**Submission Instructions**

* **Single Submission**: Only the **group leader** should upload the final **.pdf report** using this template.
* **Filename Format**: Follow this naming convention:

**AB**\_**CD**\_001220561\_001220562\_001220563\_001220564.pdf

* + **AB** → Lab number (from your timetable or Lab details in Moodle).
  + **CD** → Group number (assigned in the Excel lab group assignment file).
  + **001220561\_001220562\_001220563\_001220564** → Student IDs of all group members.
* **Ensure all guidance text needs to be removed before submission.**
* **Refer to the coursework specification for full details.**

**Algorithms and Data Structures (ADS) - COMP1819**

Develop and optimise solutions in Python with ADS and provide complexity analysis.

Group Name: **AB\_CD**

Team members: with different text colours

|  |  |  |  |
| --- | --- | --- | --- |
| Member | Name | ID | Contribution % |
| **1** | **Last, First** | **001220561** | **100%** |
| **2** | **Last, First** | **001234562** | **100%** |
| **3** | **Last, First** | **001234563** | **80%** |
| **4** | **Last, First** | **001234564** | **50%** |

Contents

[1. Creating Two Basic Working Solutions (30%) (Maximum 2 pages) 3](#_Toc189832151)

[1.1 Understanding the Problem 3](#_Toc189832152)

[1.2 Solution Implementations 3](#_Toc189832153)

[1.3 Code Submission 3](#_Toc189832154)

[2. Testing and Comparing Solutions (20%) (Maximum 2 pages) 3](#_Toc189832155)

[2.1 Test Cases & Validation 3](#_Toc189832156)

[**Results for Solution 1** 4](#_Toc189832157)

[2.2 Running on Given Test Cases 4](#_Toc189832158)

[2.3 Comparison of Solutions 4](#_Toc189832159)

[**Running time graphs** 4](#_Toc189832160)

[3. Optimising Solutions (20%) *(Maximum 2 pages)* 5](#_Toc189832161)

[3.1 Selecting a Solution for Optimisation 5](#_Toc189832162)

[3.2 Optimisation Steps & Justification 5](#_Toc189832163)

[3.3 Code Submission 5](#_Toc189832164)

[4. Comparing Performance (15%) *(Maximum 2 pages)* 5](#_Toc189832165)

[4.1 Performance Comparison 5](#_Toc189832166)

[4.2 Running on Given Test Cases 5](#_Toc189832167)

[4.3 Visualising Performance 5](#_Toc189832168)

[5. Reflecting on Teamwork (15%) *(Maximum 2 pages)* 5](#_Toc189832169)

[5.1 Contribution Marks & Team Agreement 5](#_Toc189832170)

[5.3 Summary of Roles & Contributions 6](#_Toc189832171)

[5.2 Weekly Journal & Communication Logs 6](#_Toc189832172)

[**Weekly journal** 6](#_Toc189832173)

[5.4 Final Report Quality 7](#_Toc189832174)

[Reference 7](#_Toc189832175)

[Appendix 7](#_Toc189832176)

[(Include all solutions as clear text, not screenshots) 7](#_Toc189832177)

[Appendix A.1 - Proposed solutions 7](#_Toc189832178)

[Appendix C - Further evidence of team contribution if necessary 8](#_Toc189832179)

# 1. Creating Two Basic Working Solutions (30%) (Maximum 2 pages)

## 1.1 Understanding the Problem

Each team member provides a **handwritten explanation** with at least **five examples** of different input string lengths and expected outputs. These will later be used to validate the solutions' correctness.

|  |  |  |
| --- | --- | --- |
| **#** | **Handwritten with explanation (attached)** | **By (student name)** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

## 1.2 Solution Implementations

**Solution 1:** Description of the first approach and its data structure.

Short description and highlights of the **use of the data structure in your code**

1. **if** \_\_name\_\_ == '\_\_main\_\_':
2. queue = Queue()
3. queue.put(1)
4. queue.put(2)
5. queue.put(3)
6. queue.put(4)
7. queue.put(5)

10. reverseqQueue(queue) #your implementation
11. printQueue(queue)

**Solution 2:** Description of the second approach and its data structure.

Short description and highlights of the **use of the data structure in your code**

## 1.3 Code Submission

The full source code is included in the **Appendix** (as clear text, not a screenshot).

# 2. Testing and Comparing Solutions (20%) (Maximum 2 pages)

## 2.1 Test Cases & Validation

* Test the solutions using the **handwritten examples** from **Task 1**.
* Provide a **table** summarising the test cases, expected outputs, and actual outputs with **snapshots of generated outputs**.

### **Results for Solution 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Input** | **Output** | **Pass/Fail** | **Running time (s)** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |

## 2.2 Running on Given Test Cases

* Run both solutions on the **10 given test cases**.
* Record the **output** and **execution time** (stop execution after 60 seconds if needed).
* Include these details in the report and as **comments in the code**.

## 2.3 Comparison of Solutions

* Highlight differences in **data structures** and their impact on **performance**.
* Provide a **theoretical analysis** of **time complexity** for both approaches.
* Generate a **graph** showing how runtime changes across **10 test inputs**.

