National University of Singapore

School of Computing

CS3230 - Design and Analysis of Algorithms

Midterm Test

(Semester 1 AY2025/26)

Time Allowed: 120 minutes

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains TWO (2) sections.
   It comprises NINETEEN (19) printed pages, including this page.
3. This is an **Open Book** Assessment.
   You cannot use any electronic device except one non-programmable calculator.
4. For Section A, use the boxes on page 13 (use 2B pencil).
   For Section B, answer **ALL** questions within the **boxed space**.
   If you leave the boxed space blank, you will get automatic free 0.5 mark for that box.
   However, if you write at least a single character and it is totally wrong, you will get 0 mark.
   Canceling your (incomplete) answer can be done by crossing the entire box with a big **X**.
   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 60 marks.
   It will then be scaled to 30% of the course weightage.

This page is intentionally left blank

A MCQs ($20 \times 1.5 = 30$ marks)

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

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

   b). $2^n \cdot \sqrt{n} \in \Theta(2^n)$

   c). $(n+1)^n \in \Theta(n^n)$

   d). $101^{\log \log n} \in \Theta(100^{\log \log n})$

   e). None of the above

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

   a) $n \log n$

   b) $2^{\log n}$

   c) $n^{\log n}$

   d) $(\log n)^n$

   e) 2^n

3. You are asked to solve a difficult computation problem in $o(n \log^2 n)$.
   You have been shown a proof that the lower bound to solve this problem is $\Omega(n \log^{\frac{2}{3}} n)$.
   Which target time complexity should we should aim for?

   a). $\Theta(n \log^{\frac{1}{2}} n)$

   b). $O(n \log^2 n)$

   c). $O(\log n)$

   d). $O(n \log^{\frac{7}{4}} n)$

   e). $\Theta(n^2)$

4. What is the *best (clearest)* way to show that $n \log n \in o(n \log^2 n)$?

   a). For *any* constant $c > 0$ and $n_0 = 7$, for all $n \geq n_0 : 0 \leq n \log n < c \cdot n \log^2 n$

   b). $\lim_{n \to \infty} \frac{n \log n}{n \log^2 n} = 7$

   c). $\lim_{n \to \infty} \frac{n \log^2 n}{n \log n} = \infty$

   d). For some $k \geq 0$, $n \log n \in \Theta(n \log^{k=1} n)$, so $n \log n \in o(n \log^{k=2} n)$

   e). None of the above

5. Consider the recurrence relation: $T(n) = T(\sqrt{n}) + \sqrt{\log n}$.
   Which of the following statements is **TRUE**?

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

   b) $T(n) \in \Theta(\sqrt{\log n})$

   c) $T(n) \in \Theta(\sqrt{\log n \log \log n})$

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

   e) None of the above

6. Which of the following is a solution to the recurrence $T(n) = 9 \cdot T(n/3) + n^3$?

   a) $\Theta(n^2)$

   b) $\Theta(n^2 \log n)$

   c) $\Theta(n^3)$

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

   e) None of the above

7. Consider the following algorithm:
   ALG1($A[1 \dots n]$):

   - For $i = 1$ to $n - 1$:
     - For $j = 1$ to $i - 1$:
       - If $A[j] > A[i]$: swap the values of $A[j]$ and $A[i]$

   Which of the following is a loop invariant that is true at the start of iteration i of the outer loop?

   a) A is sorted

   b) $A[1 \dots (i-1)]$ is sorted

   c) ($A[1 \dots (i-1)]$ is sorted) and ($x \leq A[i]$ for all $x \in A[1 \dots (i-1)]$)

   d) ($A[1 \dots (i-1)]$ is sorted) and ($x \leq y$ for all $x \in A[1 \dots (i-1)]$ and $y \in A[i \dots n]$)

   e) None of the above

8. You are given the Single-Source Shortest Paths (SSSP) problem on a directed weighted graph $G = (V, E)$ where $n = |V|$ and $m = |E|$.
   You are told that G is a *planar graph* where $m \in O(n)$.
   Which algorithm among the five options below has the fastest asymptotic running time?

