National University of Singapore

School of Computing

CS3230 - Design and Analysis of Algorithms

Midterm Test

(Semester 1 AY2024/25)

Time Allowed: 100 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 TWELVE (12) printed pages, including this page.
3. This is an **Open Book** Assessment.
4. For Section A, use the boxes at page 9 (use 2B pencil).
   You will still need to hand over the entire paper as the MCQ section will not be archived.
   For Section B, answer **ALL** questions within the **boxed space**.
   If you leave the boxed space blank, you will get automatic free ~~10% of the mark allocated~~.
   Unfortunately for us and fortunately for students, SoftMark smallest score is 0.5 marks.
   However, if you write at least a single character and it is totally wrong, you will get 0 mark.
   You can use either pen or pencil. Just make sure that you write **legibly**!
5. 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.
6. You can assume that all **logarithms are in base** 2.
7. The total marks of this paper is 40 marks.
   It will then be scaled to 20% of the course weightage.

A MCQs ($15 \times 1 = 15$ marks)

1. Which of the following statements is **TRUE**?

   a). $O(n^{1/\log\log n}) \subseteq \omega(\sqrt{n})$ Students either pick a)., c)., or e).

   b). $O(n^{1/\log\log n}) \subseteq \Theta(\sqrt{n})$ Unlikely Θ

   c). $\Omega(n^{1/\log\log n}) \subseteq \omega(\sqrt{n})$ Students either pick a)., c)., or e).

   d). $\Omega(n^{1/\log\log n}) \subseteq \Theta(\sqrt{n})$ Unlikely Θ

   e). None of the above Answer; rated 30-70 MCQ.

2. Which of the following statements is **TRUE**?

   a). If $f(n) \in O(g(n))$, then $2^{f(n)} \in O(2^{g(n)})$ No, e.g., $f(n) = 2n$, $g(n) = n$, $f(n) \in O(g(n))$, but $2^{2n} = 4^n \in \omega(2^n)$

   b). If $f(n) \in O(g(n))$, then $2^{g(n)} \in O(2^{f(n)})$ No, e.g., $f(n) = n$, $g(n) = 2n$, $f(n) \in O(g(n))$, but $2^{2n} = 4^n \in \omega(2^n)$

   c). If $f(n) \in O(g(n))$, then $g(n)^{f(n)} \in O(f(n)^{g(n)})$ No, e.g., $f(n) = 2n$, $g(n) = n$, $f(n) \in O(g(n))$, but $n^{2n} \in \omega(2n^n)$

   d). If $f(n) \in O(g(n))$, then $f(n)^{g(n)} \in O(g(n)^{f(n)})$ No, e.g., $f(n) = n$, $g(n) = 2n$, $f(n) \in O(g(n))$, but $n^{2n} \in \omega(2n^n)$

   e). None of the above Answer, but only after eliminating all 4 choices (2 similar pairs), rated 40-60 MCQ, will consume lots of thinking time.

3. Consider the recurrence relation: $T(n) = 1.999 \cdot T(n/2) + n$.
   Which of the following statements is **TRUE**?

   a) $T(n) \in o(n)$

   b) $T(n) \in \Theta(n)$ Answer, MT case 3, $\log_2 1.999 = 0.999$, $f(n) = n \in \Omega(n^{0.999+\epsilon})$ for some constant $\epsilon > 0$, regularity condition also met, easy MCQ, rated 90-10 MCQ.

   c) $T(n) \in \Theta(n \log n)$

   d) $T(n) \in \omega(n \log n)$

   e) None of the above

4. Consider the recurrence relation: $T(n) = a \cdot T(n/b) + n^d$ where $b > 1$ and $d \geq 0$.
   Which of the following statements is **TRUE**?

   a). If $a = 1$, we will always fall into case 3 of master theorem. No, e.g., $a = 1, b = 2, d = 0$ (binary search), we will fall into case 2

   b). If $a = 1$, we will never fall into case 1 of master theorem. Answer; as $\log_b a = 0$ on any base b as $a = 1$, so $\log_b a$ can never be smaller than $d \geq 0$ in this scenario. It easy to break a)., c)., and d). So hopefully 80-20 rated MCQ.

c). If $a > b$, we will always fall into case 1 of master theorem. No, e.g., $a = 4, b = 2, d = 2$, we will fall into case 2 as $\log_2 4 = 2$ is the same as $d = 2$.

d). If $a > b$, we will never fall into case 3 of master theorem. No, e.g., $a = 4, b = 2, d = 3$

e). None of the above

5. Which of the algorithms that have been discussed in class is **not** a Divide and Conquer algorithm?

a). Strassen's It is DnC, case 1 MT, $\Theta(n^{2.807})$, lecture 3b

b). Karatsuba's It is DnC, case 1 MT, $\Theta(n^{1.58})$, tutorial 04

c). Insertion Sort This is the obvious odd one out, hopefully now 90-10 MCQ

d). Stooge Sort It is actually is, case 1 MT, $\Theta(n^{2.7095})$, tutorial 03, although we will just use Merge or Quick sort instead of this lousier than $O(n^2)$ sorting algorithm, some students may think this is not DnC as it is 'slower' than the naive $O(n^2)$ sorting algorithm

e). None of the above Insertion Sort is the odd one out

6. Let $gcd(a, b)$ denote the greatest common divisor between two non-negative integers a and b. Consider the following recursive algorithm to compute $gcd(a, b)$.

