National University of Singapore
School of Computing

# IT5003 - Data Structures and Algorithms
# Final Assessment

(Semester 2 AY2024/25)

Time Allowed: 2 hours

---

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper contains TWO (2) sections.
   It comprises TWELVE (12) printed pages, including this page.
3. This is an **Open Book Assessment**.
   You cannot use any electronic device except one non-programmable calculator.
4. 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 (sub-)questions might be easier than they appear.
6. You can use **pseudo-code** in your answer but beware of penalty marks for **ambiguous answer**.
   You can use **standard, non-modified** classic algorithm in your answer by just mentioning its name, e.g. run Inorder Traversal on BST $T$, BFS on graph $G$, Dijkstra's on graph $G'$, etc.
7. The total marks is 100. All the best :)

## A MCQs ($5 \times 2 = 10$ marks)

Select the **best unique** answer for each question.
Each correct answer worth 2 marks.

1. What is the *tightest worst-case* time complexity of the following Python code in terms of $n$? Notice the keyword 'tightest', i.e., if the best answer is $x$ but you choose an option that is worse than $x$ (which is still true in Big O notation, but not the tightest), you will be marked as wrong. Similarly if you choose an option that is better than $x$ (impossible), you will be marked as wrong.

```python
def foo(n): # what is the time complexity in terms of n?
    if n == 1:
        ans = 0
    elif n%2 == 0:
        ans = n//2 + foo(n//2)
    else:
        ans = (n-1)//2 + foo((n-1)//2+1)
    return ans

n = int(input()) # n can be any positive integer
print(foo(n))
```

   a). $O(1)$
   b). $O(\log n)$
   c). $O(n)$
   d). $O(n \log n)$
   e). $O(n^2)$

2. The implementation of function `def foo(n):` in Q1 can be simplified into:

   a). `return n-1`
   b). `return n**2`
   c). `return n*2`
   d). `return n`
   e). `return n//2`

3. You are given a Python list `L` containing $n$ Integers between -9 to 9.
   Your task is to find the maximum product (multiplication) of **three** Integers in `L`.
   For example, if $L1$ = [3, 2, 1, 3], the the answer is $3 \times 2 \times 3 = 18$,
   But if $L2$ = [-3, 2, -1, 3], the the answer is $-3 \times -1 \times 3 = 9$.
   What is the answer if $L$ = [7, -9, 7, -6, 1, -3, -4, -8, -9, -2, -6, 6, -1, 8, 4, -7, -4, 5, 1, 5]?
   PS: If you use one of the technique taught in IT5003, the computation is trivial.

a). 343

b). 392

c). 504

d). 576

e). 648

4. You are given a Python list `L` containing $2n$ positive Integers. You need to return `True` if these $2n$ Integers can be grouped into $n$ pairs of Integers so that both Integers in each pair is equal, or return `False` otherwise. Which solution is correct and fastest?

   a). `return all(nums.count(ni)%2 == 0 for ni in L)`

   b). `return all(v%2 == 0 for k, v in Counter(L).items())`

   c). `return 7 in L`

   d). `return sorted(L) == L`

   e). None of the above

5. You are given the following Python code. You are told that it correct, but *a bit slow*. Using what you have learned in IT5003, propose the **best time complexity improvement**. Note that you **must maintain the correctness** of this function and *len*(*arr*) > 100 000.

```
from bisect import bisect_left # you may want to use this ,  # L1
                                                             # L2
def bar(arr, k): # arr is a sorted list , k is an int        # L3
                                                             # L4
    for i in range(1, 2001):                                 # L5
                                                             # L6
        if not i in arr:                                     # L7
            k -= 1                                           # L8
            if k == 0:                                       # L9
                return i                                     # L10
    return None                                              # L11

print(bar([2,3,4,7,11], 5)) # will output 9
print(bar([1,2,3,4], 2)) # will output 6
```

   a). Delete line #L11

   b). Add `pos = bisect_left(arr, i)` in line #L6
   Then change line #L7 to `if pos >= len(arr) or arr[pos] != i:`

   c). Change line #L7 to `if not i in set(arr):`

   d). Add `arr = set(arr)` in line #L4

   e). None of the above

## B Application Questions ($6 \times 15 = 90$ marks)

### B.1 Deleting Characters (15 marks)

Given a string $S$ with $n$ lowercase/digit characters, please keep doing this operation repeatedly:

> Delete the **first digit** and the **closest non-digit character to its left** of $S$.