   a). Dijkstra's algorithm with Binary heap

   b). Dijkstra's algorithm with Fibonacci heap

   c). Use the Duan et al. algorithm (STOC 2025) with time complexity $O(m \log^{\frac{2}{3}} n)$

   d). BFS

   e). DFS

9. Recall that BINARYSEARCH$(A, \text{lb}, \text{ub}, x)$ works as follows.

   - If lb $>$ ub, return **NO**.
   - Else
     - mid $\leftarrow \lfloor(\text{lb}+\text{ub})/2\rfloor$.
     - If $x = A[\text{mid}]$, return **YES**.
     - If $x > A[\text{mid}]$, return BINARYSEARCH$(A, \text{mid}+1, \text{ub}, x)$.
     - If $x < A[\text{mid}]$, return BINARYSEARCH$(A, \text{lb}, \text{mid}-1, x)$.

   If we replace mid $\leftarrow \lfloor(\text{lb}+\text{ub})/2\rfloor$ with mid $\leftarrow \lfloor\text{lb}+\sqrt{\text{ub}-\text{lb}}\rfloor$, then what is the resulting worst-case time complexity for searching in a sorted array of size n?

   a). $\Theta(1)$
   b). $\Theta(\log n)$
   c). $\Theta(\sqrt{n})$
   d). $\Theta(n)$
   e). None of the above

10. Consider the problem of finding the maximum sum of any sub-array of a given array of integers of size n. (A sub-array is a set of contiguous elements of the given array.) One approach to do this is to divide the array into two equal halves, find the solution for each of them, and then compare these to the maximum sum of a sub-array that starts in the first half and ends in the second half. Assuming the last part is computed with a linear-time algorithm, what is the complexity of the entire algorithm?

    a) $\Theta(n)$
    b) $\Theta(n \log n)$
    c) $\Theta(n^2)$
    d) $\Theta(n^2 \log n)$
    e) None of the above

11. If Quicksort always chooses the first element as the pivot and the input array is already sorted, then what is the number of comparisons?

    a). $\Theta(n)$
    b). $\Theta(n \log n)$
    c). $\Theta(n^2)$
    d). $\Theta(n^2 \log n)$
    e). None of the above

12. Which of the five events below has the *highest* probability of happening?

    a). Randomly picking an integer from set $\{1, 2, 3, 4, 5, 6, 7\}$ and it is odd

    b). Randomly picking an integer from set $\{2, 3, 4, 5\}$ and it is a prime

    c). Randomly picking an element from an array of size n that contains a majority element appearing at most $\lfloor \frac{3 \cdot n}{5} \rfloor$ times, and the picked element is the majority element

    d). Randomly picking $x \in [-1.0..1.0]$ and $y \in [-1.0..1.0]$ and found that (x, y) is inside a circle centered at $(0, 0)$ with radius $r = 1.0$

    e). Randomly picking $x \in [0.0..1.0]$ and $x \geq 0.3$

13. Consider the following algorithm:
    ALG2(a, b):

    - Set $s = 0$
    - For $i = a$ to b: $s = s + i$
    - Output s

    Determine the average-case complexity of the above algorithm with its inputs a and b are chosen in the following manner: a is chosen to be a random number from $\{1, 2, \ldots, n\}$, and then b is chosen to be a random number from $\{a, a + 1, \ldots, n\}$.

    a) $\Theta(n)$

    b) $\Theta(n \log n)$

    c) $\Theta(n^2)$

    d) $\Theta(n^2 \log n)$

    e) None of the above

14. Partition the vertex set of a graph $G = (V, E)$ (without self-loops) into two parts $V = V_1 \cup V_2$ randomly as follows.

    - Each vertex $v \in V$ flips a coin independently.
      - If the outcome is head, v joins V_1.
      - If the outcome is tail, v joins V_2.

