $(1, 2), (1, 1), (1, 6), (1, 3)$
2). $n = 10$, weights $10, 8, 4, 28, 19, 2, 7, 5, 9, 1$, $m = 3$, fish sellers $(2, 4), (1, 5), (4, 3)$

a). $9 \times 6 + 5 \times 3 + 4 \times 2 = 54 + 15 + 8 = 77$ SGD.
b). $28 \times 5 + (19 + 10) \times 4 + (9 + 8 + 7 + 5) \times 3 = 140 + 116 + 87 = 343$ SGD.
Marking scheme: 0 (blank or both wrong); 0.5 (one correct; no other partial); 1 (both correct).

b). (3 marks) Describe the **optimal sub-structure** of this problem and **prove** its correctness.

Suppose we sell 1 of our fish (let's say fish i) to fish seller j. The sub-problem is how to optimally sell our $n - 1$ other fish (fish i is no longer considered), and the fish seller j state becomes $(x_j - 1, p_j)$ (if x_j drops to 0, we remove fisherman j from consideration).

We can use a cut-and-paste argument. Consider any optimal solution. If fish i with weight w_i is sold to fish seller j with state (x_j, p_j), then we obtain $w_i \times p_j$ SGD, fish i is no longer considered, and fish seller j state becomes $(x_j - 1, p_j)$ — and if $x_j - 1 = 0$, we also remove fisherman j from consideration. Now, we claim that the remaining sub-problem must be optimal, too. Because if it does not and it gives a higher profit, then we can add $w_i \times p_j$ to that profit (as we can sell fish i to fish seller j who is willing to buy 1 more fish at a price p_j) to get an even higher profit. Contradiction! So, this establishes the optimal sub-structure of this problem.

Marking scheme: 0 (very wrong); 0.5 (blank); 1 (no proof); 2 (the proof already involves greedy choice, i.e., saying removing heaviest fish with most generous fish seller at this answer box, notice that optimal sub-structure deals with the components of an optimal solution (don't forget to mention this otherwise your proof has an issue) and we can remove any fish i (not necessarily the heaviest) that was paired with fish seller j (not necessarily the most generous) in the proof); 3 (correct).

c). (4 marks) Describe the **greedy choice** that works for this problem and **prove** its correctness.
PS: If your greedy choice is correct but your proof is not, you will still get partial marks.

We sort the n fish in non-increasing weight. We sort the m fish seller in non-increasing price per kilogram. We greedily sell our next heaviest fish to the still available fish seller (that fish seller still wants to buy 1 or more fish) that is willing to pay the highest price per kilogram (the most generous).

Let fish i be such that it's weight $w(i)$ is the maximum and fish seller j be such that $p(j)$ is the maximum. We claim that there exists an optimal solution that sells fish i to fish seller j.

There are a few cases involving exchange arguments (between OPT and greedy strategy) to consider. Most students miss one-or-two cases but we mark this leniently.

- Case 0 (not actually crucial): If fish i is sold to fish seller j, we are done.

- Case 1: We sell at least one fish $i' \neq i$ to fish seller j, and fish i is sold to some other fish seller j'. By assumption, $w(i') \leq w(i)$ and $p(j') \leq p(j)$. Therefore, $w(i') \times p(j) + w(i) \times p(j') \leq w(i) \times p(j) + w(i') \times p(j')$ and we are not worse off if we sell i' to j' and i to j, so there exists an optimal solution which sells fish i to seller j.
- Case 2: We do not sell any fish to fish seller j, and fish i is sold to some other fish seller j'. In this case, we can sell fish i to fish seller j instead of fish seller j'. By assumption, $p(j') \leq p(j)$. Therefore, $w(i) \times p(j') \leq w(i) \times p(j)$, so there exists an optimal solution which sells fish i to fish seller j.
- Case 3: We sell at least one fish $i' \neq j$ to fish seller j, but fish i is not sold. By assumption, $w(i') \leq w(i)$. We can instead sell fish i to fish seller j and choose not to sell fish i' to fish seller j.
- Impossible case 4 (not actually crucial): We do not sell any fish to fish seller j (with $x_j \geq 1$), and fish i is not sold. This is not possible as by selling fish i to fish seller j, get can easily get an extra $w(i) \times p(j)$ more profit.

In all cases, we make at least as much by selling fish i to fish seller j. Establishing our greedy choice correctness.

d). (2 marks) Combine the **optimal sub-structure** (in B.2.b) and the **greedy choice** (in B.2.c) to design an algorithm that always outputs an optimal solution. Analyze the time complexity of your greedy solution in terms of n and m. Is it polynomial, pseudo-polynomial, or exponential (choose the best option)?