Output the resulting final string after you can no longer perform this operation.
It is guaranteed that it is possible to delete all digits in $S$ using just this operation.
Here are a few examples to help you understand this task:

- $S1$ = 'xyz', the output is also 'xyz', as we cannot perform the requested operation,
- $S2$ = 'steve~~n7~~halim', the output is 'stevehalim'. After the first (and only) digit '7' and the closest non-digit character 'n' to its left are removed, we have $S2'$ = 'stevehalim',
- $S3$ = 'ab~~c1~~2c', the output is 'ac'. After the first digit '1' and the closest non-digit character 'c' to its left are removed, we have $S3'$ = 'ab2c'. After the (now) first digit '2' and the closest non-digit character 'b' to its left are removed, we have $S3''$ = 'ac',
- $S4$ = 'th3qu1ckbr0wnf0xjumps0v3rth3l4zyd0g', the output is 'tqckbwnxjumprtzyg'.

Design any correct algorithm that solves this task for 8 marks.
For full 15 marks, that correct algorithm must runs in $O(n)$.

## B.2 Strength Competition (15 marks)

Given a Python list $str$ with $n$ Integers where $str[i]$ represents the strength of the $i$-th competitor. There is a competition between these $n$ competitors. The competition are as follows: On each turn, the two current strongest competitors $a$ and $b$ fight each other. The result of this fight is as follows:

- If $a = b$, both $a$ and $b$ are exhausted and eliminated from the competition, or
- If $a \neq b$ and assume $a > b$, then $b$ is exhausted and eliminated from the competition, while $a$ remains in the competition, but with remaining strength $a - b$.

If there is a winner of the competition, output the (last) strength of the last competitor standing; otherwise output 0. Here are a few examples to help you understand this task:

- $str1$ = [7, 7], the output is 0. After the only fight between two competitors with $a = 7$ and $b = 7$, both are eliminated and there is no winner,
- $str2$ = [1, 5, 2], the output is 2. The first fight is between $a = 5$ and $b = 2$, $b = 2$ is eliminated whereas $a$'s strength is now $5 - 2 = 3$. The second fight is between $a = 3$ and $b = 1$, $b = 1$ is eliminated whereas $a$'s strength is now $3 - 1 = 2$ and is the winner,
- $str3$ = [2, 7, 4, 1, 15, 1], the output is 0,
- $str4$ = [99, 70, 96, 60], the output is 7.

Design any correct algorithm that solves this task for 8 marks.
For 10/12/15 marks, that correct algorithm must runs in $O(n^2 \log n)/O(n^2)/O(n \log n)$, respectively.

## B.3 Counting Triples (15 marks)

Given a **strictly increasing** Python list $L$ with $n$ Integers and a positive Integer $d$,
count how many triplets $(i, j, k)$ such that $0 \leq i < j < k < n$, $L[j] - L[i] = d$, and $L[k] - L[j] = d$.
Here are a few examples to help you understand this task:

- $L1$ = [1, 2, 4, 5, 7, 10] and $d = 3$, the output is 2. $(0, 2, 4)$ and $(2, 4, 5)$ are the triplets,
- $L2$ = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] and $d = 3$, the output is 4. $(0, 3, 6)$, $(1, 4, 7)$, $(2, 5, 8)$, and $(3, 6, 9)$ are the triplets.
- $L3$ = [1, 2, 3, 4, 5, 77, 79, 81] and $d = 2$, the output is 2.

Design any correct algorithm that solves this task for 8 marks.
For 12/15 marks, that correct algorithm must runs in $O(n \log n)/O(n)$, respectively.