    Suppose the coin is biased in such a way that the the outcome is head with probability $2/3$ and is tail with probability $1/3$. What is the expected number of edges crossing V_1 and V_2?

    a). $(3/9)|E|$

    b). $(4/9)|E|$

    c). $(5/9)|E|$

    d). $(6/9)|E|$

    e). None of the above

15. Consider the following randomly generated graph with n vertices $V = \{1, 2, \ldots, n\}$ (n is a multiple of 3): For each pair of vertices i and j (where $i \neq j$), the edge $\{i, j\}$ exists in the graph with probability 1/3. What is the expected number of undirected edges that go between vertices in the set $\{1, 2, \ldots, n/3\}$ and those in the set $\{n/3 + 1, n/3 + 2, \ldots, n\}$?

    (a) $2n^2/9$

    (b) $2n^2/27$

    (c) $n^2/9$

    (d) $n^2/27$

    (e) None of the above

16. You can climb a staircase with n steps by jumping either one or two steps at a time. Let $T(n)$ be the number of distinct ways to climb n steps. For example, $T(1) = 1$ (only one way: 1), $T(2) = 2$ (two ways: 1+1, 2), and $T(3) = 3$ (three ways: 1+1+1, 1+2, 2+1). What is the value of $T(10)$?

    a). 23

    b). 45

    c). 55

    d). 89

    e). None of the above

17. Suppose the correct 7-lines (Python) implementation shown in class for `https://leetcode.com/problems/longest-common-subsequence`:

```
class Solution:
  def longestCommonSubsequence(self, text1: str, text2: str) -> int:
    @cache
    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(len(text1)-1, len(text2)-1)
```

    is modified (only the last 3 lines are changed) into:

```
class Solution:
  def longestCommonSubsequence(self, text1: str, text2: str) -> int:
    @cache
    def LCS(i, j):
      if i == len(text1) or j == len(text2): 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(0, 0)
```

What will happen if this code is submitted to LeetCode?

a). It will get Time Limit Exceeded as its runtime becomes exponential

b). It will get Wrong Answer as the optimal substructure is wrong

c). It will get Run Time Error as the base cases are off by one

d). It will get Memory Limit Exceeded as the space complexity is too big

e). It will get Accepted

18. Which `LCS(A, B)` has the longest length?

a). `LCS('steven', 'seven')`

b). `LCS('awerczvxew', imijuiiynu')`

c). `LCS('stevenhalim', 'milahnevets')`

d). `LCS('gatagaca', 'gattaca')`

e). `LCS('abcdefghij', 'jihgfedcba')`

19. You have coins of denominations 1, 5, 10, and 20.

- Let $M(n)$ be the minimum number of coins to make up value n.
- Let $M^*(n)$ be the minimum number of coins to make up value n, under the constraint that you must use at least one coin of denomination 20.

Which of the following statements is **TRUE** for all $n > 20$?

a). $M^*(n) = 1 + \min\{M(n-1), M(n-5), M(n-10), M(n-20)\}$

b). $M^*(n) = 1 + \min\{M^*(n-1), M^*(n-5), M^*(n-10), M(n-20)\}$

c). $M^*(n) = 1 + \min\{M(n-1), M(n-5), M(n-10), M^*(n-20)\}$

d). $M^*(n) = 1 + \min\{M^*(n-1), M^*(n-5), M^*(n-10), M^*(n-20)\}$

e). None of the above

20. Suppose you wish to count the number of binary strings of length n that do not contain a consecutive pair of 1's (e.g., "1010" is counted, but not "0110"). For i in the range $\{0, 1, \ldots, n\}$ and b in $\{0, 1\}$, denote by $A[i][b]$ the number of strings of length i that satisfy this condition and whose last character is b. Which of the following is **TRUE**?

(a) $A[i][0] = A[i-1][1]$, and, $A[i][1] = A[i-1][0] + A[i-1][1]$

(b) $A[i][0] = A[i-1][0] + A[i-1][1] + 1$, and, $A[i][1] = A[i-2][0] + A[i-2][1]$

(c) $A[i][0] = A[i-1][0] + A[i-1][1]$, and, $A[i][1] = A[i-1][0]$

(d) $A[i][0] = A[i-1][1]$, and, $A[i][1] = A[i-1][0] + A[i-2][0]$

(e) None of the above

B Essays (30 marks)

B.1 Discrete Fourier Transform (10 marks)

(Note that in this problem, it is possible to solve parts (B.1.3) and (B.1.4) WITHOUT solving (B.1.1) and (B.1.2).)

