National University of Singapore

School of Computing

CS3230 - Design and Analysis of Algorithms

Final Assessment

(Semester 1 AY2025/26)

Time Allowed: 150 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 TWENTY-FOUR (24) printed pages, including this page.
3. This is an **Open Book** Assessment.
   The only allowed electronic device is a non-programmable calculator.
4. For Section A, use the boxes at page 15 (use 2B pencil).
   For Section B, answer **ALL** questions within the **boxed spaces** at page 16-23.
   If you need extra working space, you can use page 24, but indicate clearly in the respective box.
   Each essay question in Section B has custom marking scheme.
   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 100 marks.
   It will then be scaled to 40% of the course weightage.

A MCQs ($26 \times 2 = 52$ marks)

Answers: bbacb abecb aebaa baede bddaa b

1. For any two functions f and g such that $f(n) \in o(g(n))$,
   which of the following statements is **TRUE**?

   a). $f(n) < g(n)$ for all positive integers n. Not true for small $n < n_0$

   b). There are infinitely many positive integers n such that $f(n) < 0.001 \cdot g(n)$. Answer; yes any $n \geq n_0$; Lecture 01b review; Rated 80-20 MCQ

   c). There exists a positive integer n such that $f(n) < (g(n))^{0.99}$. No

   d). $g(n) \in o(f(n))$. Clearly no

   e). None of the above. Option b). is the answer

2. Which of the following functions grows fastest, asymptotically, as n increases?

   a) $\log(n!)$ By Stirling's approximation, this is also $\Theta(n \log n)$, same as option e)., so also rule this out quickly

   b) $2^{\log^2 n}$ Answer; this versus option d). as option a)., c)., and e). have n not as exponent. Rated 70-30 MCQ

   c) n^2 Grows faster than option a). and e)., but option b). and d). will beat this eventually.

   d) $(\log n)^{\log n}$ Do limit test between option b)./option d). The result is the limit grows to infinity, so b). grows faster.

   e) $n \log n$ Rule this out immediately

3. Consider the recurrence relation: $T(n) = T\left(\frac{999}{1000} \cdot n\right) + T\left(\frac{1}{1000} \cdot n\right) + n$.
   Which of the following statements is **TRUE**?

   a). $T(n) \in \Theta(n \log n)$. Answer; need to draw recursion tree and realizes the recursion depth is still $\log n$; kinda discussed in the Finale lecture 12: Q1 of VA OQ 12; Rated 70-30 MCQ

   b). $T(n) \in \Theta(n^{1.001})$.

   c). $T(n) \in \Theta(n^{1.999})$.

   d). $T(n) \in \Theta(n^2)$.

   e). None of the above.

4. Which of the following is true about the solution to the recurrence $T(n) = n \cdot T(n/2) + 1$?
   (For simplicity, assume that n is always a power of 2.)

   a) $T(n) \in o(n^{(\log n)/2})$

   b) $T(n) \in \Theta(n^{(\log n)/2})$

   c) $T(n) \in \omega(n^{(\log n)/2}) \cap o(n^{\log n})$ Answer. $T(n) = \Theta(n^{(\log n+1)/2})$. Rated 20-80 MCQ

   d) $T(n) \in \Theta(n^{\log n})$

e) $T(n) \in \omega(n^{\log n})$

5. Which of the following statements correctly describes the fundamental *loop invariant* maintained at the start of every iteration of Dijkstra's algorithm (regardless of implementation details, e.g., the chosen data structure), where s is the source vertex, R is the set of R(esolved) vertices, and $d[u]$ is the computed distance to vertex u?

   a). For all vertices v in the set $V \setminus R$, the distance $d[v]$ is always larger than the length of the actual shortest path from s to v. No, they are not (confirmed) shortest path values yet, it can be overestimated at this point, it can be larger or equal, the word 'always' is too strong

   b). For all vertices u in the set R, the distance $d[u]$ is equal to the length of the actual shortest path distance `dist(s, u)`. Answer. Easy MCQ. Rated 80-20

   c). The algorithm guarantees that the total cost of updating distances (relaxation) performed so far is $O(m + n \log n)$, where $n = |V|$ and $m = |E|$. Loop invariant is for proving correctness, not about time complexity analysis; anyway the exact cost depends on the chosen data structure: Binary vs Fibonacci heap, and this info is not told

   d). The total number of edges relaxed so far is $O(m)$ and the total number of vertices inside set R is $O(n)$. Similar issue as option c). not about time complexity. Anyway, the number of vertices inside set R is always $O(n)$ as there are n vertices anyway

   e). The set of edges that cross the partition $(R, V \setminus R)$ is minimized, ensuring the greedy choice remains optimal. This is kinda MST-related proof of Prim's/Kruskal's shown in Lecture 12, not about Dijkstra's

6. Consider the classic Binary Search algorithm written this way:
   BinarySearch(A, lb, ub, x):

   1. If ($lb \geq ub$), return NO
   2. Else:
      i. $mid \leftarrow \lfloor \frac{lb+ub}{2} \rfloor$
      ii. If ($x = A[mid]$), return YES
      iii. If ($x > A[mid]$), BinarySearch(A, $mid + 1$, ub, x)
      iv. If ($x < A[mid]$), BinarySearch(A, lb, $mid - 1$, x)

   Unfortunately, despite being shown in class that BinarySearch has proof of correctness for this recursive algorithm, it is still buggy. Which line of code is the cause of the issue?

   a). 1. There is a typo, it says $lb \geq ub$ instead of $lb > ub$. The equal sign causes case where $lb == ub$, it does not go to the else statement for potential $x = A[mid]$, but immediately reported as NO. The proof of correctness shown in class is for the correct version that checks $lb > ub$. This should be an easy MCQ if student is careful enough. Rated 90-10 MCQ

   b). 2-i. no

c). 2-ii. no

d). 2-iii. no

e). 2-iv. no

7. Consider the following algorithm that takes as input an array of numbers and outputs a number: ALG1(A):

   - $k = \text{length}(A)$
   - If $k < 3$, return $A[0]$
   - Split A into three arrays $A_1 = A[0 \dots \lfloor k/3 \rfloor - 1]$, $A_2 = A[\lfloor k/3 \rfloor \dots \lfloor 2k/3 \rfloor - 1]$, $A_3 = A[\lfloor 2k/3 \rfloor \dots (k-1)]$, // assume that the cost of splitting is $O(1)$
   - Return $(\text{ALG1}(A_1) + \text{ALG1}(A_2) - \text{ALG1}(A_3))/3$

   Which of the following is the running time of the above algorithm when given as input an array of length n? Assume, for simplicity, that n is always a power of 3.

   a) $\Theta(\log_3(n))$

   b) $\Theta(n)$ Answer. Simple MT Case 3: $T(n) = 3 \cdot T(\frac{n}{3}) + \Theta(1)$, so $\Theta(n)$. Rated 90-10 MCQ

   c) $\Theta(n \cdot \log_3 n)$

   d) $\Theta(n^2)$

   e) $\Theta(n^3)$

8. Run randomized quicksort on an array of $n \geq 2$ distinct numbers. Let $(a_1, a_2, \dots, a_n)$ be the numbers in sorted order, and let Y_i denote the expected number of comparisons involving a_i. Which of the following statements is **TRUE**?

   a). There exists $i \in \{1, 2, \dots, n\}$ such that $Y_i \in o(\log n)$.

   b). There exists $i \in \{1, 2, \dots, n\}$ such that $Y_i \in \omega(\log n)$.

   c). $Y_1 < Y_2 < \cdots < Y_n$.

   d). $Y_1 > Y_2 > \cdots > Y_n$.