- If $b = 0$, the algorithm returns a (indeed, in this case, $gcd(a, b) = a$).
- Otherwise, the algorithm returns $gcd(b, a\%b)$, which is computed recursively (indeed, in this case, $gcd(a, b) = gcd(b, a\%b)$).

Which of the following statements is **FALSE**?

a). For all a and b with $n = a + b$, the recursive algorithm to compute $gcd(a, b)$ runs in $o(\log n)$ in the worst case. $gcd(a, b)$ terminates quickly indeed for many random test-cases of a and b, but it can be made to run in $\Omega(\log n)$, option e). I rate this as 60-40 MCQ as some students probably tried and/or have access to last semester's assignments.

b). $gcd(553, 511) = 7$ just an annoying calculator work, yes both are divisible by 7 (and we are left with 79 and 73, which are primes).

c). $gcd(10000, 17) = 1$ $b = 17$ is prime, the answer is always 1.

d). $gcd(0, b) = b$ $gcd(0, b) = gcd(b, 0\%b) = gcd(b, 0) = b$

e). There exist a and b with $n = a + b$ such that the recursive algorithm to compute $gcd(a, b)$ runs in $\Omega(\log n)$ time in the worst case. yes, if a and b are two consecutive Fibonacci numbers, see `https://visualgo.net/en/recursion?slide=3-4`

7. Most sorting library routines in various programming languages use comparison-based sorting algorithms. Recall that in comparison-based algorithm, elements can only be compared with each other using $<, \leq, =, \geq$, or $>$ operators. Curiously, C++ `std::sort` only needs its users to provide `bool cmp(const Type1& a, const Type2& b);`, a comparison function object which returns `true` if the first argument `a` is less than (i.e., is ordered before) the second argument `b`, i.e., only the $<$ operator needs to be defined. Which of the following statements is **TRUE**?

a). C++ `std::sort` can only be used to sort comparable elements that has **no duplicates** (i.e., no two elements a and b where $a = b$) No, it can sort duplicates without problem.

b). C++ `std::sort` can only be used to sort elements only in **increasing order** No, it can sort in decreasing order.

c). C++ `std::sort` can only be used to sort comparable elements that has **no duplicates** and only in **increasing order** No, it can sort in decreasing order.

d). There is an $O(1)$-time operation to deduce whether $a = b$, if we can test strict inequality (i.e., check $x < y$ for any given x and y) in $O(1)$ time This is the one, if $a < b$ is false and $b < a$ is false, then the conclusion is $a = b$. This is indeed two checks, but is still $O(1)$. By method of elimination, I hope most students can reach this even if they are not sure with e)., Rated 70-30 MCQ.

e). None of the above Post grading remarks: Unfortunately option d). has a technical bug on the original wording. So we accept both d). and e). for this paper.

8. Consider a variant of the deterministic quicksort algorithm where the pivot selection is done using a function f that maps each positive integer k to an index in $\{1, 2, \ldots, k\}$: If the size of the current array A is k, the algorithm selects $f(k)$-th element $A[f(k)]$ as the pivot. Let $T_f(n)$ be the worst-case time complexity of the deterministic quicksort algorithm on an array of n distinct numbers already in the sorted order where the pivot selection is done using f. Let $f_1(k) = 1$, $f_2(k) = \lceil k/2 \rceil$, and $f_3(k) = k$. Which of the following statements is **TRUE**?

a). $T_{f_1}(n) = \Theta(n)$, $T_{f_2}(n) = \Theta(n)$, and $T_{f_3}(n) = \Theta(n)$

b). $T_{f_1}(n) = \Theta(n)$, $T_{f_2}(n) = \Theta(n \log n)$, and $T_{f_3}(n) = \Theta(n^2)$

c). $T_{f_1}(n) = \Theta(n^2)$, $T_{f_2}(n) = \Theta(n \log n)$, and $T_{f_3}(n) = \Theta(n)$

d). $T_{f_1}(n) = \Theta(n^2)$, $T_{f_2}(n) = \Theta(n^2)$, and $T_{f_3}(n) = \Theta(n^2)$

e). None of the above Answer

9. Consider an execution of the randomized quicksort algorithm on an array of n distinct numbers. Define $\mathcal{E}_{i,j}$ as the event that the algorithm compares the ith smallest element with the jth smallest element in the array during the algorithm. For the case of $1 \leq i < k < l < j \leq n$, what is the value of $\mathbf{Pr}[\mathcal{E}_{i,j}$ and $\mathcal{E}_{k,l}]$?