Consider a natural number n that is a power of 2, and an array of numbers A of size n. The Discrete Fourier Transform (DFT) of A is another array $\hat{A}$, also of size n, whose entries are defined are follows for $k \in \{0, \ldots, n-1\}$:

$$\hat{A}[k] = \sum_{j=0}^{n-1} A[j] \cdot (e^{-\frac{2\pi i}{n}})^{k \cdot j}$$

Above, i is the imaginary square root of 1, and $e^{-\frac{2\pi i}{n}}$ is an n-th root of unity.
What these mean are not important for this problem, except for the following important properties:

$$(e^{-\frac{2\pi i}{n}})^{\frac{n}{2}} = e^{-\pi i} = -1 \qquad (e^{-\frac{2\pi i}{n}})^2 = e^{-\frac{2\pi i}{n/2}}$$

Note that $e^{-\frac{2\pi i}{n}}$ is used in the DFT definition above because n is the size of the array A.
If its size had been some other number m, we would have used $e^{-\frac{2\pi i}{m}}$ instead.

Assume for simplicity that any arithmetic operation such as addition, multiplication, *and especially* raising e to any complex power, etc., can be performed in $O(1)$ time. Then, the most straightforward algorithm that, given A, computes its DFT $\hat{A}$ one index at a time using the definition above takes times $\Theta(n^2)$.

There is, however, a faster Divide and Conquer algorithm to compute the DFT. Consider two arrays E and O, of size $n/2$ each, that consist of the entries at the even and odd locations in A, respectively. That is, for each $j \in \{0, \ldots, n/2-1\}$:

$$E[j] = A[2 \cdot j] \qquad O[j] = A[2 \cdot j + 1]$$

We then express $\hat{A}$ in terms of $\hat{E}$ and $\hat{O}$ (the DFT's of E and O).
Note that while $\hat{A}$ has size n, the DFT's $\hat{E}$ and $\hat{O}$ have size only $n/2$.

B.1.1 Prove Part 1 (2 marks)

For any $k \in \{0, \ldots, n/2-1\}$, we have the following relationship between $\hat{A}[k]$, $\hat{E}[k]$, and $\hat{O}[k]$:

$$\hat{A}[k] = \hat{E}[k] + (e^{-\frac{2\pi i}{n}})^k \cdot \hat{O}[k]$$

Complete the following proof of the above statement,
providing adequate reasoning for each step in your derivation:

$$\begin{aligned}\hat{A}[k] &= \sum_{j=0}^{n-1} A[j] \cdot (e^{-\frac{2\pi i}{n}})^{k \cdot j} \\ &= \sum_{j=0}^{n/2-1} A[2 \cdot j] \cdot (e^{-\frac{2\pi i}{n}})^{k \cdot 2 \cdot j} + \sum_{j=0}^{n/2-1} A[2 \cdot j + 1] \cdot (e^{-\frac{2\pi i}{n}})^{k \cdot (2 \cdot j + 1)} \\ &= \ldots \textit{(fill in the rest)} \ldots\end{aligned}$$

B.1.2 Prove Part 2 (2 marks)

For any $k \in \{0, \ldots, n/2 - 1\}$, we have the following relationship between $\hat{A}[k + n/2]$, $\hat{E}[k]$, and $\hat{O}[k]$:

$$\hat{A}[k + n/2] = \hat{E}[k] - (e^{-\frac{2\pi i}{n}})^k \cdot \hat{O}[k]$$

Complete the following proof of the above statement,
providing adequate reasoning for each step in your derivation:

$$\begin{aligned}\hat{A}[k + n/2] &= \sum_{j=0}^{n-1} A[j] \cdot (e^{-\frac{2\pi i}{n}})^{(k+n/2) \cdot j} \\ &= \sum_{j=0}^{n/2-1} A[2j] \cdot (e^{-\frac{2\pi i}{n}})^{(k+n/2) \cdot 2j} + \sum_{j=0}^{n/2-1} A[2j + 1] \cdot (e^{-\frac{2\pi i}{n}})^{(k+n/2) \cdot (2j+1)} \\ &= \sum_{j=0}^{n/2-1} A[2j] \cdot (e^{-\frac{2\pi i}{n}})^{k \cdot 2j} \cdot (e^{-\frac{2\pi i}{n}})^{\frac{n}{2} \cdot 2j} + \sum_{j=0}^{n/2-1} A[2j + 1] \cdot (e^{-\frac{2\pi i}{n}})^{k \cdot 2j} \cdot (e^{-\frac{2\pi i}{n}})^{k} \cdot (e^{-\frac{2\pi i}{n}})^{\frac{n}{2} \cdot (2j+1)} \\ &= \ldots \textit{(fill in the rest)} \ldots\end{aligned}$$

B.1.3 Divide and Conquer Algorithm (4 marks)

Construct a Divide and Conquer algorithm that uses the above observations to compute the DFT of a given array A of size n (that is a power of 2), and runs faster than $\Theta(n^2)$ (4 marks)
PS: You need to write the pseudocode for the algorithm.
You may use terms like $(e^{-\frac{2\pi i}{n}})^{a \cdot b}$ in your code *and assume* that these are computable in $O(1)$ time.

B.1.4 Running Time (2 marks)

Compute the running time of the algorithm you presented in subsection B.1.3 above!

B.2 Random Greedy Maximal Independent Set (10 marks)

Let $G = (V, E)$ be a *simple* undirected graph (no self-loops and no parallel edges). Consider the following randomized procedure. Choose a uniformly random permutation π of V. Process vertices in the order π, maintaining a set $S \subseteq V$ initialized to $\varnothing$ (empty set). When a vertex v is processed, add v to S if none of its already-processed neighbors was previously added to S. At the end, output S.

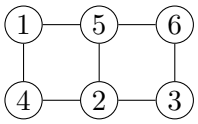


Figure 1: To see how the procedure works, let us look at a simple example of 2×3 grid graph. Each vertex is labeled by its rank in a randomly chosen permutation. In this run, the algorithm produces the set $S = \{1, 2, 6\}$. Of course, the outcome depends on the permutation: For instance, if the vertices are processed in the order $6 \to 5 \to 4 \to 3 \to 2 \to 1$, then the procedure instead outputs $S = \{4, 6\}$.

B.2.1 Complete Graphs (2 marks)

A *complete graph* K_n is an n-vertex simple graph in which each pair of vertices is connected by an edge. What is the size of S when $G = K_n$? You only need to provide the final answer; no proof or explanation is required.

B.2.2 Complete Bipartite Graphs (4 marks)

A *complete bipartite graph* $K_{a,b}$ is a simple graph whose vertex set is partitioned into two parts L and R with $|L| = a$ and $|R| = b$ such that there are no edges within L or within R, and every vertex of L is adjacent to every vertex of R. Prove that for $G = K_{a,b}$,

$$\mathbb{E}\big[\,|S|\,\big] \;=\; \frac{a^2 + b^2}{a + b}.$$

B.2.3 Lower Bound (4 marks)

Prove that the lower bound

$$\mathbb{E}\big[\,|S|\,\big] \;\geq\; \sum_{v \in V} \frac{1}{\deg(v) + 1}$$

holds for any graph, where $\deg(v)$ denotes the degree of v (i.e., its number of neighbors).

Figure 2: Examples of complete graphs and complete bipartite graphs.

B.3 Longest Contiguous Increasing Stock Prices (10 marks)

You are given three floating-point (`double` data type) arrays `price1`, `price2`, and `price3`, each of length n, representing the stock prices of your favorite blue-chip company based in Singapore (D05) over the last n days, as reported by three different market makers.

You want to analyze a trading strategy using this historical data. Your strategy depends on finding the length of the **longest contiguous strictly increasing stock prices**, assuming you can switch between market makers on any day at no cost.