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

9. Consider random numbers $a_1, a_2, \dots, a_n$, where each a_i is drawn independently at random from $\{1, 2, \dots, n\}$. What is the expected number of triples (a_i, a_j, a_k) such that $a_i = a_j \neq a_k$ and $i < j < k$?

   (a) $\Theta(\sqrt{n})$

   (b) $\Theta(n)$

   (c) $\Theta(n^2)$ Answer; $Pr(a_i = a_j \neq a_k) = Pr(a_i = a_j) \cdot Pr(a_i \neq a_k) = \frac{1}{n} \cdot (1 - \frac{1}{n}) = \frac{n-1}{n^2}$. The expected value is $C(n, 3) \cdot \frac{n-1}{n^2} = \frac{n \cdot (n-1) \cdot (n-2)}{3 \cdot 2 \cdot 1} \cdot \frac{n-1}{n^2} \approx c \cdot n^2$. Quite math heavy, rated 50-50 MCQ

(d) $\Theta(n^3)$

(e) None of the above.

10. Given an array of n characters S, your task is to count the number of palindromic subsequences of S. For example, if $S = [1]$, the number of palindromic subsequences is 1 (which is just "1"). And if $S = [1, 0, 0]$, the count is 4 (which are 1, **0**, 0, and 00). Note that if the same sequence (e.g. "0") appears more than once as a subsequence (as the **second** and third answers as highlighted above), it is counted each time.

    For $i \leq j$, let $C[i][j]$ denote the number of palindromic subsequences of the sub-array $(S[i], S[i+1], \ldots, S[j])$. Which of the following equations is true for i, j such that $i < j$ and $S[i] = S[j]$?

    (a) $C[i][j] = C[i][j-1] + C[i+1][j] - C[i+1][j-1] + 1$

    (b) $C[i][j] = C[i][j-1] + C[i+1][j] + 1$ Answer. Rated 70-30 MCQ

    (c) $C[i][j] = C[i][j-1] + C[i+1][j] + C[i+1][j-1] + 1$

    (d) $C[i][j] = C[i][j-1] + C[i+1][j] - C[i+1][j-1]$

    (e) None of the above.

11. Suppose the top-down Dynamic Programming (DP) $Fib(n)$ (or $F(i)$) is written in this way:

```
from functools import lru_cache

# in class demo, we use @cache, which essentially @lru_cache(maxsize = None)
@lru_cache(maxsize = 3) # explained below
def F(i):
  if i in [1, 2]: # we use section B.2. definition
    return 1
  else: # i >= 3
    return (F(i-1) + F(i-2)) % M # Calls F(i-1) first before F(i-2)

M = 10**9+7 # a prime modulo
print(F(77)) # is this fast or slow?
```

    Python's `functools.lru_cache` decorator provides an easy way to implement a **Least Recently Used (LRU)** cache for functions. User can specify the `maxsize` parameter to control the cache's memory footprint and ensure that only a limited number of the most recently used results are stored. Basically, `@lru_cache(maxsize = 3)` stores the **last three** executed $F(i)$ values.

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

    a). $F(i)$ still runs in $O(n)$ With `maxsize = 3`, a value of $F(i)$, $F(i-1)$, and $F(i-2)$ are always in the cache, allowing for fast computation of $F(i+1)$. Rated 60-40 MCQ

    b). $F(i)$ runs very slowly as there are lots of cache miss, we must set `maxsize = n` (or `None`) No, not slow. But many will pick this.

c). $F(i)$ becomes wrong No, it still computes the i-th Fibonacci correctly, easy to rule out

d). $F(i)$ causes runtime error No, easy to rule out

e). None of the above Option a). is correct.

12. Suppose the correct Python implementation shown in class for `https://leetcode.com/problems/longest-common-subsequence`:
is edited a bit, again about changing `@cache` to `@lru_cache`.

```
class Solution:
  def longestCommonSubsequence(self, text1: str, text2: str) -> int:
    m, n = len(text1), len(text2) # m, n are length of text1/text2, respectively
    # @lru_cache(maxsize = SEE THE MCQ OPTIONS BELOW)
    def LCS(i, j):
      if i < 0 or j < 0: return 0
      return 1+LCS(i-1,j-1) if text1[i]==text2[j] else max(LCS(i-1,j), LCS(i,j-1))
    return LCS(m-1, n-1)
```

Which of the following choice allows for $O(m \cdot n)$ DP, where $m = len(text1)$ and $n = len(text2)$, respectively?

a). `@lru_cache(maxsize = m)` No

b). `@lru_cache(maxsize = n)` No

c). `@lru_cache(maxsize = max(m, n))` No, but I think some students will pick this (the longer dimension)

d). `@lru_cache(maxsize = min(m, n))` No

e). None of the above It has to be set as `@lru_cache(maxsize = m*n)` (or `None`). This time the recursion branches in a complicated way and depends on text1 and text2 content (see the recursion tree animation at `https://visualgo.net/en/recursion?slide=5-5`). Only saving last m, last n, or last $max(m, n)$ is not okay as this is not a bottom-up DP that does processing row by row (or column by column). We have to save last $m \cdot n$ calls... Rated 40-60 MCQ

13. Consider the 0/1-Knapsack problem where you have n items with weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$, respectively, and the capacity W. The task is to pick a subset of these items such that their total weight is at most W, and their total value is as large as possible. Suppose you are guaranteed that the weights are in non-increasing order ($w_1 \geq w_2 \geq \cdots \geq w_n$) and the values are in non-decreasing order ($v_1 \leq v_2 \leq \cdots \leq v_n$). Which of the following is a greedy-choice property possessed by the problem in this case? (Assume that the optimal solution is non-empty.)

a) There is an optimal solution that contains the first item Clearly wrong, as taking heaviest and lowest value item is clearly not optimal. $W = 2$, $w = [2, 1]$, $v = [1, 100]$, clearly taking the first item is a bad idea.

b) There is an optimal solution that contains the last item Answer, very hard to break, and indeed there is a simple exchange argument for this special case. Rated 30-70 MCQ as many will pick e).

c) There is an optimal solution that contains the item with value-to-weight ratio closest to 1 Easily shown to be not optimal too. $W = 2$, $w = [1, 1]$, $v = [1, 100]$, clearly taking the first item with value-to-weight ratio of 1 is a bad idea.

d) There is a greedy-choice property, but it is not any of the above As option b). is correct, this is not correct.

e) There is no greedy-choice property as the problem does not have a correct greedy algorithm Quite tricky, many who just remember that 0/1-Knapsack is NP-complete will assume there is no greedy algorithm

14. Consider an alphabet of size n with symbol frequencies:

$$\left\{\frac{2^{-1}}{m}, \frac{2^{-2}}{m}, \frac{2^{-3}}{m}, \ldots, \frac{2^{-n}}{m}\right\}, \quad \text{where } m = \sum_{i=1}^{n} 2^{-i}.$$

Let γ be the corresponding Huffman code.
Consider the average bit length (ABL) and the height of the Huffman tree.
Which of the following statements is **TRUE**?

a). ABL is $\Theta(1)$; tree height is $\Theta(n)$. Answer. After checking that specific distribution for small n (and m) and quickly running Huffman algorithm, we can see that the Huffman Tree will be quite skewed, with height $\Theta(n)$ and the ABL is not growing fast, not as fast as $\Theta(\log n)$. Rated 60-40 MCQ, but the wording was $O(1)$ vs $(\log n)$, now changed to Θ notation

b). ABL is $\Theta(1)$; tree height is $\Theta(\log n)$.

c). ABL is $\Theta(\log n)$; tree height is $\Theta(n)$. No longer the answer when the ABL is changed to $\Theta(\log n)$. But for this paper, we accept both a). and c), as the previous version $O(\log n)$ is also correct.

d). ABL is $\Theta(\log n)$; tree height is $\Theta(\log n)$.

e). None of the above.