a). $\frac{4}{(j-i+1)(l-k+1)}$ Answer: $\mathbf{Pr}[\mathcal{E}_{i,j}] = \frac{2}{j-i+1}$ and $\mathbf{Pr}[\mathcal{E}_{k,l}] = \frac{2}{l-k+1}$, and they are independent events: Conditioning on $\mathcal{E}_{k,l}$, still each element in the interval $(a_i, \ldots, a_j)$ has equal chance to be the first one selected as the pivot in the interval.

b). $\frac{2}{(j-i+1)(l-k+1)}$

c). $\frac{2}{j-i+1} + \frac{2}{l-k+1}$

d). $\frac{2}{j-i+1} + \frac{2}{l-k+1} - \frac{4}{(j-i+1)(l-k+1)}$

e). None of the above

10. Place 100 balls into 100 bins. Each ball placement is done independently and uniformly at random. Let X_i be the number of balls in the i-th bin. Let $\mathcal{E}_{i,j}$ be the event that the i-th ball is in the j-th bin. What is the value of $\mathbf{E}[X_1|\mathcal{E}_{1,1}] + \mathbf{E}[X_1|\mathcal{E}_{1,1} \wedge \mathcal{E}_{2,1}]$?

    a). 5

    b). 4.99

    c). 4.98

    d). 4.97 Answer

    e). None of the above.

11. Let $G = (V, E)$ be a complete graph over 11 vertices. Each vertex selects one of its 10 incident edges independently and uniformly at random. Let $E^* \subseteq E$ be the set of all edges selected by at least one of the two endpoints. What is the expected value of $|E^*|$?

    a). 11 Unlikely; there are 11 vertices and $C(11, 2) = 55$ edges in this complete graph; as each vertex select one random incident edge, it is very unlikely to have all 11 vertices choose an incident endpoint that is not the other 10 vertices at all times

    b). 10.55

    c). 10.45 Answer;
    The probability a random edge (u, v) is chosen by vertex u or v is $\frac{1}{10}$.
    The probability a random edge (u, v) is not chosen by vertex u or v is $1 - \frac{1}{10} = \frac{9}{10}$.
    The probability a random edge (u, v) is not chosen by both vertex u and v is $\frac{9}{10} \times \frac{9}{10} = \frac{81}{100}$.
    The probability a random edge (u, v) is chosen by either vertex u or v is $1 - \frac{81}{100} = \frac{19}{100}$.
    Let sum over all 55 edges:
    So, $|E^*| = \sum_{e \in E} E[X_e] = \sum_{e \in E} \frac{19}{100} = 55 \cdot \frac{19}{100} = \frac{1045}{100} = 10.45$
    Tricky probability (math) question, not all CS students will be able to do this if their probability foundation is not strong, so either they choose b)., correct c)., or this, so rated at 30-70 MCQ.

    d). 10 Unlikely

    e). None of the above. c). is correct

12. A palindrome is a string that reads the same forward and backward.
    We define $\mathbf{lps}(S)$ as the *length* of the longest palindrome subsequence of $S = s_1 s_2 \cdots s_{n-1} s_n$.
    Which of the following statements is **TRUE**?

    a). If $s_1 = s_n$, then $\mathbf{lps}(s_2 \cdots s_n) = \mathbf{lps}(s_1 \cdots s_{n-1})$. No

    b). If $s_1 \neq s_n$, then $\mathbf{lps}(s_2 \cdots s_n) = \mathbf{lps}(s_1 \cdots s_{n-1})$. No

    c). If $s_1 \neq s_n$, then $\mathbf{lps}(s_2 \cdots s_{n-1}) = \mathbf{lps}(s_1 \cdots s_n)$. Not necessarily.
    $\mathbf{lps}(s_1 \cdots s_n) = max(\mathbf{lps}(s_1 \cdots s_{n-1}), \mathbf{lps}(s_2 \cdots s_n))$. This is tricky 50-50 MCQ.

    d). If $\mathbf{lps}(s_2 \cdots s_{n-1}) = \mathbf{lps}(s_1 \cdots s_n)$, then $s_1 \neq s_n$. Yes, this is a stronger condition than c). Page 361 of CP4 Book 2.

e). None of the above.