## B.4 Combining Integers from Two BSTs (15 marks)

Given two Binary Search Trees (BSTs) of Integers rooted at $r1$ and $r2$, respectively, return a new Python list containing all the Integers from both BSTs, in non-decreasing order. Assume that the BSTs can be unbalanced, each BST contains $n$ distinct Integers, but there can be an Integer in one BST that also appears in the other BST, e.g., 27 in Figure 1 below.

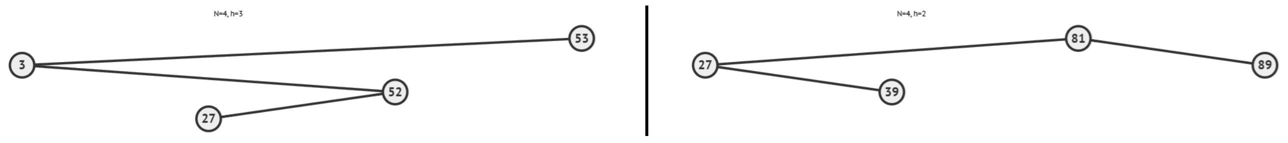


Figure 1: BST1 = {3, 27, 52, 53/$r1$}, BST2 = {27, 39, 81/$r2$, 89},
Output: sorted Python list [3, 27, 27, 39, 52, 53, 81, 89].

Design any correct algorithm that solves this task for 10 marks.
For 15 marks, that correct algorithm must runs in $O(n)$.

### B.5 Pairs with the Maximum Friendship Score (15 marks)

There are $n$ ($2 \leq n$) people and $m$ ($0 \leq m \leq n*(n-1)/2$) bidirectional friendship information between these $n$ people (there is at most one friendship information between any pair and no one is defined as a friend of him/herself).

We define friendship score of two different people $a$ and $b$ as the number of distinct friendship(s) that involves either $a$ or $b$ (note that if $a$ and $b$ are friends with each other, do not double count this friendship).

What is the maximum friendship score between any possible pairs of people?

An example is shown below:

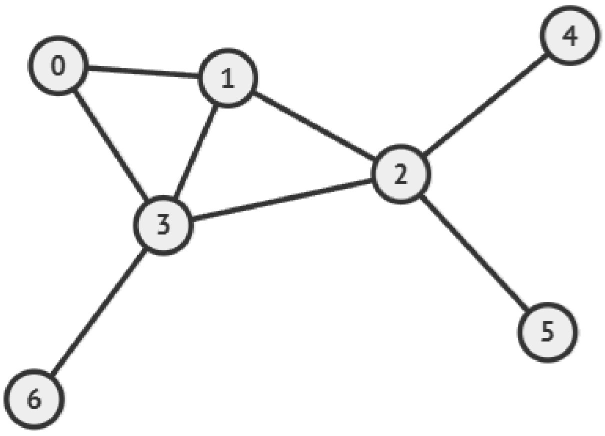


Figure 2: The friendship score of $a = 2$ and $b = 3$ is 7 and this is the maximum possible friendship score; There are 7 distinct friendships (every friendships, except friendship between 0 and 1, involve either $a = 2$ or $b = 3$)

Design any correct algorithm that solves this task for 10 marks.

For 15 marks, that correct algorithm must runs in $O(n^2)$.

Clarification: You are not told how this friendship information is given.

Add the necessary data structure(s) to help you solve this task.

## B.6 City with the Strongest Reachability (15 marks)

There are $n$ cities in your country (labeled with $[0..n-1]$). You are given an Edge List $E$ where $E[i] = (u_i, v_i, w_i)$ describes a bidirectional road between two distinct cities $u_i$ and $v_i$, with length $w_i$ kilometers between them. There can be up to $m = n \cdot (n-1)/2$ edges in $E$ and you have to assume this possibility, i.e., $m \in O(n^2)$.

You own an electric car and unfortunately it has a limited range of $k$ kilometers (for this problem, assume that you can only charge your electric car at home, you leave home with full battery, but you cannot charge your battery anywhere else).