15. In a dynamic table, the array size doubles whenever it becomes full. It is known that under this rule, the amortized cost per operation over n insertions is $O(1)$. Now consider a variant: when the current array of size k becomes full, the new array size becomes $\lceil k + k^{0.3} \rceil$ instead of $2k$. Which of the following bounds is the **tightest asymptotic upper bound** on the amortized cost per operation over n insertions?

a). $O(n^{0.7})$. Answer. Rated 20-80 MCQ

b). $O(n^{0.3})$.

c). $O(\log n)$.

d). $O(\log \log n)$.

e). $O(1)$.

16. Consider a data structure that supports two kinds of operations op_1 and op_2. In a *random sequence* of n operations, each operation is randomly chosen to be either op_1 or op_2, with probability 1/2 each. Suppose the amortised cost of op_1 is 1 and that of op_2 is 3. Which of the following statements is implied by this amortised analysis?

    a) The expected actual cost of a random sequence of n operations is equal to $2n$

    b) The expected actual cost of a random sequence of n operations is at most $2n$ Answer. Mixing randomization and amortization, some students will be confused. Rated 50-50 MCQ

    c) The worst-case actual cost of any sequence of n operations is equal to $2n$

    d) The worst-case actual cost of any sequence of n operations is at most $2n$

    e) None of the above

17. Consider the following algorithm that takes as input two non-negative integers:

    ALG2(a, b):

    - If $a < b$, return ALG2(b, a)
    - If $b = 0$, return a
    - Return ALG2($a - b$, b)

    Suppose a and b are numbers that can be written using n bits each. Which among the following is the tightest upper-bound on the worst-case complexity of ALG2(a, b)?

    a) $O(2^n)$ This is the original intended correct answer. This is GCD algorithm, but done naively instead of using ALG2($a\%b$, b), so $O(m)$ where m is the integer value. Input size is n bits, means $m = 2^n$. Rated 50-50 MCQ

    b) $2^{O(n)}$ This answer is also accepted, because it is also reasonable to assume that the cost of comparing two n-bit integers ($a < b$ and $b = 0$) is $\Theta(n)$ and not $\Theta(1)$. This leads to the time complexity bound $n \cdot 2^n$, which is only correct for $2^{O(n)}$. We accept both a). and b). for this paper.

    c) $O(\log n)$

    d) $O(n^2)$

    e) $O(n)$

18. For any two decision problems A and B such that $A \leq_P B$,
    which of the following statements is **TRUE**?

    a). If A can be solved in $O(2^n)$ time, then B can be solved in $O(7^n)$ time.

    b). If A cannot be solved in $O(2^n)$ time, then B cannot be solved in $O(7^n)$ time.

c). If B can be solved in $O(2^n)$ time, then A can be solved in $O(7^n)$ time.

d). If B cannot be solved in $O(2^n)$ time, then A cannot be solved in $O(7^n)$ time.

e). None of the above. Answer. Hopefully most students can see that the 4 options above are wrong. Rated 70-30 MCQ

19. Suppose there is a polynomial-time reduction algorithm R from decision problem A to decision problem B. Given an instance α of A, the reduction produces an instance $\beta = R(\alpha)$ of B. Which of the following statements is *not* implied by the above?

a) If α is a YES instance of A, then β is a YES instance of B

b) If β is a NO instance of B, then α is a NO instance of A

c) If there is a polynomial-time algorithm for B, then there is one for A as well

d) If A is NP-complete, then B is also NP-complete Answer. B is NP-hard, but not necessarily in NP. Hopefully a bit more will choose this answer. Rated 60-40 MCQ

e) None of the above Hopefully less will pick this, thinking that there is no issue with the other 4 options (option d). has an issue)

20. Which of the following decision problems is **NP-complete**?

a). Does a graph G have a clique of size at most 10000?

b). Does a graph G have an independent set of size at most 10000?

c). Does a graph G have a vertex cover of size at most 10000?

d). Is the longest common subsequence length between two strings A and B at most 10000? Clearly no, we know this has polynomial solution

e). None of the above. Answer; very tricky, requires understanding of the meaning of 'at most 10000'. Rated 20-80 MCQ

21. Let $T = (V, E)$ be an undirected tree with at least two vertices. A vertex subset $D \subseteq V$ is called a *dominating set* if for every vertex $v \in V \setminus D$, there exists a vertex $u \in D$ such that $\{u, v\} \in E$. A dominating set D is *minimum* if $|D| \leq |D'|$ for every dominating set D' of T.
Which of the following statements is **TRUE**?

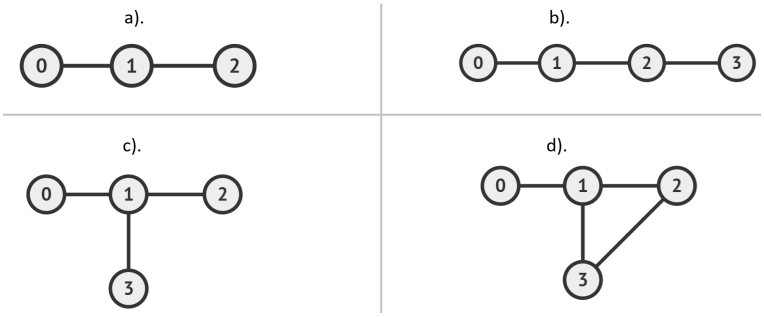
a). For any leaf v in T, there exists a minimum dominating set D such that $v \in D$.

b). For any leaf v in T, there exists a minimum dominating set D such that $u \in D$, where u is the unique neighbor of v. Answer. Rated 33-66 MCQ

c). Let v be a vertex of maximum degree in T. There exists a minimum dominating set D such that $v \in D$.

d). Let v be a vertex of maximum degree in T. There exists a minimum dominating set D such that $u \in D$, where u is any neighbor of v.

e). None of the above.



22. Which of the following graph a)., b)., c)., d). has dominating set (see the previous Q21) of minimum size that is smaller than vertex cover of minimum size on the same graph?

    If there is no answer, select option e). None of the above.

