

National University of Singapore
School of Computing
CS2040S - Data Structures and Algorithms
Midterm Test
(S1 AY2026/27; Thu, 01 October 2026)

Time Allowed: 90 minutes

INSTRUCTIONS TO CANDIDATES:

1. Do **NOT** open this assessment paper until you are told to do so.
2. This assessment paper comprises TEN (10) printed pages, including this page.
3. This is an **Open Book Assessment**.
You cannot use any electronic device except one 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.
The first two questions in Section B have Generative AI answers (not necessarily generated with the correct prompt) that you can use/fix/improve (or rewrite).
6. If you choose to write your own answer and for the last question in Section B,
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 MergeSort on `ArrayList<Integer> A`, run QuickSort on `int[] array B`, etc.
7. The total marks is 50. All the best :)

A Midterm Feedback (5 marks)

Write two paragraphs about your experience with AI in the first half of CS2040S of S1 AY2026/27.

The first paragraph is to elaborate how do you use AI specifically for this course so far.

The second paragraph is about your subjective judgment: Do you think using AI (as you described in the first paragraph) makes you more confident, neutral, or less confident in facing this pen-and-paper proctored paper without AI assistance?

You will be given 1 mark per each logical sentence that you write, for up to 5 marks for this section. You can distribute 1 + 4, 2 + 3, 3 + 2, or 4 + 1 sentences across the two questions.

B Application Questions ($3 \times 15 = 45$ marks)

B.1 Oddly Even (15 marks)

Given a Java `ArrayList<Integer> A` consisting of n ($n \geq 1$) positive integers (each not more than 10^9), your task is to rearrange `A` so that all odd integers appear before all even integers. The relative order between the group of odd integers does not matter (same for the even integers).

Here are several examples:

- Input: `A = [3, 1, 2, 4, 7, 6, 5]` // odd, odd, even, even, odd, even, odd
 Output: `[3, 1, 7, 5, 2, 4, 6]` // odd, odd, odd, odd, even, even, even
 Explanation: There are many possible valid outputs, e.g., `[1, 3, 5, 7, 2, 4, 6]` is also OK.
- Input: `A = [74, 19, 43, 8, 92]`
 Output: `[19, 43, 92, 8, 74]`
 Explanation: even, odd, odd, even, even becomes odd, odd, even, even even
- Input: `A = [1, 3, 5]`
 Output: `[1, 5, 3]`
 Explanation: There is no even integer in `A`, so we list all odd integers (in any order) and done.
- Input: `A = [8, 2, 4]`
 Output: `[2, 4, 8]`
 Explanation: There is no odd integer in `A`, so we list all even integers (in any order) and done.

Solve this problem in Java. To break ties among multiple correct solutions, prioritize in order:

1. Tightest worst-case time complexity.
2. Smallest auxiliary space complexity.

B.2 Valid Brackets (15 marks)

In class, we have learned about bracket matching. A Java String S that only contains '(' (an open bracket) and ')' (a close bracket) is defined as valid if and only if:

- S is empty,
- S can be written as XY (a valid String X is concatenated with another valid String Y), or
- S can be written as (Z) , where Z is a valid string.

You are given a Java String S that contains n ($n \geq 1$) '(' or ')' characters that is not necessarily valid upfront. In one move, you can choose to insert an opening or a closing bracket *at any position* of the string. Your task is to return the minimum number of moves required to make S valid.

Here are several examples:

- Input: $S = ")()"$
Output: 1
Explanation: You can insert one opening bracket at the very front, e.g., $S = "_>()"$.
- Input: $S = "(()())"$
Output: 0
Explanation: S is already valid upfront.
- Input: $S = ")))"$
Output: 2
Explanation: You can insert two opening brackets, e.g., $S = "_()_())"$ or $S = "_()_()_)"$.
- Input: $S = ")))("$
Output: 3
Explanation: As above, plus one more closing bracket at the end, e.g., $S = "_()_())_"$.

Solve this problem in Java. To break ties among multiple correct solutions, prioritize in order:

1. Tightest worst-case time complexity.
2. Smallest auxiliary space complexity.

B.3 Prime Powers (15* marks)

Given a Java `ArrayList<Integer> P` consisting of n ($n \geq 1$) distinct prime integers up to $2^{31} - 1 = 2\,147\,483\,647$, return the first k ($k \geq 1$) prime powers¹ generated by P in sorted ascending order. You are guaranteed that the k -th value will not exceed the 32-bit signed integer maximum ($2\,147\,483\,647$).