Your task is to identify the city index that if you live in that city, you can reach the **largest number of other cities** with your limited-range electric car (for this problem, assume that you do not need to care whether you can return back to your home). If there are multiple such cities, identify the city with the largest index.

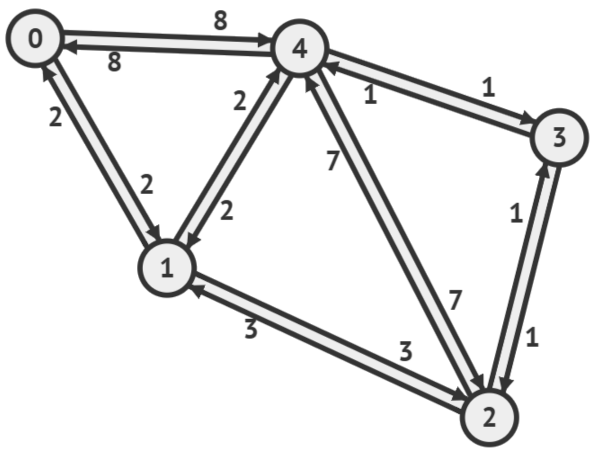


Figure 3: With $k = 2$, if you live at city 4, you can reach 3 other cities: 1, 3, and 2

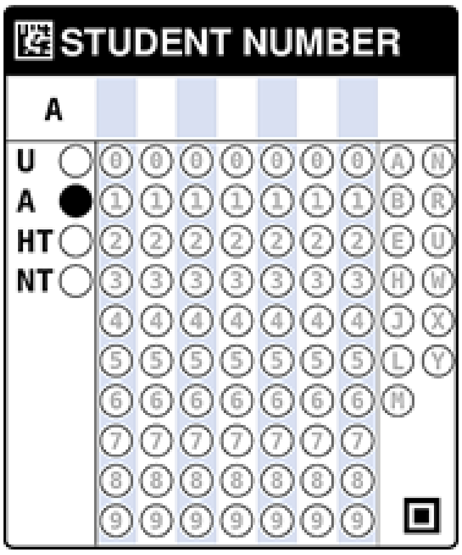
See Figure 3. If your electric car only has range $k = 2$ kilometers, then if you live in:

- City 0, you can only reach at most 1 other city: City 1,
- City 1, you can reach 2 other cities: City 0 and 4,
- City 2, you can reach 2 other cities: City 3 and 4 (via path $2 \to 3 \to 4$)
- City 3, you can reach 2 other cities: City 2 and 4,
- City 4, you can reach 3 other cities: City 1, 3, and 2 (via path $4 \to 3 \to 2$)

Design any correct algorithm that solves this task for 10 marks.
For 14/15 marks, that correct algorithm must runs in $O(n^3 \log n)/O(n^3)$, respectively.

# The Answer Sheet for Semester 2 AY2024/25

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



This portion is for examiner's use only

| Section | Maximum Marks | Your Marks | Grading Remarks |
|---|---|---|---|
| A | 10 | | |
| B | 90 | | |
| Total | 100 | | |

**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 |
|---|---|---|---|---|---|
| A | ○ | ○ | ○ | ○ | ○ |
| B | ○ | ○ | ○ | ○ | ○ |
| C | ○ | ○ | ○ | ○ | ○ |
| D | ○ | ○ | ○ | ○ | ○ |
| E | ○ | ○ | ○ | ○ | ○ |

Box B.1 Deleting Characters

I claim that my solution runs in $O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$.

Box B.2. Strength Competition

I claim that my solution runs in $O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$.

Box B.3. Counting Triples

I claim that my solution runs in $O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$.

Box B.4. Combining Integers from Two BSTs

I claim that my solution runs in $O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$.

Box B.5. Pairs with the Maximum Friendship Score

I claim that my solution runs in $O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$.

Box B.6. City with the Strongest Reachability

I claim that my solution runs in $O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$.

– END OF PAPER; All the Best –