13. As discussed in class, we define **lcs**(A, B) as the *length* of the longest common subsequence of $A = a_1 a_2 \cdots a_n$ and $B = b_1 b_2 \cdots b_m$. For two sequences A and B (with $n > 0$ and $m > 0$), which is **not** a possible answer of **lcs**(A, B)?

    a). 0 Possible, if A and B has no common value at all.

    b). 1 Possible, if A and B only has one common value.

    c). n Possible, if A = ... of length n and B (of length $m \geq n$) shares common subsequence with all values of A.

    d). m The reverse of c).

    e). $n + m$ Not possible, max value of **lcs**$(A, B) = min(n, m)$. Hopefully easy 90-10 MCQ.

14. In class, we show how to compute the length of **lcs**(A, B), i.e., function `L(i,j)`, in $O(mn)$ time. What is the worst-case time complexity if we compute `L(i,j)` recursively *without* memoization?

    a). $O(m^2 n^2)$ No, worse than that

    b). $\Omega(2^n)$ This was discussed by Diptarka last semester (but somehow not mentioned by Yi-Jun during lecture 06 - or I didn't catch it during the recording). The worst case happens when A and B has no common element: $T(n, m) = T(n-1) + T(n, m-1)$ (when $a_i \neq b_j$). This is the number of paths from $(1, 1)$ to (n, m). $T(n, m) \geq C(n + m, n) > 2^n$ (assuming $m \approx n$). Even if students haven't analyse this properly, this is the only option that is exponential. I rate this as 80-20 MCQ.

    c). $o(n^3)$ No, worse than that

    d). $O(n \log m)$ No, worse than that

    e). $\Theta(mn)$ No, this is *without* memoization

15. In `Change-making` problem, our goal is to find the minimum number of coins denominations $d_1, d_2, \ldots, d_k$ that add up to n cents. The denominations are $\{5, 10, 20, 50, 100\}$ cents (Singapore coin denominations, third series (2013-present; note that 100 cents coin refer to the 1 dollar coin)[1]). Assume that we are only using coins for our (small) purchases in Singapore. Which of the following statements is **FALSE**?

    a). There is no solution if $n\%5 \neq 0$ Yes, because there is no 1 cent coin, the smallest is 5 cents

    b). If $n = 115$ cents, we need at least 3 coins Yes, $100 + 10 + 5 = 115$, 3 coins

    c). If $n = 115$ cents, there are 4 coins that add up to n cents Yes, $100 + 5 + 5 + 5$, 4 coins

    d). If $n = 775$ cents, there are 9 coins that add up to n cents No, minimally we need 10 coins, see option e). Should be an easy MCQ, 90-10 MCQ.

    e). If $n = 775$ cents, we need at least 10 coins Yes, $7 \cdot 100 + 1 \cdot 50 + 1 \cdot 20 + 1 \cdot 5 = 775$, 10 coins

[1] https://www.mas.gov.sg/currency/circulation-currency/circulation-currency-coins

B Essay (25 marks)

B.1 Light Bulbs (7 marks)

There are n light bulbs. You have a special tester which tests whether the bulbs are faulty. To operate this tester, you put a **non-empty set of bulbs** into the tester. The tester will report one of two possible outcomes:

1. *All* the bulbs are good, or
2. *At least one* bulb is faulty, but with a caveat:
   You will not have any additional information on **which bulb** is faulty

B.1.1 Come-up with a Valid Testing Strategy (4 marks)

For all the n bulbs, you need to determine whether they are faulty or not.
Design a valid testing strategy to be able to determine which bulbs are faulty.
Any strategy that works will be given 2 marks.
The optimal strategy that uses minimum number of testings in the worst-case is given full 4 marks.

Split the n bulbs into 1 bulb and $n-1$ others. Test this 1 bulb (a non-empty set of bulbs) with the tester (we will then know with certainty if that 1 bulb is good or actually faulty). We repeat this one-by-one testing until we are done. In total, we test n times... and sounds bad, and sounds not **Divide-**and-Conquer-ish... But actually, this is the best that we can do... (4 marks). PS: The name is actually still abbreviated as D&C: It is **Decrease-**and-Conquer.

PS: A weird strategy is to try all possible subsets of 2^n bulbs, see if this set of all bulbs are good and the other set is all faulty... The number of tries is 2^n... But with generous marking scheme, this will be given 2 marks.

Post grading remarks: Actually, not many tried 2^n tests like that. We realized that $\frac{1}{3}$ of the class tried to force themselves to write something-like-binary-search: Test all bulbs, if something faulty, divide the bulbs into two portion of $\frac{n}{2}$ bulbs each (one side is slightly bigger by one if odd). Repeat the test on **BOTH** sides: This is a valid strategy but will do $2n-1$ tests overall, i.e., $n-1$ tests more in the worst-case. This is judged as 2 marks for this part. PS: Unfortunately, this divide by two strategy and **CHOOSE ONLY ON SIDE** and later claim $\log n$ tests, is unfortunately buggy and judged as wrong (0 mark).