    Answer is d). Minimum VC will need at least {1,2} or {1,3} of size 2 vertices. Minimum dominating set is just {1}. For the other three options, both Minimum VC and minimum dominating set sizes are the same: both 1 for option a). and c).; both 2 for option b). The graphs are small enough for students to brute force. Option a)., b)., and c). Rated 80-20 MCQ

23. In the median-of-medians linear-time selection algorithm, we partition the input array A into $\lceil n/5 \rceil$ groups of size 5 (except possibly the last one), and select x as the median of the medians of all groups. It is known that

$$\max\{\, |\{y \in A \mid y < x\}|,\ |\{y \in A \mid y > x\}|\,\} \leq \tfrac{7}{10}\, n.$$

    Now suppose we modify the algorithm by partitioning A into $\lceil n/15 \rceil$ groups of size 15 (except possibly the last one). Which of the following gives the **tightest upper bound** on

$$\max\{\, |\{y \in A \mid y < x\}|,\ |\{y \in A \mid y > x\}|\,\}?$$

    a). $\frac{8}{15}n$.

    b). $\frac{9}{15}n$.

    c). $\frac{10}{15}n$.

    d). $\frac{11}{15}n$. Answer; nice extrapolation from lecture 11 median of median (group 5) example to group of 15. With a bit of work, this is computeable. Rated 70-30 MCQ

    e). $\frac{12}{15}n$.

24. In class, we have seen that we can find the k-th order statistic of an array (selection problem) in $O(n)$ time whereas sorting the entire array has a tight lower bound of $\Omega(n \log n)$ for comparison-based algorithms. Which of the following statements is **TRUE**?

    a). Selecting the k-th order statistic of an array is computationally easier than sorting the entire array first and report the k-th sorted index. Answer, mentioned in the lecture note 11/tutorial 11, Rated 90-10 MCQ

b). $O(n)$ selection algorithm can break the comparison-based sorting lower bound of $\Omega(n \log n)$ because the algorithm does not do comparisons. No, it does a lot of comparisons, e.g., finding medians of 5.

c). $O(n)$ selection algorithm only achievable in best-case scenario. No, it is the worst-case analysis.

d). We can reduce the sorting problem to selection problem to get an $O(n)$ solution. To reduce sort into n calls of selection, we will end up with $O(n \cdot n) \in O(n^2)$ algorithm.

e). None of the above. Option a). is true.

25. The DP formulation of the Bellman–Ford algorithm computes shortest path lengths $L(v, i)$ for vertices $v \in V$ using up to i edges. This requires $O(|V|^2)$ space if implemented verbatim, but can be optimized to use only $O(|V|)$ space. Which statement is the reason for this optimization?

a). Calculating $L(v, i)$ only relies on the computed values from the immediately preceding iteration $(i - 1)$, so we just need 1D array L that can be constantly overwritten. Answer. This is discussed in finale lecture 12. Rated 80-20 MCQ

b). The saving is achieved by switching the data structure from an Adjacency List to an Edge List, simplifying memory access during relaxation. No, this is not the reason.

c). The total space required is $O(|V|)$ because the space requirement can be amortized. This does not make sense.

d). The shortest paths is at most $O(|V|)$ distinct edges due to the Optimal Substructure property, so only $O(|V|)$ space is needed. This does not make sense.

e). None of the above.

26. Which of the following statements about Minimum Spanning Tree (MST) is **TRUE**?

a). Prim's algorithm to solve the MST problem is a Dynamic Programming algorithm. No, it is a greedy algorithm

b). There is a greedy algorithm for the MST problem where the greedy choice is to delete the **maximum** weighted edge e first if deleting e will not lead to a disconnected graph. Yes actually, this is the Reverse-Delete algorithm. Not popular because it is slow (far slower than the $O(E \log V)$ Kruskal's or Prim's). Not many knows this but if students understand the discussion of MST in the final lecture, students can try proving this greedy choice even without knowing Reverse-Delete algorithm, anywah this is rated 50-50 MCQ

c). Prim's algorithm runs in $O(V^2)$ time if we use Binary Heap data structure. No, it is $O(E \log V)$

d). There is no other algorithm to solve the MST problem other than Prim's algorithm. No, in the final lecture, the name Kruskal's and Boruvka's are/(will be) mentioned, although option b). Reverse-Delete is not.

e). None of the above. Option b). should be chosen. Otherwise student will choose this.

B Essay

B.1 Optimal Internships (16 marks)

In today's job market, it can be difficult to secure a good full-time job after graduation. Therefore, gaining experience through ~~as many~~ high-quality internships is important.

You are given a positive integer n representing the number of internships and a 2D floating-point array *internships* of size $n \times 3$, where $internships[i] = [from_i, to_i, exp_i]$, with:

- $from_i$ and to_i ($1.0 \leq from_i < to_i \leq 10^9$) — the start and end times of internship i (in arbitrary time units), and
- exp_i ($1.0 \leq exp_i \leq 10^9$) — the experience gained from completing the internship (in arbitrary experience units).

Return the **maximum total experience** you can earn by selecting a subset of internships such that no two chosen internships overlap in time (you cannot be interning at two different places at once). Note that you may start a new internship immediately after finishing another — that is, if internship i ends at time to_i and another lucrative internship j starts at time $from_j$, you may take both if $from_j + 0.0001 \geq to_i$, to cater for floating-point precision issue, as shown in the example below.

For example, if $n = 3$ and $internships = [[3.3333, 6.6666, 77.7], [1.0, 4.0, 70.0], [3.9999, 7.0, 7.6]]$, the maximum total experience is 77.7, achieved by just taking $internship[0]$. Note that $internship[1]$ and $internship[2]$ can be taken one after another since $internship[1]$ ends at 4.0 and $internship[2]$ starts at 3.9999, and $3.9999 + 0.0001 \geq 4.0$ — however, the total experience gained would be only $70.0 + 7.6 = 77.6$.

B.1.1 All Seven Point Sevens (7 marks)

Prof Halim said that if all internships worth the same experience of 7.7 (i.e., $exp_i = 7.7, \forall i \in [0..n-1]$), this problem is amenable to greedy algorithm and students should just try to secure **as many** internships as possible. Prove that Prof Halim is correct by describing a correct **greedy algorithm** with the following four-points format:

1. Give the optimal substructure proof,
2. State the appropriate greedy choice and prove its correctness,
3. Outline a greedy algorithm (and any required data structure(s)), and
4. Analyze the time complexity of the greedy algorithm.

This is intended to be a giveaway as this is a tiny variation of tut07 q4+5 about activity selection problem. Student can say that this final question: **unweighted (actually constant-weighted)** interval scheduling reduces to that activity selection problem (kinda review of Lecture 09 and Tutorial 07), i.e., we can take as many non-overlapping intervals (internships) as much you can as each internship worth the same. Once we get the answer of the activity selection problem (just the cardinality, not

the set of selected activities), we transform back the solution for this 'unweighted' interval scheduling by multiplying the maximum number of non-overlapping intervals by 7.7 to get the final answer (forgetting to multiply by 7.7 has a safety net of 4 out of 7 partial marks).

The four-points format answers are exactly like the one shown in tut07 q5 and thus not copy-pasted again here (only small adjustment is needed, e.g., that multiplication of 7.7). PS: The accepted time complexity is in $O(n \log n)$ because of the comparison-based sorting (it is a bit hard to argue for $O(n)$ sorting here due to floating-point inputs).

Grading remarks from Alvin Yan:

Remarks for B11: This question is indeed equivalent to the activity selection tutorial, so one can check the solution file for how the proofs should be written. For the optimal substructure proof, answers that conveyed the correct cut-and-paste idea in this context were accepted. Ideally some notation is used to make the proof clean and easier to follow (e.g. let S be an assumed optimal solution to the problem, consider removing an internship i from S, and so on) but I was not too strict. One common error was to say that the subproblem is the full set of internships minus some fixed internship $i \in S \rightarrow$ all other internships overlapping with i should also be removed.

