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)

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.

   b). There are infinitely many positive integers n such that $f(n) < 0.001 \cdot g(n)$.

   c). There exists a positive integer n such that $f(n) < (g(n))^{0.99}$.

   d). $g(n) \in o(f(n))$.

   e). None of the above.

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

   a) $\log(n!)$

   b) $2^{\log^2 n}$

   c) n^2

   d) $(\log n)^{\log n}$

   e) $n \log n$

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

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

   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.

   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)`.

   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 $m = |E|$.

   d). The total number of edges relaxed so far is $O(m)$ and the total number of vertices inside set R is $O(n)$.

   e). The set of edges that cross the partition $(R, V \setminus R)$ is minimized, ensuring the greedy choice remains optimal.

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.

   b). 2-i.

   c). 2-ii.

   d). 2-iii.

   e). 2-iv.

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 $A_1 = A[0 \ldots \lfloor k/3 \rfloor - 1]$, $A_2 = A[\lfloor k/3 \rfloor \ldots \lfloor 2k/3 \rfloor - 1]$, $A_3 = A[\lfloor 2k/3 \rfloor \ldots (k-1)]$, // 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) $\Theta(n)$

   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 $(a_1, a_2, \ldots, a_n)$ 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 $i \in \{1, 2, \ldots, n\}$ such that $Y_i \in o(\log n)$.

   b). There exists $i \in \{1, 2, \ldots, n\}$ 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.

9. Consider random numbers $a_1, a_2, \ldots, a_n$, where each a_i is drawn independently at random from $\{1, 2, \ldots, n\}$. 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) $\Theta(n^2)$

   (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**, <u>0</u>, and 00). Note that if the same sequence (e.g. "0") appears more than once as a subsequence (as the **second** and <u>third</u> 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) $C[i][j] = C[i][j-1] + C[i+1][j] + 1$
    (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 $O(n)$
    b). $F(i)$ runs very slowly as there are lots of cache miss, we must set `maxsize = n` (or `None`)
    c). $F(i)$ becomes wrong
    d). $F(i)$ causes runtime error
    e). None of the above

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)`

b). `@lru_cache(maxsize = n)`

c). `@lru_cache(maxsize = max(m, n))`

d). `@lru_cache(maxsize = min(m, n))`

e). None of the above

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

b) There is an optimal solution that contains the last item

c) There is an optimal solution that contains the item with value-to-weight ratio closest to 1

d) There is a greedy-choice property, but it is not any of the above

e) There is no greedy-choice property as the problem does not have a correct 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 $O(1)$; tree height is $\Theta(n)$.

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

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

d). ABL is $O(\log n)$; 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). $O(n^{0.7})$.

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 $2n$

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) $O(2^n)$
    b) $2^{O(n)}$
    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.

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 <u>*not*</u> 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
    e) None of the above

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?
    e). None of the above.

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 $v \in D$.

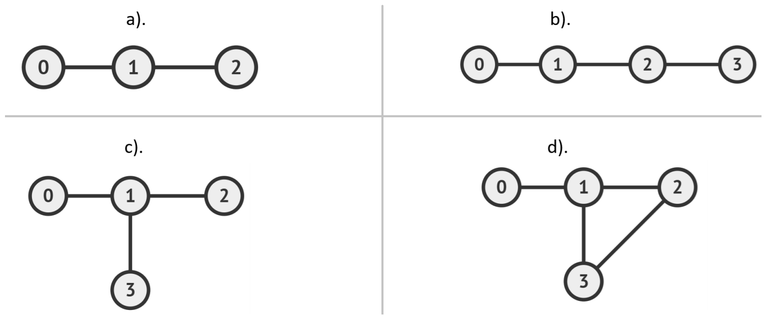
    b). For any leaf v in T, there exists a minimum dominating set D such that $u \in D$, where u is the unique neighbor of v.

    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.

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). $\frac{11}{15}n$.

    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.

    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.

    c). $O(n)$ selection algorithm only achievable in best-case scenario.

    d). We can reduce the sorting problem to selection problem to get an $O(n)$ solution.

    e). None of the above.

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.

    b). The saving is achieved by switching the data structure from an Adjacency List to an Edge List, simplifying memory access during relaxation.

    c). The total space required is $O(|V|)$ because the space requirement can be amortized.

    d). The shortest paths is at most $O(|V|)$ distinct edges due to the Optimal Substructure property, so only $O(|V|)$ space is needed.

    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.

    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.

    c). Prim's algorithm runs in $O(V^2)$ time if we use Binary Heap data structure.

    d). There is no other algorithm to solve the MST problem other than Prim's algorithm.

    e). None of the above.

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 ($1.0 \le from_i < to_i \le 10^9$) — the start and end times of internship i (in arbitrary time units), and
- exp_i ($1.0 \le exp_i \le 10^9$) — 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 $from_j + 0.0001 \ge to_i$, 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 $3.9999 + 0.0001 \ge 4.0$ — 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.

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

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 $o(n^2)$.

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, $A[i^*] \leftarrow 1$, $A[i^* - 1] \leftarrow 0$, $A[i^* - 2] \leftarrow 1$, $A[i^* - 3] \leftarrow 0$, $\ldots$, 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 $\mathsf{Increment}(A)$? (1 mark)
3. What is the cost of $\mathsf{Increment}(A)$? (1 mark)

No justification is required; write down only the final answers.

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.

B.2.3 Alternative Algorithm (6 marks)

Since some integers can be represented in multiple ways using Fibonacci numbers, alternative implementations of $\mathsf{Increment}(A)$ 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 $\mathsf{Increment}(A)$ 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 $\mathsf{Increment}(A)$ 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 $\mathsf{Increment}(A)$ 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.

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: $A = [1, 5, 6, 2, 2]$, $T = 10$
2. Coin Change instance: $D = [1, 2, 3]$, $R = 10$, $K = 5$
3. Coin Change instance: $D = [2, 4, 6, 8]$, $R = 25$, $K = 8$

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.

B.3.3 Algorithm for CC (4 marks)

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

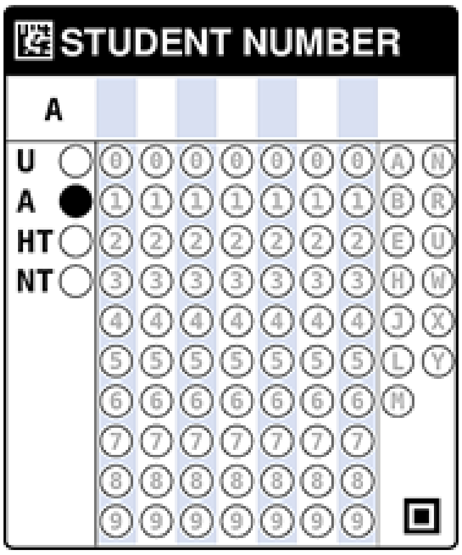
Your answer must contain *clear pseudocode* for your algorithm, along with an explanation for why its running time is as required above. If either of these is missing, it will be considered incorrect.

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.

The Answer Sheet

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



STUDENT NUMBER

A

U ○	0	0	0	0	0	0	0	A	N
A ●	1	1	1	1	1	1	1	B	R
HT ○	2	2	2	2	2	2	2	E	U
NT ○	3	3	3	3	3	3	3	H	W
	4	4	4	4	4	4	4	J	X
	5	5	5	5	5	5	5	L	Y
	6	6	6	6	6	6	6	M	
	7	7	7	7	7	7	7		
	8	8	8	8	8	8	8		
	9	9	9	9	9	9	9		

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 –