B.1.2 Minimum Number of Testings in the Worst-Case (3 marks)

Find the minimum number of testings required in the worst-case.
Please give the exact answer in terms of n.
If need be, update your previous answer in B.1.1 to use the optimal strategy.

There are n bulbs, and either one can be in two state: good, or faulty. So there are 2^n possible decisions (leaves of decision tree).

A decision tree model needs to be as tall as $h = \log_2 2^n = n$ in order to cover all 2^n possible decisions. Therefore $h = n$, i.e., n is the lowerbound.

So, the previous naive testing strategy that test 1 bulb at a time (not divide by 2, by 3, etc in some form of Divide and Conquer way) is already the optimal one (as it requires exactly n tests in the worst-case and we have just shown that we need at least n tests). PS: if you already arrive at that in 2.1.1, you do not need to think further. Post grading remarks: So many students did not use decision tree model, so we decide to be very lenient and accept most n tests answers. Some who answer n tests but with incorrect explanation were marked down.

B.2 Graph Coloring (8 marks)

A graph $G = (V, E)$ is *bipartite* if its vertex set can be partitioned into two parts $V = X \cup Y$ such that the two endpoints of every edge $e \in E$ belong to different parts.

Let $G = (V, E)$ be any n-vertex bipartite graph where each vertex v is associated with an *arbitrary* set $L(v)$ of $\lceil \log_2 n \rceil + 1$ colors. We emphasize that *different* vertices u and v *may have different* sets $L(u)$ and $L(v)$.

B.2.1 Design and Analysis (6 marks)

Design a randomized or deterministic algorithm that selects a color $\phi(v) \in L(v)$ for every vertex $v \in V$ such that the chosen colors form a *proper* coloring: $\phi(u) \neq \phi(v)$ for every edge $e = \{u, v\} \in E$.

Show that your algorithm computes a desired coloring in polynomial time with a probability of at least $1/2$. You can assume that the bipartition $V = X \cup Y$ of the vertex set V is already given.

Let's say the bipartition is $X \cup Y$.
For each color in the union of all colours $L(v), \forall v \in V$, we randomly assign that color to either to X or Y (with 50-50 chance). Now, each color either belongs to an X-color or Y-color (but never belong to both).

For each vertex $x \in X$, if $L(x)$ contains an X-color, choose any of them, else we fail.
Similarly, for each vertex $y \in Y$, if $L(y)$ contains a Y-color, choose any of them, else we fail.
This can be done in time linear in $\sum_{x \in X} |L(x)| + \sum_{y \in Y} |L(y)|$ (which is polynomial in the input size).

We fail if there is a vertex that we cannot choose a color.
For each vertex, the probability of failure is $\frac{1}{2}^{\lceil \log_2 n \rceil + 1} \leq \frac{1}{2n}$.
Using union bound, the total failure probability $\leq \frac{1}{2}$.

If we successfully color everything, then it must be a proper coloring, since for all $x \in X$, $y \in Y$, $\phi(x) \neq \phi(y)$ as $\phi(x)$ is a X-color and $\phi(y)$ is a Y-color.

Post grading remarks: Some common mistakes are as follows.

- Doing a very brute-force search over all possible proper coloring, involving backtracking when running out of available colors (most likely not finishing in polynomial time).
- Assuming that the input color lists are random (have to do worst-case analysis over all possible inputs).
- A greedy strategy like iteratively selecting an unused color most likely does not work.
- Assuming that all vertices in one part have a common color (not a valid assumption).
- Using a SAT-solver or anything similar (while they might run in linear time for most cases in practice, SAT is not known to be polynomial-time solvable).

B.2.2 Special Case (2 marks)

Suppose you are told that *all* vertices have *the same set* of FOUR (4) colors, i.e., for every vertex $v \in V, L(v) = \{$'red', 'green', 'blue', 'black'$\}$. Solve the above problem *deterministically* in polynomial time for this special case. You can still assume that the bipartition $V = X \cup Y$ of the vertex set V is already given.

Every bipartite graph can be bicolored. Since for this task the bipartition $V = X \cup Y$ is already given, we simply colour $x \in X$ with one color, e.g., 'red' and $y \in Y$ with any other color, e.g., 'green'. This deterministic algorithm will run in $O(n)$ time and is always successful.

Post grading remarks: Some common mistakes are as follows.

- Doing a very brute-force search over all possible proper coloring, involving backtracking when running out of available colors (most likely not finishing in polynomial time).
- A greedy strategy like iteratively selecting an unused color *might* not work, depending on the specific implementation detail.
- Randomness not allowed in this problem.
- We do not assume that the maximum degree is at most 3.
- The four color theorem cannot be used here as the considered graph could be non-planar.