For greedy choice, a number of answers gave incorrect greedy choices - e.g. take the shortest internship, take the internship with fewest conflicting internships - those are incorrect. Otherwise most proofs were written sufficiently well.

Some issues:

- in an assumed optimal solution S, the exchange has to be with the internship that starts last (or ends earliest) in S with the greedy solution, not just exchange any internship in S for the greedy choice.
- some proofs did not do an exchange argument and instead made hand wavy statements like 'this choice allows for more time to take another internship' which is not sufficient as proof.

More or less students had no issue describing the algorithm. A more efficient $O(n)$ time implementation to iterate through internships once and ignoring overlaps, or $O(n^2)$ by doing an additional loop to remove overlaps were accepted. For running time, one issue is that some indicated $O(n)$ running time with no justification - even the sample instances given show it may not be sorted. Some claimed radix sort for $O(n)$ time algorithm, but radix sort cannot be used on arbitrary precision floating point values. Be careful of when algorithms you learn can be applied.

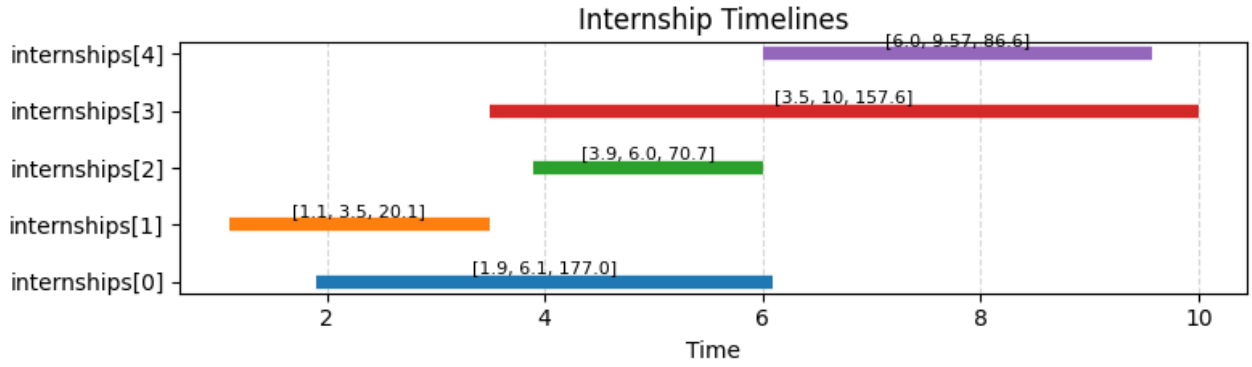
B.1.2 Testing Your Understanding (2 marks)

For both sub-questions, just output the answer (a floating-point).
What is the maximum total experience if $n = 5$ but:

1. $internships = [[1.9, 6.1, 7.7], [1.1, 3.5, 7.7], [3.9, 6.0, 7.7], [3.5, 10, 7.7], [6.0, 9.57, 7.7]]$,
2. $internships = [[1.9, 6.1, 177.0], [1.1, 3.5, 20.1], [3.9, 6.0, 70.7], [3.5, 10, 157.6], [6.0, 9.57, 86.6]]$.

For the first one, the answer is $3 \times 7.7 = 23.1$, as *internships*[1] (orange), *internships*[2] (green), and *internships*[4] (purple) are not overlapping. This test case is to help students answer Section B.1.1.

For the second one, the answer is 177.7. This test case is to help students answer Section B.1.3. There are three (very) close possibilities (refer to the interval chart), hence the grading is super strict:



1. Taking *internships*[1] (orange) and *internships*[3] (red) results in $20.1 + 157.6 = 177.7$ (this is the maximum),
2. Taking *internships*[1] (orange), *internships*[2] (green), and *internships*[4] (purple) (the typical output of a greedy solution for the unweighted version) results in $20.1 + 70.7 + 86.6 = 177.4$ (not the maximum, so wrong answer),
3. Taking *internships*[0] (blue) only gains 177.0 experience (also not the maximum, so wrong answer), PS: Other combinations give worse total experience.

Leaving this question blank is 0 mark, but save a bit exam time, but it may be worthwhile to get up to 2x1 = 2 marks by drawing a sketch like above in a few minutes.

Grading remarks from Alvin Yan:

Remarks for B121: Most students could get it correct. It may be the case that those did not might have thought that you cannot do internship [3.9, 6.0, 7.7] and [6.0, 9.57, 7.7] together - read the constraints in the question again carefully.

Remarks for B122: Quite a number of students were tricked into taking the solution that gives 177.4 experience, instead of the optimal of 177.7.

B.1.3 Not All Internships Are Equal (7 marks)

Solve the full problem described in section B.1 using any technique(s) that you learn in class. There are two possible marks depending if your correct solution is $O(n^2)$ or in $o(n^2)$.

We need to first sort the *internships* by $from_i$ field (it is ok if there are ties; break the ties arbitrarily) so that we can gain a certain speed-up below. The $from_i$ field is in floating-point, making the standard form of Counting Sort and/or Radix Sort 'not really applicable'. It is okay to just use $O(n \log n)$ comparison-based sorting algorithm (e.g., Merge Sort) as to get full marks, we just need to have a

final solution that is in $o(n^2)$. This is kinda review of Lecture 02 (Merge Sort), 04+05 ((Randomized) Quick Sort), 11 (Linear-time Sorting).

Now the expected solution is DP with state $maxExp(i)$ – what is the maximum total experience if we consider internship $[i..n-1]$? - we can also start from the back $n-1$ to front 0 and the answer below has to be adjusted a bit.

$maxExp(n) = 0$, base case, no more internship to consider.

$maxExp(i) = max(maxExp(i+1), internship[i][2] + maxExp(j))$.
Basically, skip internship i and go to subproblem $maxExp(i+1)$ – must be optimal otherwise we can do cut-and-paste proof to get 'a more optimal answer', or take it (and get its experience), but jump to next legal internship j, bypassing index $[i..j-1]$ as they all overlap with the interval of $internship[i]$, again subproblem $maxExp(j)$ must also be optimal (same cut-and-paste argument).

Now this j can be computed slowly in $O(n)$, by iterating through the *internships* array and find the earliest internship j that starts right after $internship[i]$ ends at $internship[i][1]$ ends (just ensure that floating-point are handled correctly - this small detail is not important during quick grading). But if student answers this version, the time complexity analysis will be: n distinct states, each state computed in $O(n)$ time, so overall $O(n^2)$ DP solution.

For full marks, notice that this j can be computed via a classic Divide and Conquer algorithm: Binary Search (Lecture 03+04), as the *internships* array is sorted by $from_i$ field (again, floating-point detail is not important during quick grading). If student uses this, the time complexity analysis will be: n distinct states, each state computed in $O(\log n)$ time, so overall $O(n \log n) \in o(n^2)$ DP solution.

Prof Halim expects a big fraction of students insist to extend from the greedy algorithm mentioned in Section B.1.1. Unfortunately all such attempts will be judged as wrong answer as weighted interval scheduling problem is equivalent to finding a maximum independent set (a known NP-hard optimization problem), but in an interval graph (hence there is that DP algorithm above).