The input size is $n \log 1024$ bits for the n fish information, plus $m \log(n+1024)$ bits for fish seller information, or $n + m \log n$.

First, sort the n fish (by non-increasing weights) and m fish sellers (by non-increasing price per kilogram). These two two sorting routines cost us $O(n \log n + m \log m)$.

Then, we do one $O(n)$ greedy bipartite matching step that always matches the current heaviest fish (we remove this fish afterward) with the most generous fish seller (we reduce his/her quota by one, and if this drops to 0, we remove this seller). We do this only $O(n)$ time *overall* even if each individual fish seller can buy up to n fishes.

So, this algorithm is $O(n \log n + m \log m)$ overall. This is a polynomial time algorithm.

B.3 Priority Queue (10 marks)

Recall that the priority queue data structure supports the following operations:

- `add(x)`: inserts x into the queue; $O(\log n)$ (worst-case) time complexity (n refers to the number of elements in the queue)
- `top()`: returns the largest element in the queue; $O(1)$ (worst-case) time complexity
- `pop()`: removes the largest element from the queue; $O(\log n)$ (worst-case) time complexity (n refers to the number of elements in the queue)

We now wish to support another operation with the following specifications:

- `remove_larger_than(x)`: removes all items larger than x from the queue

`remove_larger_than(x)` works by repeatedly checking whether `top()` is larger than x, and if so, calls `pop()` until `top()` is $\leq x$. More precisely, you may assume the following implementation:

```
remove_larger_than(x):
    while (priority queue is not empty and top() > x):
        pop()
```

Given an initially empty priority queue, prove that any sequence of n (of the above four types of) operations takes at most $O(n \log n)$ time. You can use any of the three amortized analysis techniques.

Grading remarks:

- 2 points awarded for identifying the cost of `remove_larger_than(x)`, and correctly identifying that `remove_larger_than` calls `pop()` k times and `top()` $k+1$ times where k is the number of items removed.
- Potential function = size of queue does not work. Similarly, for accounting method, charging 1 for each add does not work. Those who do this, max 5 points.
- Some say that "in the worst case, the entire queue is cleared during `remove_larger_than(x)`" — this is not obvious. Max 7/10 for making this assumption.

Correct solution (potential method):

- Define a potential function = $(\log n + 1) \times$ size of queue. check: initially, potential = 0, potential is non-negative.
- add: change in potential function = $\log n + 1$, true cost = $\log n$, amortized cost = $2logn + 1$.
- pop: change in potential function = $-(\log n + 1)$, true cost = $\log n$, amortized cost = -1.
- top: change in potential function = 0, true cost = 1, amortized cost = 0.

- remove_larger_than: let k be the number of items removed. then change in potential function $= k \times (\log n + 1)$. true cost $= k \times logn + (k + 1)$, this is due to k invocations of pop and $k + 1$ invocation of top, amortized cost $= 1$.

So all operations run in amortized $O(\log n)$ time, so that n operations run in $O(nlogn)$ time.

Note: for remove_larger_than, many students forget about the cost of top, and simply only count the true cost of $k \times logn$, and using potential function size of $queue \times (\log n)$. I did not penalize this.

Correct solution (accounting method):

- Deposit $\log n+1$ to bank balance for every `add()`, so that the amortized cost of add is $2\times\log n+1$. Do not charge anything for `top()`.
- For `pop()`, we can use the $\log n + 1$ bank balance to pay for the cost of `pop()`, so that `pop()` is free.
- For `remove_larger_than`, let k be the number of items removed. True cost $= k \times logn + (k+1)$. We can now use $k \times (\log n + 1)$ bank balance to pay for it.

Bank balance does not go negative - this is true as each element in the queue holds $(\log n + 1)$ charge. So all operations run in amortized $O(\log n)$ time, so that n operations run in $O(nlogn)$ time.

Note: Again, for remove_larger_than, many students forget about the cost of top, and simply only count the true cost of $k \times logn$, charging only $\log n$ for each add. I did not penalize this.

B.4 Respectful Coloring is NP-complete! $(1 + 3 + 6 = 10$ marks)

Suppose there are k persons $P_1, P_2, \cdots, P_k$ and n balls $B_1, B_2, \cdots, B_n$. Each person provides his/her preferred coloring of n balls from the set of five colors $\{red, blue, green, yellow, pink\}$. So essentially, each person P_i provides a sequence of colors $c_{i1}, c_{i2}, \cdots, c_{in}$, where $c_{ij} \in \{red, blue, green, yellow, pink\}$ denotes P_i's preferred color for the ball B_j.