Some students are probably not aware of the assumption that the bipartition $V = X \cup Y$ is already given, so they design an algorithm (e.g., by BFS or DFS) to find this partition. That is of course allowed and should be considered a correct answer, assuming that the algorithm is done correctly.

B.3 LinkedIn (10 marks)

Role play: You are a top student who already satisfied the graduation requirements from a certain university and is now planning on 'learning and expanding network' as much as possible in your upcoming last (optional) semester. Given:

- $1 \leq N$, the integer number of remaining courses in that university that you can take in your last (optional) semester,
- $1 \leq K$, the maximum amount of integer units that you can still take in your last (optional) semester based on that university's rule, and
- a list of N pairs (d_i, s_i) that describe the number of integer unit $1 \leq d_i \leq K$ units and the lecturer name (a short string between 1 to 10 lowercase alphabet characters) of course i (numbered from course 1 to course N).

You need to figure out which courses to take in your final semester in order to maximize the amount of units taken in your last (optional) semester (yes, you like to study so much), but without breaking the university's limit of K units in that last (optional) semester. However, you have an additional requirement: You do not want to take more than one course given by the same lecturer (as each new lecturer that you study with is a new potential connection on LinkedIn![2]).

B.3.1 Subtask 1: At Most Two Different Lecturers (2 marks)

Suppose that of the N remaining courses, there are **at most two** different (challenging) lecturers left. For example $N = 3$ and $K = 10$ and the 3 courses left are: {(1, 'chang'), (2, 'halim'), (5, 'chang')}, then the optimal strategy is take the second course taught by (Prof Steven) Halim for 2 units and the third course taught by (Prof) Chang (Yi-Jun) for another 5 units. This way, you can study 7 more units in your final semester, which is not more than $K = 10$ units and without taking more than one course with the same lecturer.

Design a $\Theta(N^2)$-time complete search/brute force/try all algorithm that can correctly solve this subtask 1. There is no need to supply proof of correctness as long as your complete search is trying all possibilities.

First, in $\Theta(N)$, split N courses into the units taught by the first lecturer and the units taught by the second lecturer (if any, as the wording is 'at most two') - and lecturer name(s) is/are not important, so {(1, 'chang'), (2, 'halim'), (5, 'chang')} is now processed into $\{units_1 = (1, 5), units_2 = (2)\}$. This is $\Theta(N)$.

We take max of $units_1 = \max(1, 5) = 5$ and max of $units_2 = \max(2) = 2$ and max of both, i.e., $\max(5, 2) = 5$. This is the max if we only study from one lecturer. This is $\Theta(N)$. ~~If students miss this possibility, deduct 1 mark~~. Post grading remarks: Doing this will decrease the average of the midterm by around 0.7-0.8 marks, as there are more than $\frac{2}{3}$ students who got this question correct but failed to notice that 'at most two different lecturers' can mean 'there can be just one lecturer'.

Then we try all pairs of $a \in c1$ and $b \in c2$ (possibly an empty set) and as long as $a + b \leq K$, we keep the running max. This is the max if we study from these two lecturers. This is $\Theta(N^2)$. Overall complexity of this Complete Search solution is $\Theta(N^2 + 2N) = \Theta(N^2)$. Post grading remarks:

[2]You are welcome to add our LinkedIn profiles at `https://www.linkedin.com/in/sss1213/` and `https://www.linkedin.com/in/steven7halim/`

$\approx \frac{2}{3}$ of the class are declared correct. If students forgot to check $a + b \leq K$ and only report max units of a pair, then the solution is actually (very) wrong. A few students unfortunately did this and were penalised -2 marks.

B.3.2 Subtask 2: No two courses have the same lecturer (3 marks)

Suppose that of the N remaining courses, no two courses have the same lecturer. For example $N = 3$ and $K = 10$ and the 3 courses left are: {(7, 'diptarka'), (2, 'halim), (5, 'chang')}, then the optimal strategy is take (Prof) Diptarka (Chakraborty)'s and (Prof Steven) Halim's courses for a total of $7 + 2 = 9$ units. There is no other better combination.

Use an algorithm that you have learned in class *during lecture on Week 06* (i.e., Fibonacci, LCS, Knapsack, or Change-making) to correctly solve this subtask 2. There is no need to supply proof of correctness (as we have done so in class) as long as you can show how to use that algorithm correctly. Analyze the Big O time complexity in terms of N and/or K.