PS1: Prof Halim has a few other variant for this weighted interval scheduling problem. What if there is a limit $1 \leq k \leq n$ of the number of internships that you can take? What if $k = 2$? What if k a small constant? What if k can be up to n?, etc. These are reserved for future CS3230 PA2 tasks.

PS2: You can check whether your (alternative) solution(s) - if not as described above is correct or not by coding and testing directly at LeetCode 1235 - maximum-profit-in-job-scheduling.

Grading remarks: The average score for all 378 students is 3.38 + 1.53 + 1.61 = 6.52/16.0.

B.2 Fibonacci Counters (16 marks)

Recall that the Fibonacci numbers are defined as follows:

$$F(i) = \begin{cases} 1, & i \in \{1, 2\}, \\ F(i-1) + F(i-2), & i \geq 3. \end{cases}$$

We represent an integer k by an array of bits A where $A[i]$ *is the coefficient of* $F(i)$. Thus $k = \sum_{i=1} A[i] \cdot F(i)$. For example, we can represent 10 as the array [00111], since $10 = 2 + 3 + 5 = F(3) + F(4) + F(5)$, or alternatively as [11011], since $10 = 1 + 1 + 3 + 5 = F(1) + F(2) + F(4) + F(5)$.

Fibonacci Counter: Let us consider a variant of the binary counter that uses Fibonacci numbers as its base.

- Increment(A)

  1. Let i^* be the smallest index such that $A[i^*] = 0$ (if i^* exceeds the current length, treat missing entries as 0 and extend the array as needed).
  2. If $i^* \in \{1, 2\}$, set $A[i^*] \leftarrow 1$.
  3. Otherwise, overwrite the subarray $A[2..i^*]$ with an alternating pattern that *ends with a 1 at position* i^*. Formally, for each $j \in \{2, \ldots, i^*\}$ set

$$A[j] \leftarrow \begin{cases} 1, & \text{if } (i^* - j) \text{ is even,} \\ 0, & \text{if } (i^* - j) \text{ is odd.} \end{cases}$$

     (Equivalently, $A[i^*] \leftarrow 1$, $A[i^* - 1] \leftarrow 0$, $A[i^* - 2] \leftarrow 1$, $A[i^* - 3] \leftarrow 0$, ..., continuing down to index 2.)

Example 1: Let $A = [101]$, representing $F(1) + F(3) = 1 + 2 = 3$. The smallest index with $A[i^*] = 0$ is $i^* = 2$. After Increment(A), we obtain $A = [111]$, which represents $F(1) + F(2) + F(3) = 1 + 1 + 2 = 4$.

Example 2: Let $A = [11111]$, representing $F(1) + F(2) + F(3) + F(4) + F(5) = 1 + 1 + 2 + 3 + 5 = 12$. The smallest index with $A[i^*] = 0$ is $i^* = 6$. After Increment(A), we overwrite $A[2..6]$ by the rule above, yielding $A = [110101]$, which represents $F(1) + F(2) + F(4) + F(6) = 1 + 1 + 3 + 8 = 13$.

Example 3: Let $A = [1111001]$, representing $F(1)+F(2)+F(3)+F(4)+F(7) = 1+1+2+3+13 = 20$. The smallest index with $A[i^*] = 0$ is $i^* = 5$. After Increment(A), we obtain $A = [1010101]$, representing $F(1) + F(3) + F(5) + F(7) = 1 + 2 + 5 + 13 = 21$.

As with the binary counter, the *cost* of an increment is the number of bit flips. In the above examples, the respective costs are 1, 3, and 3.

B.2.1 Testing Your Understanding (3 marks)

Answer the following questions for $A = [11001]$:

1. What is the integer that A represents? (1 mark)
2. What is the array resulting from applying Increment(A)? (1 mark)
3. What is the cost of Increment(A)? (1 mark)

No justification is required; write down only the final answers.
The answers are: (1) 7,
(2) [10101], and
(3) 2.

Explanations: Array $A = [11001]$ represents $F(1) + F(2) + F(5) = 1 + 1 + 5 = 7$.
The smallest index with $A[i^*] = 0$ is $i^* = 3$. After $\mathsf{Increment}(A)$, the array becomes $A = [10101]$, representing $F(1) + F(3) + F(5) = 1 + 2 + 5 = 8$.
The number of bit flips is 2 (flipping $i* = 3$ from 0 to 1 and $j = 2$ from 1 to 0.

Grading remarks from Allen Bi: B21 (7, [10101], 2,): 340/378, 308/378, 304/378 got correct for part 1,2,3 respectively.

On B21.2: Some students wrote an '8' for B21.2 (around 10 20 students). They misunderstood the question, thinking it was asking for the integer the array represents.

B.2.2 Amortized Analysis (7 marks)

Prove that the amortized cost per operation is $O(1)$ over a sequence of n increments starting from the all-zero array.
Each operation performs *exactly one* $0 \to 1$ flip (at i^*) and possibly several $1 \to 0$ flips (within $A[2..i^* - 1]$). We use the accounting method and charge a fictitious cost of two dollars to each increment:

- One dollar pays for the *current* $0 \to 1$ flip at i^*.
- The second dollar is stored with the new 1-bit at position i^*.

We maintain the invariant that the bank holds one dollar for every 1-bit; i.e., balance $= \sum_i A[i] \geq 0$ at all times. The base case holds for the all-zero array. During an increment, any bit that flips $1 \to 0$ uses its stored dollar to pay for its flip and removes one dollar from the bank, preserving the invariant. Thus the total amortized cost is at most two per operation, which is $O(1)$.

- Using the number of *leading* 1-bits (i.e., the index of the first zero minus 1), or any variant of this quantity, as the potential function or bank account balance is incorrect, since this quantity can increase significantly in a single operation (for example, 101111111 $\to$ 111111111).
- Charging a constant cost per operation (e.g., 2 or 3) followed by an incorrect or vague analysis, without clearly specifying a valid potential function or bank account balance, is most likely considered incorrect.
- Stating that a constant amount is charged per operation (e.g., 2 or 3) where constant amount (e.g., 1 or 2) is deposited into the bank account most likely qualifies for partial marks. However, if the constants are clearly invalid (for example, storing \$5 per increment while withdrawing \$3 from the bank for each bit flip, both $0 \to 1$ and $1 \to 0$, which fails since $3 + 3 > 5$), then no marks will be given.

- Stating a potential function or bank account balance that is linear in the number of 1-bits also qualifies for partial marks.
- Storing money in the bank account *only* when $i^* \in \{1, 2\}$ may or may not yield a valid proof; however, doing so without a rigorous analysis is generally considered incorrect.
- Although the same accounting or potential arguments used for the binary counter is applicable here, the Fibonacci counter still differs from the binary counter. Simply claiming that the two data structure are essentially identical without a proper explanation does not constitute a valid proof.
- It is incorrect to say that bit flips from $1 \to 0$ occur only when the array size is expanded.
- Giving examples (e.g., verifying correctness only for the first few iterations) does not constitute a proof.
- While it might be possible to use the aggregation method, the analysis is not as straightforward as in the binary counter case.
- The following informal argument is generally considered incorrect: claiming that before any expensive operation of cost k, there must have been (linear in) k preceding operations and therefore storing a constant amount per operation is sufficient. A more precise justification is required, such as explicitly associating one dollar with each 1-bit.