Now, there is a painter whose job is to finally color all these n balls. Unfortunately, in his/her color palette, only two colors, *red* and *blue*, are available. The painter would like to color balls using colors available in his/her palette while respecting the color preference of every person for at least one ball. More specifically, we call a $red - blue$ coloring of balls a *respectful coloring* if for each person P_i there exists a ball B_j such that the coloring of B_j is the same as P_i's preferred color for B_j (i.e., c_{ij}). If such a respectful final coloring exists, the painter would like to use that (if multiple red-blue respectful colorings exist, choose one arbitrarily) to color the balls.

Red-Blue Respectful Coloring problem: Given color-preferences of k persons on n balls (as described in the first paragraph), the problem is to decide whether there exists a $red - blue$ respectful coloring or not.

For example, suppose there are 3 persons and 4 balls. The color preferences of three persons are as follows: $P_1 : green, blue, blue, yellow$, $P_2 : red, red, pink, green$, $P_3 : pink, yellow, blue, blue$. Then the answer should be YES since $red, blue, blue, blue$ is a valid $red - blue$ respectful coloring.

a). (1 mark) **Judge your understanding:** There are 4 persons and 3 balls. The color preferences of four persons are as follows: $P_1 : red, blue, yellow$, $P_2 : pink, red, green$, $P_3 : yellow, pink, blue$, $P_4 : blue, pink, red$. Does there exist a $red - blue$ respectful coloring? (Tick one of the following options; if your answer is YES, then also provide a $red - blue$ respectful coloring as a sequence of three colors.)

i). YES. Your $red - blue$ respectful coloring: _____

ii). NO. Answer.

Grading Remarks for B4a:
YES (wrong answer, so 0 mark), NO (correct one, so 1 mark)

b). (3 marks) Show that the Red-Blue Respectful Coloring problem is in NP.

The certificate would be an n-length sequence $f_1, f_2, \cdots, f_n$ where each $f_j \in \{red, blue\}$. A certifier algorithm should verify whether the above sequence is a valid $red - blue$ respectful coloring or not, which can easily be done by making a pass over color-preferences of each person. Clearly, the certificate is polynomial in size, and the verification can be done in polynomial time.

Grading Remarks for B4b:
For providing the correct (or almost correct) polysize certificate +1.5 marks
For providing justification on how to verify in polytime +1.5 marks
For leaving blank 0.5 marks

c). (6 marks) Show that the Red-Blue Respectful Coloring problem is NP-hard.

(You may show a reduction from any of the NP-complete problems introduced in the lectures/ tutorials/ assignments/ practice set.)

Hint: Try a reduction from CNF-SAT or 3-SAT.

Consider a 3-SAT instance with n variables $x_1, \cdots, x_n$ and k clauses $C_1, \cdots, C_k$. Now, create an instance of the Red-Blue Respectful Coloring problem as follows: For each variable x_j, consider a ball B_j, and for each clause C_i, consider a person P_i. The create color preference list for a person P_i as: If x_j appears in C_i as a positive literal, then set $c_{ij} = blue$, else if x_j appears in C_i as a negative literal then set $c_{ij} = red$, otherwise, set c_{ij} to be an arbitrary color from $\{green, yellow, pink\}$.

Clearly, the above reduction takes polynomial time. Next, we argue that the 3-SAT instance is satisfiable if and only if the reduced Red-Blue Respectful Coloring problem instance is a YES instance.

To show **3-SAT instance is satisfiable implies the reduced Red-Blue Respectful Coloring problem instance is a YES instance**, consider a satisfying assignment σ and build a red-blue coloring by setting $f_j = blue$ if $x_j = 1$ in σ; otherwise, $f_j = red$. Since σ satisfies each clause C_i, by the construction, for each P_i there exists a ball B_j such that $f_j = c_{ij}$.

For the **converse**, consider a valid $red - blue$ respectful coloring $f_1, \cdots, f_n$. Then create an assignment α by setting $\alpha(x_j) = 1$ if $f_j = blue$; $\alpha(x_j) = 0$ otherwise. Again, by an argument similar to that in the last paragraph, α is a satisfying assignment.

Grading Remarks for B4c:
For providing correct/almost correct transformation function from 3-SAT to respectful coloring +2 marks
For providing a clear proof on "if and only if" part of the reduction +4 marks
For providing some justification (but not a complete one, e.g. only one side) on "if and only if" part of the reduction +2 marks
For leaving blank 0.5 marks
Many students provided (tried to argue) a reduction from respectful coloring to 3-SAT, which, even if correct, is useless for this question (because it does not show whether respectful coloring is NP-hard or not), and thus get 0

– END OF PAPER; All the Best –