As no two courses have the same lecturer, we can simply ignore that all-distinct lecturer names information for all N courses, so this problem degenerates to the standard 0-1 `KNAPSACK` problem discussed in (the second half of) Lecture 06. We set maximum knapsack size $W = K$, we set (w_i, v_i) of each item into (d_i, d_i), i.e., taking a course i uses up d_i units but also increases total value by d_i. Nothing else needs to be changed. The overall time complexity is $O(N \cdot K)$. Post grading remarks: Initially, we hope at least $\frac{1}{3}$ of the class can see this connection, to be revisited during problem reduction in the second half of the semester (Lec09). In reality, we think closer to $\frac{1}{2}$ of the class got this connection, good job :). PS: We ignore small issues as long as the solution is already close to this. This results in $\approx \frac{1}{2}$ of the class are declared correct.

PS: For those who already familiar, this Subtask 2 is an optimization version of `SUBSET-SUM` problem. Post grading remarks: A few (estimated around 10) is aware of this and use the simplified DP SUBSET-SUM solution.

B.3.3 Manually Solve the Full Problem (1 mark)

For example $N = 4$ and $K = 10$ and the 4 courses left are: {(1, 'chang'), (7, 'diptarka'), (2, 'halim), (5, 'chang')}, then the optimal strategy is take (Prof) Chang (Yi-Jun)'s 1 unit course (avoid the 5 units course), (Prof) Diptarka (Chakraborty)'s, and (Prof Steven) Halim's courses for a total of $1 + 7 + 2 = 10$ units. This is the optimal answer.

To ascertain that you have fully understand the full version of this problem, give one integer and a short explanation on how to get that answer, that describes the optimal output for the following test case: $N = 8$, $K = 100$, and the 8 courses are: {(79, 'p'), (43, 'v'), (94, 'a'), (91, 'c'), (25, 'p'), (31, 'a'), (70, 'a'), (12, 'v')}. That's it, there are 4 lecturers {'p', 'v', 'a', 'c'}. You will get partial marks if your answer is within 5 units away of the optimal answer.

Notice that 'a' teaches a course with 94 units (the largest), so the answer is at least this if you only take this 94 course. This is easiest to spot but this is exactly 5 units away from the optimal answer 99 so will be given ~~0.3 marks~~. Post grading remarks: Softmark cannot do 0.3, so this is 0.5 marks.

Now for pairing two courses, with a bit more observation, we see that we can pair 'p' 25 units with 'a' 70 units, totalling $25 + 70 = 95$, one higher than 94 above. This now worth ~~0.6 marks~~. Post grading remarks: Softmark cannot do 0.6, so this is also 0.5 marks, no differentiation with the other one... Lesson learnt, we will scale up final paper to 100 marks, knowing that the smallest unit that Softmark can give is 0.5 marks.

The correct answer is 99 and to spot it one need to probably split 8 courses into $2, 2, 3, 1$ units taught by {'p', 'v', 'a', 'c'}, respectively. Then notice that 'c' has 91 units course that can't really be combined with anyone else (no one else taught a course with at most $100 - 91 = 9$ units... But if we combine 'p', 'v', and 'a', we can have $25 + 43 + 31 = 99$ and this is the maximum possible. Post grading remarks: Although many ($\frac{1}{2}$ of the class) get this, we actually expected much more than that to get this. Perhaps students ran out of time.

B.3.4 Solve the Full Problem in $o(N^2 \cdot K)$ (4 marks)

Design a Dynamic Programming (DP) algorithm (just describe the base case(s) and the inductive step) to fully solve this problem and analyze its Big O time complexity in terms of N and/or K. In order to get full marks, your DP algorithm must run in polynomial time and in $o(N^2 \cdot K)$.

Same with Subtask 1, first, in $\Theta(N)$, split N courses into the units taught by the individual lecturers (but now for this, we need a Hash Table that maps a string (a lecturer name) into a list of integer unit(s) of course(s) that he/she teaches). So {(79, 'p'), (43, 'v'), (94, 'a'), (91, 'c'), (25, 'p'), (31, 'a'), (70, 'a'), (12, 'v')} is now processed into {'p' = (79, 25), 'v' = (43, 12), 'a' = (94, 31, 70), 'c' = (91)}. This is $O(N)$ - a bit hard to analyse Hash Table efficiency using Θ notation.

Then we clean up that data a bit more as we do not really care about the lecturer names. We just need the list of integer unit(s) of course(s) that he/she teaches. So {'p' = (79, 25), 'v' = (43, 12), 'a' = (94, 31, 70), 'c' = (91)} is now processed into $units = [(79, 25), (43, 12), (94, 31, 70), (91)]$. Let's introduce a new variable here, let M be the number distinct lecturers where $1 \leq M \leq N$. This process is still $O(M + N) = O(N)$. Now the key observation is that for each distinct lecturer, we can only take at most one of his/her course, or skip this lecturer altogether.