B.2.3 Alternative Algorithm (6 marks)

Since some integers can be represented in multiple ways using Fibonacci numbers, alternative implementations of Increment(A) are possible. We say that an array A is *good* if it contains no consecutive ones and satisfies $A[1] = 0$. Design a variant of Increment(A) that ensures all representations remain good while maintaining a constant amortized cost.

Correctness: If A is good and represents an integer k, then the array A' produced by Increment(A) is also good and represents the integer $k + 1$.

Amortized Cost: The amortized cost per increment over a sequence of n operations, starting from the all-zero array, is $O(1)$.

For example, under this rule, the number 12 cannot be represented as [11111]; instead, it must be represented as [010101]. You may assume that whenever Increment(A) is applied, the input array A is guaranteed to be good.

To receive full marks, provide both a correct algorithm and an amortized analysis. A formal proof of correctness is not required.

The new algorithm is as follows.

1. Let $i^* \geq 2$ be the smallest index such that $A[i^*] = A[i^* + 1] = 0$, where any $A[i]$ with i exceeding the current array length is treated as 0.

2. Overwrite the subarray $A[2..i^*]$ by $[0 \cdots 01]$ (only the last bit is 1).

The amortized analysis is exactly the same as before, as the new algorithm still has the property that each increment performs exactly one $0 \to 1$ flip and possibly several $1 \to 0$ flips.

- Combining the algorithm for Problem B22 with a post-processing step that repeatedly applies the transformation $110 \to 001$ is considered a correct algorithm. It is important to note that newly created 1 may lead to the formation of another occurrence of the pattern 110, requiring additional transformations.
- An alternative approach is to first increment $A[2]$ (or $A[1]$ in case $A[2] = 1$), and then perform a single left-to-right sweep through the array, replacing every occurrence of 110 by 001 whenever it is encountered.
- An alternative way to do the amortized analysis is to observe that the transformation $110 \to 001$ reduces the total number of 1s by 1, so regardless of how many times we do this, we have enough money in the bank account to pay, if we store 3 dollars for each 1-bit.

Grading remarks: The average score for all 378 students is 2.52 + 2.49 + 1.11 = 6.12/16.0.

B.3 Reductions and NP-Hardness (16 marks)

Note: Each of the sub-sections in this problem can be solved independently of the others.

Consider the following two decision problems.

Subset Sum (SS):

- *Instance:* An array A of n *positive* integers $A[1], A[2], \ldots, A[n]$; and a target sum $T \in \mathbb{N}$
- *Question:* Is there a subset $S \subseteq \{1, 2, \ldots, n\}$ such that $\sum_{i \in S} A[i] = T$?

Coin Change (CC):

- *Instance:* An array D of m *positive* integers $D[1], D[2], \ldots, D[m]$; a target sum $R \in \mathbb{N}$; and a target length $K \in \mathbb{N}$
- *Question:* Is there a sequence $d[1], d[2], \ldots, d[K]$ such that each $d[j]$ is equal to $D[i]$ for some i, and $\sum_j d[j] = R$?

In other words, the Coin Change problem asks whether it is possible to make change for the value R using *exactly* K coins, each of which has one of the denominations in the array D, with an infinite supply of each type of coin. Note that here we ask for the change to have exactly K coins as opposed to at most K coins, which is the more common variant.

B.3.1 Testing Your Understanding (3 marks)

Solve each of the instances below, providing YES/NO as your answer.

1. Subset Sum instance: $A = [1, 5, 6, 2, 2]$, $T = 10$ YES. $S = \{1, 2, 4, 5\}$
2. Coin Change instance: $D = [1, 2, 3]$, $R = 10$, $K = 5$ YES. $3 + 3 + 2 + 1 + 1 = 10$
3. Coin Change instance: $D = [2, 4, 6, 8]$, $R = 25$, $K = 8$ NO. All denominations are even but target is odd.

Grading remarks from Allen Bi: B31 (Y/Y/N): 364/378, 363/378, 360/378 got correct for part 1,2,3 respectively.

B.3.2 Reducing SS to CC (7 marks)

Present a polynomial-time Karp reduction from the Subset Sum problem to the Coin Change problem. You need to clearly describe the reduction algorithm, and show that it maps YES instances to YES instances, and NO instances to NO instances.

Hint: In both SS and CC, given an array and a target, the task is to pick a certain number of copies of each element in the array such that their sum is equal to the target. The major difference is that in SS you are only allowed to pick either 0 or 1 copy of each element, whereas in CC there is no such restriction. However, in CC there is still the restriction that the total number of copies of all elements is K. One might try to somehow set up the numbers and target in the instance of CC constructed by the reduction to be such that the SS restriction is somehow indirectly imposed.

Given an instance (A, T) of the Subset Sum problem, the reduction constructs an instance (D, R, K) of Coin Change as follows. First, pick a base $b = \max(n+1, T+1)$. The array D is of length $m = 2n$, and is defined as follows for $i \in \{1, \ldots, n\}$:

$$D[2i-1] = b^i$$
$$D[2i] = A[i] \cdot b^{n+1} + b^i$$

Set the target sum $R = T \cdot b^{n+1} + \sum_{i=1}^{n} b^i$, and target length $K = n$. This reduction clearly runs in polynomial time in the input length.

Suppose (A, T) is a YES instance of Subset Sum. This means that there is a set $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} A[i] = T$. Then, we can obtain the value R by considering the following sequence d for $i \in [n]$:

$$d[i] = \begin{cases} D[2i-1] & \text{if } i \notin S \\ D[2i] & \text{if } i \in S \end{cases}$$

Note that d is of length $K = n$ as required. We have:

$$\begin{aligned}\sum_{i=1}^{n} d[i] &= \sum_{i \in S} D[2i] + \sum_{i \notin S} D[2i-1] \\ &= \sum_{i \in S} (A[i] \cdot b^{n+1} + b^i) + \sum_{i \notin S} b^i \\ &= \sum_{i \in S} A[i] \cdot b^{n+1} + \sum_{i=1}^{n} b^i \\ &= T \cdot b^{n+1} + \sum_{i=1}^{n} b^i = R\end{aligned}$$

So the instance (D, R, K) is a YES instance of Coin Change.

Next, suppose that the produced instance (D, R, K) is a YES instance of Coin Change. This means that there is a sequence d of length $K = n$ composed of values from D such that $\sum_i d[i] = R = T \cdot b^{n+1} + \sum_{i=1}^{n} b^i$. Because $b \geq n + 1$, for any i, we have that $n \cdot b^i < b^{i+1}$. So the only way for the term $\sum_{i=1}^{n} b^i$ to show up in the sum of a sequence of n elements of D is for the sequence to contain exactly one element from each pair $(D[2i-1], D[2i])$.

Without loss of generality, say that the sequence d is such that for each i, $d[i]$ is either $D[2i-1]$ or $D[2i]$. We can construct the required solution S to the Subset Sum problem by adding i to S if and only if $d[i] = D[2i]$. Then, we have:

$$\begin{aligned}\sum_{i=1}^{n} d[i] &= \sum_{i \in S} D[2i] + \sum_{i \notin S} D[2i-1] \\ &= \sum_{i \in S} A[i] \cdot b^{n+1} + \sum_{i=1}^{n} b^i\end{aligned}$$

By our assumption about d, we have that the above sum is equal to $R = T \cdot b^{n+1} + \sum_{i=1}^{n} b^i$. This implies that $\sum_{i \in S} A[i] = T$, as required, and thus the instance of Subset Sum we started with must have been a YES instance.