Here are several examples:

- Input: $P = [3, 2]$, $k = 7$
 Output: $[2, 3, 4, 8, 9, 16, 27]$
 Explanation: $[2^1 = 2, 3^1 = 3, 2^2 = 4, 2^3 = 8, 3^2 = 9, 2^4 = 16, 3^3 = 27]$.
- Input: $P = [5, 997, 7]$, $k = 7$
 Output: $[5, 7, 25, 49, 125, 343, 625]$
 Explanation: 997^1 would be the 8-th element in the sorted sequence.
- Input: $P = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53]$, $k = 3$
 Output: $[2, 3, 4]$
 Explanation: $[2^1 = 2, 3^1 = 3, 2^2 = 4]$.
- Input: $P = [131071]$, $k = 1$
 Output: $[131071]$
 Explanation: A larger k is impossible under the maximum value guarantee, as the next power already overflows a 32-bit signed integer ($131071^2 > 2^{31} - 1$).

Notice:

- n can be as large as the total number of primes up to $2^{31} - 1$ (i.e., $n \leq 105\,097\,565$).
- k can be significantly smaller than n (e.g., $k = 1$).

Solve this problem in Java. To break ties among multiple correct solutions, prioritize in order:

1. Tightest worst-case time complexity.
2. Smallest auxiliary space complexity.

¹In mathematics, a prime power is a positive integer that can be written as a single prime number raised to a positive integer exponent. Mathematically, it is expressed as $n = p^k$, where p is a prime number and k is a positive integer.

The Answer Sheet for S1 AY2026/27

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

Section	Maximum Marks	Your Marks	Grading Remarks
A	5		
B	45		
Total	50		

Box A.1. Feedback about usage of AI in CS2040S S1 AY2026/27:

How do you use AI in this course so far?

Subjectively, are you now more confident, neutral, or less confident to do this paper?

Box B.1. Oddly Even

```
import java.util.ArrayList;
import java.util.Comparator;

public class SolutionTwin {

    public static void rearrangeOddEven(ArrayList<Integer> A) {
        A.sort(new Comparator<Integer>() {
            @Override
            public int compare(Integer a, Integer b) {
                int keyA = (Math.abs(a) % 2 == 1) ? 1 : 0;
                int keyB = (Math.abs(b) % 2 == 1) ? 1 : 0;

                return Integer.compare(keyA, keyB);
            }
        });
    }

    public static void main(String[] args) {
        ArrayList<Integer> A = new ArrayList<>();
        A.add(3);
        A.add(1);
        A.add(2);
        A.add(4);
        A.add(7);
        A.add(6);
        A.add(5);

        rearrangeOddEven(A);
        System.out.println(A);
    }
}
```

My analysis for SolutionTwin is ($5 \times 1 = 5$ marks):

Ignoring time complexity, it gives correct output for all valid inputs: ☐ YES | ☐ NO, I will elaborate

Ignoring correctness, it runs in: ☐ $O(n^2)$ | ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ Other, I will elaborate

The **auxiliary** space that it uses is: ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ Other, I will elaborate

It can crash on at least one valid corner case input: ☐ YES, I will elaborate | ☐ NO

I can improve upon the Generated AI code: ☐ YES, I will elaborate | ☐ NO, it is the best possible

Elaboration of my analysis above (10 marks):

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ Other, as written above,
the **auxiliary** space that it uses is: ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ $O(1)$ | ☐ Other, as written above.

Box B.2. Valid Brackets

```
import java.util.Stack;

class SolutionTalk {
    public int minMoves(String S) {
        Stack<Character> stack = new Stack<>();
        int insertOpen = 0;

        for (char c : S.toCharArray()) {
            if (c == '(') {
                stack.push(c);
            } else {
                if (!stack.isEmpty()) {
                    stack.pop();
                } else {
                    insertOpen++;
                }
            }
        }

        return insertOpen + stack.size();
    }

    public static void main(String[] args) {
        String S = "(()())";
        System.out.println(new SolutionTalk().minMoves(S));
    }
}
```

My analysis for SolutionTalk is ($5 \times 1 = 5$ marks):

Ignoring time complexity, it gives correct output for all valid inputs: ☐ YES | ☐ NO, I will elaborate

Ignoring correctness, it runs in: ☐ $O(n^2)$ | ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ Other, I will elaborate

The **auxiliary** space that it uses is: ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ Other, I will elaborate

It can crash on at least one valid corner case input: ☐ YES, I will elaborate | ☐ NO

I can improve upon the Generated AI code: ☐ YES, I will elaborate | ☐ NO, it is the best possible

Elaboration of my analysis above (10 marks):

Only if you need to improve upon the Generated AI code.

My faster+correct code runs in: ☐ $O(n \log n)$ | ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ Other, as written above,
the **auxiliary** space that it uses is: ☐ $O(n)$ | ☐ $O(\log n)$ | ☐ $O(1)$ | ☐ Other, as written above.

Box B.3. Prime Powers (This is a STARRED task without Generative AI assistance)

- ☐ I give up, just give me the free 2 marks (you must leave the box below totally empty or cross it)
- ☐ I know how to solve this, I will elaborate below (pseudocode is allowed and encouraged):

My solution runs in $O(\text{-----})$ and uses $O(\text{-----})$ **auxiliary** space.

– END OF PAPER; All the Best –