So, we still going to use `0/1-KNAPSACK` Dynamic Programming (DP) solution like in Subtask 2 (or for those who use `SUBSET-SUM` route), but we need to modify it a bit further. Like in Subtask 2, we also set maximum knapsack size $W = K$, But this time, the inductive/recursive part needs to be changed as follows:

Let $m[i, j]$ = the max units that you can take in an optimal solution when considering the first i lecturers and we have j units left:

Similar like `0/1-Knapsack` base cases:

$m[i, < 0] = -\infty$, we cannot read any more course.
$m[0, j] = 0$, no more course to read.

If the i-th lecturer is to be skipped (we skip all his/her courses),
then $m[i, j] = m[i-1, j]$, i.e., we can still read j units, unchanged.

If we want to take any of the i-th lecturer's course(s) – can be 1,2,...,up to N (there is a loop here),
then $m[i, j] = max(d_i + m[i-1, j-d_i])$ (we can take **only one**, and move on).

Nothing else needs to be changed (i.e., still implement this in bottom-up or in top-down but with memoization). Not mentioning this cause issue as the recurrence is exponential if executed verbatim.

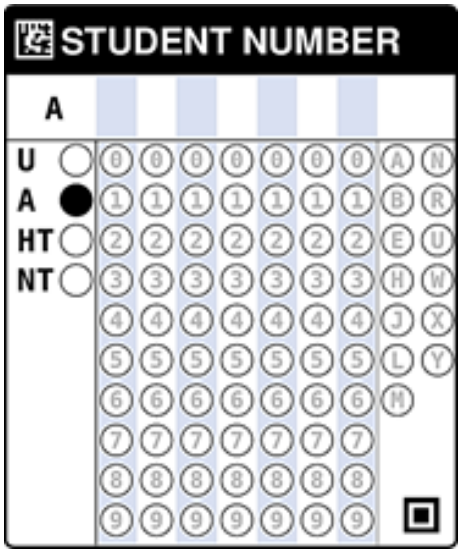
The overall time complexity is $O((M \cdot K) \cdot X)$ where $M \cdot K$ is the size of the 2D DP table of 0/1-Knapsack (the number of distinct states) and X is the average number of courses of each lecturer that we have to iterate per state. One possible way to argue (there are a few acceptable ways): As we are splitting N courses (N balls) into M distinct lecturers (M bins), we can treat this as balls and bins problem, therefore the expected number of courses/balls for each distinct lecturer/bin is $\frac{N}{M}$ – this is what we expect if all lecturers have uniform number of courses. But if there is one or a few super lecturer(s) with many courses, we will expect M to drop and the rest to have very few courses. We can thus simplify the overall time complexity to $O((M \cdot K) \cdot \frac{N}{M}) = O(N \cdot K)$ again. Post grading remarks: There are also other strategies that are acceptable other than 'grouping courses by lecturers as shown above', i.e., to sort the N courses by lecturer names, so that courses that are taught by one lecturer are contiguous. We can design an $O(1)$ 'jump' to the first course of the next lecturer sub-routine or use $O(\log N)$ binary search to find the last course taught by this lecturer (the one after this is the first course of the next lecturer). Both are acceptable ways that are in little-$o(N^2 \cdot K)$.

A wrong analysis is to analyze this as $O((N \cdot K) \cdot N = O(N^2 \cdot K)$ (not tight, as we want something better than that, hence the requirement is little-$o(N^2 \cdot K)$). Post grading remarks: But we mostly ignore this as there were only ≈ 80 attempts of this question and only less than ≈ 20 answers are judged (nearly) correct, with most wrong answers judged as 'not able to efficiently deduce if a certain lecturer has been used before from an earlier course that has been taken'.

This task may be used as future (semi)-PA2 task (as it is actually from an online judge, but the actual source URL is not shown).

The Answer Sheet

Write your Student Number in the box below using **(2B) pencil**.
Do NOT write your name.



Write your MCQ answers in the special MCQ answer box below for automatic grading.
We do not manually check your answer.
Shade your answer properly (use (2B) pencil, fully enclose the circle; select just one circle).

No	1	2	3	4	5	6	7	8	9	10
A	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
B	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
D	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
E	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

No	11	12	13	14	15
A	◯	◯	◯	◯	◯
B	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯
D	◯	◯	◯	◯	◯
E	◯	◯	◯	◯	◯

Box B.1.1. Come-up with a valid testing strategy

Box B.1.2. Minimum number of testings in the worst-case

Box B.2.1. Design and Analysis

Box B.2.2. Special case: all vertices have the same set of FOUR (4) colors

Box B.3.1. Subtask 1: At most two different lecturers

Box B.3.2. Subtask 2: No two courses have the same lecturer

Box B.3.3. Manually solve the full problem

Box B.3.4. Solve the full problem in $o(N^2 \cdot K)$

– END OF PAPER; All the Best –