We have proven that the reduction produced a YES instance of Coin Change if and only if it was given a YES instance of Subset Sum. This proves that the reduction is a valid Karp reduction.
Common mistakes:

1. Target is embedded into least significant bits with insufficient padding, as a result of which sums can overflow into higher order bits used to control size of coin change
2. Setting the entries of D to be $b + A[i]$ rather than something of the form $A[i] \cdot b^{n+1}$

B.3.3 Algorithm for CC (4 marks)

Construct an algorithm for the Coin Change problem that runs in time $O(mKR)$.

Your answer must contain *clear pseudocode* for your algorithm, along with an explanation for why its running time is as required above. If either of these is missing, it will be considered incorrect.

The algorithm will use Dynamic Programming, and work as follows:

1. Initialise Boolean array $DP[i][j]$ to $false$, where $i \in \{0, \ldots, K\}$ and $j \in \{0, \ldots, R\}$
2. Set $DP[0][0]$ to $true$
3. For i from 1 to K:
   - For j from 1 to R:
     - For ℓ from 1 to m:
       - If $j \geq D[\ell]$ and $DP[i-1][j-D[\ell]] = true$, set $DP[i][j]$ to $true$
4. Return $DP[K][R]$

The algorithm has three nested loops, with K, R, and m iterations respectively, with $O(1)$ in each iteration. So its complexity is $O(mKR)$, as required.
Common mistakes:

1. Algorithm runs in $\Theta(m^2KR)$ time rather than $O(mKR)$, due to additional, avoidable, loop
2. In the alternate DP solution based on using the first i coins, missing the case where the i^{th} coin is not used

B.3.4 NP-Hardness (2 marks)

It is well-known that the Subset Sum problem is NP-hard, and the reduction described in Section B.3.2 implies that the Coin Change problem is NP-hard as well. Because of this, we believe that there is no polynomial-time algorithm for these problems. Explain briefly why the existence of the algorithm described in Section B.3.3 does not contradict this belief.

There is no contradiction because the algorithm asked for in Section B.3.3 is only pseudo-polynomial time and not polynomial time. In fact, we already know from the lectures, that Subset Sum has a pseudo-polynomial time algorithm.

Grading remarks from Allen Bi: Most students leave it blank.

I give full mark if I see 'pseudo polynomial' or if student justify that 'R grows exponentially w.r.t input size'.

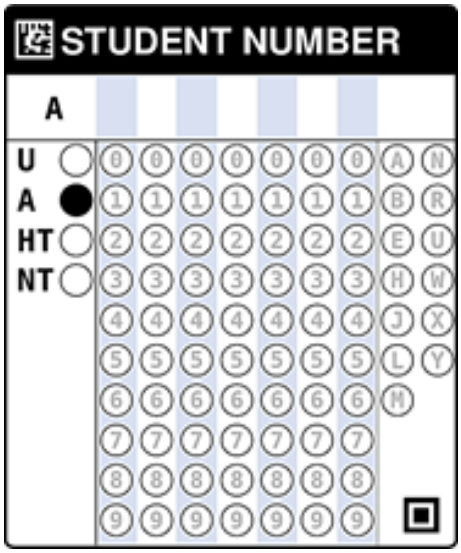
A few students (less than 10) mention the algorithm is not polynomial, but not explicitly write down 'w.r.t input size' neither 'exponentially', this does not convince me that they understand the exponential relationship between input and R; or that algorithm complexity is based on input size, so I (Allen) decide to gave 1 mark.

0 marks for answers that are totally nonsense.

Grading remarks: The average score for all 378 students is 2.86 + 1.10 + 1.34 + 1.16 = 6.46/16.0.

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	11	12	13
A	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
B	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
D	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
E	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

No	14	15	16	17	18	19	20	21	22	23	24	25	26
A	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
B	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
D	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
E	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

Box B.1.1. All Seven Point Sevens

Grading scheme: Your answer will be graded using the special+ marking scheme below.

Your answer	Mark(s)
Has two, three, or all four part(s) that are wrong	0
Blank	1
+Has at most one part that is slightly incorrect	4
Has all four components correct	7

Box B.1.2. Testing Your Understanding (for each question: 1/0/0 mark for correct/blank/wrong):

$n = 5$ and $internships = [[1.9, 6.1, 7.7], [1.1, 3.5, 7.7], [3.9, 6.0, 7.7], [3.5, 10, 7.7], [6.0, 9.57, 7.7]]$.

$n = 5$ and $internships = [[1.9, 6.1, 177.0], [1.1, 3.5, 20.1], [3.9, 6.0, 70.7], [3.5, 10, 157.6], [6.0, 9.57, 86.6]]$.

Box B.1.3. Not All Internships Are Equal

Grading scheme: Your answer will be graded using the special+ marking scheme below.

Your answer	Mark(s)
Incorrect	0
Blank	1
+Correct, but in $O(n^2)$ — not the optimal solution	4
Correct and $\in o(n^2)$	7

Box B.2.1. Testing Your Understanding (for each question: 1/0/0 mark for correct/blank/wrong):

What is the integer that A represents?

What is the array resulting from applying Increment(A)?

What is the cost of Increment(A)?

Box B.2.2. Amortized Analysis

Grading scheme: Your answer will be graded using the special+ marking scheme below.

Your answer	Mark(s)
Incorrect	0
Blank	1
+Incorrect, with a correct potential function or bank account balance clearly stated	3
Correct	7

Box B.2.3. Alternative Algorithm

Grading scheme: Your answer will be graded using the special marking scheme below.

Your answer	Mark(s)
Incorrect algorithm	0
Blank	1
Correct algorithm + incorrect amortized analysis	3
Correct algorithm + correct amortized analysis	6

Box B.3.1. Testing Your Understanding (for each question: 1/0/0 mark for correct/blank/wrong):

Subset Sum instance: $A = [1, 5, 6, 2, 2]$, $T = 10$

Coin Change instance: $D = [1, 2, 3]$, $R = 10$, $K = 5$

Coin Change instance: $D = [2, 4, 6, 8]$, $R = 25$, $K = 8$

Box B.3.2. Reducing SS to CC

Grading scheme: Your answer will be graded using the special marking scheme below.

Your answer	Mark(s)
Incorrect reduction	0
Blank	1
Correct reduction + incorrect proof in both directions	3
Correct reduction + incorrect proof in one direction	5
Correct algorithm + correct proof	7

Box B.3.3. Algorithm for CC

Grading scheme: Your answer will be graded using the special marking scheme below.

Your answer	Mark(s)
Incorrect/unclear algorithm	0
Blank	1
Partially correct algorithm + analysis, with significant errors	2
Correct algorithm + analysis	4

Box B.3.4. NP-Hardness

Grading scheme: Your answer will be graded using the special marking scheme below.

Your answer	Mark(s)
Incorrect explanation	0
Blank	1
Correct explanation	2

If you need extra working space, you can use the space below, but indicate clearly in the respective box.

– END OF PAPER; All the Best –