More formally, define a "combined" ~~integer~~ array `price` of length n. For each index $i \in [0..n-1]$, you may assign `price[i]` to be *either* `price1[i]`, `price2[i]`, or `price3[i]`. Your goal is to find the length of the **longest contiguous non-empty subarray** of `price` such that the prices are strictly increasing.

An example is shown in Table 1.

	1	2	3	4	5	6	7	8	9
`price1`	50.75	**50.48**	50.47	50.52	**50.85**	**50.88**	50.80	51.39	51.53
`price2`	50.76	50.46	**50.54**	50.53	50.80	50.83	50.85	51.38	51.54
`price3`	50.77	50.49	50.81	**50.81**	50.79	50.81	**51.40**	51.37	51.55

Table 1: Example of the last $n = 9$ days.
All floating-point values start with 5 and a '.' in the middle, so focus on the last three digits.
The 6 days of contiguous strictly increasing D05 stock prices are **highlighted** (other solutions exist).

B.3.1 Subtask 1: One `price1` only; Design an $O(n \log n)$ algorithm (3 marks)

Suppose there is only **one** market maker that you trust, providing `price1` only. For example, see Table 2. Design an $O(n \log n)$ algorithm to solve Subtask 1.

	1	2	3	4	5	6	7	8	9
`price1`	50.75	50.48	**50.47**	**50.52**	**50.85**	**50.88**	50.80	51.39	51.53

Table 2: Example of the last $n = 9$ days (only `price1`).
The 4 days of contiguous strictly increasing D05 stock prices are **highlighted** (unique solution).

B.3.2 Subtask 2: Design an $o(n \log n)$ algorithm (5 marks)

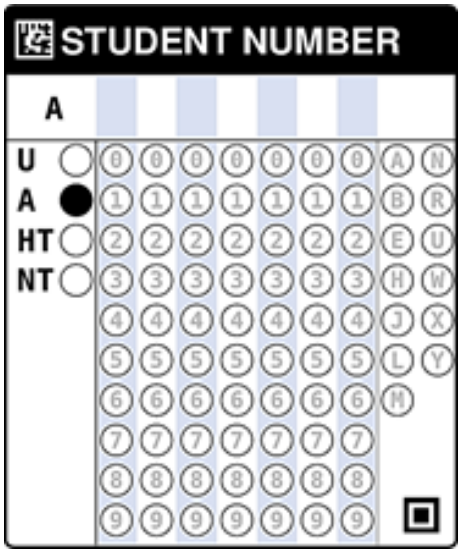
Design an algorithm to solve the general version of task B.3 (the one with *3 (THREE)* different prices for each given day). To be given marks, your algorithm must runs in little $o(n \log n)$ – supply a proof of correctness. You can use additional space (as much as you need, as long as it is reasonable).

B.3.3 Subtask 3: Solve Subtask 2 with only $\Theta(1)$ additional space (2 marks)

Now this is the hardest version of this task.
Please ensure that you are confident with your Subtask B.3.2 solution first.
For the last 2 marks, your correct algorithm must run in little $o(n \log n)$ and only use $\Theta(1)$ additional space (note that the $\Theta(n)$ input size to store the prices is *not* considered in the analysis).

The Answer Sheet

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



Fill in 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	16	17	18	19	20
A	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
B	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
C	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
D	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯
E	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

Box B.1.1. Prove Part 1

Box B.1.2. Prove Part 2

Box B.1.3. Divide and Conquer Algorithm

Box B.1.4. The Running Time of D&C Algorithm in B.1.3

Box B.2.1. Random Greedy Maximal Independent Set - Complete Graphs

Box B.2.2. Random Greedy Maximal Independent Set - Complete Bipartite Graphs

Box B.2.3. Random Greedy Maximal Independent Set - Lower Bound

Box B.3.1. Longest Contiguous Increasing Stock Prices - One price1 only, $O(n \log n)$ algorithm

Box B.3.2. Longest Contiguous Increasing Stock Prices - Design an $o(n \log n)$ algorithm

Box B.3.3. Longest Contiguous Increasing Stock Prices - only $\Theta(1)$ additional space

– END OF PAPER; All the Best –