### **Running time graphs**

A graph with red and blue lines

Description automatically generated

# 3. Optimising Solutions (20%) *(Maximum 2 pages)*

## 3.1 Selecting a Solution for Optimisation

* Choose **one basic solution** to optimise, improving efficiency while ensuring all test cases run within **60 seconds**.

## 3.2 Optimisation Steps & Justification

* Clearly explain each **optimisation step** and its effect on **performance**. **Restrictions:** Miller-Rabin method and external libraries **are not allowed**.

## 3.3 Code Submission

* Include the final **optimised code** in the **Appendix** (as clear text, not a screenshot).

# 4. Comparing Performance (15%) *(Maximum 2 pages)*

## 4.1 Performance Comparison

* Compare the optimised solution with the basic solution.
* Highlight differences in data structures, algorithms, or implementation approaches.

## 4.2 Running on Given Test Cases

* Run the optimised solution on the 10 given test cases.
* Record outputs and runtime measurements (stop after 60 seconds if necessary).
* Include these details in the report and as comments in the code.

## 4.3 Visualising Performance

* Generate a graph comparing runtime performance between the optimised solution and the basic solution.
* Analyse and explain the time complexity and Big-O notation for the optimised solution.

# 5. Reflecting on Teamwork (15%) *(Maximum 2 pages)*

## 5.1 Contribution Marks & Team Agreement

* Assign contribution marks to each member based on mutual agreement.

(There is a group mark decided by the marker. Each team member's overall contribution is assessed on a scale from 0% to 100%, with agreement from the team. For instance, if a member did not contribute to problem optimisation, they might receive 80% out of the 100%. An 80% individual effort could result in 80% of the group mark, but the final decision rests with the marker)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Name | ID | Task 1 (30%) | Task 2 (20%) | Task 3 (20%) | Task 4 (15%) | Task 5 (15%) | **Contribution mark**  **(100%)** |
| **Last, First (Group leader)** | **001234561** | **30%** | **20%** | **20%** | **15%** | **15%** | **100%** |
| **Last, First (Developer 1)** | **001234562** | **30%** | **20%** | **20%** | **15%** | **15%** | **100%** |
| **Last, First (Developer 2)** | **001234563** | **30%** | **20%** | **20%** | **15%** | **15%** | **100%** |
| **Last, First (Analyst )** | **001234564** | **20%** | **20%** | **10%** | **0%** | **10%** | **60%** |

## 5.3 Summary of Roles & Contributions

* Provide a clear summary of each member’s role and their specific contributions.

Suggested Roles for a 4-Member Team:

* **Group Leader** – Manages deadlines, organises meetings, ensures report clarity, and oversees final submission.
* **Developer** **1** – Implements one distinct solution, ensures correctness, and documents runtime measurements, and maintains the team journal.
* **Developer** **2** – Implements one distinct solution, optimizes one solution, justifies improvements, updates the Appendix.
* **Analyst** – Take charge of testing, runs solutions on given inputs, records outputs, and analyses performance with graphs.

## 5.2 Weekly Journal & Communication Logs

* Maintain a weekly journal tracking meetings, discussions, and individual contributions.

### **Weekly journal**

|  |  |  |
| --- | --- | --- |
|  | **Task note** | **Status** |
| **Week 1: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Developer 1) |  |  |
| Last, First (Developer 2) |  |  |
| Last, First (Analyst ) |  |  |
| **Week 2: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Developer 1) |  |  |
| Last, First (Developer 2) |  |  |
| Last, First (Analyst ) |  |  |
| **Week 3: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Developer 1) |  |  |
| Last, First (Developer 2) |  |  |
| Last, First (Analyst ) |  |  |
| **Week 4: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Developer 1) |  |  |
| Last, First (Developer 2) |  |  |
| Last, First (Analyst ) |  |  |
| **Week 5: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Developer 1) |  |  |
| Last, First (Developer 2) |  |  |
| Last, First (Analyst ) |  |  |
| **Week 6: from date-date** |  |  |
| Last, First (Group leader) |  |  |
| Last, First (Developer 1) |  |  |
| Last, First (Developer 2) |  |  |
| Last, First (Analyst ) |  |  |

## 5.4 Final Report Quality

* Ensure the report is well-structured, clearly written, properly referenced, and follows all given specifications.

# Reference

Tuan Vuong, COMP1819ADS, (2022), GitHub repository, <https://github.com/vptuan/COMP1819ADS>

# Appendix

(Include all solutions as clear text, not screenshots)

# Appendix A.1 - Proposed solution 1

You can try to use Pycharm or VSCode to paste Python code into Word document. Note that it is important to keep the Python code in clear-text, good structure, and text format for readability.

1. """
2. Spyder Editor: Spyder 4.2.1
4. This demo is for Lab 02 - Ex1 MinMax function
5. """
6. **import** time
8. **def** minmax(sequence):
9. min = max = sequence[0] # assuming no-empty
10. **for** val **in** sequence:
11. **if** (val > max):
12. max = val
13. **if** (val < min):
14. min = val
15. **return** (min,max)
17. #print(minmax([1,2,3,5]))

20. **def** measure\_time(input\_size):
21. sequence = [i **for** i **in** range(input\_size)] # input = a list [0,1,2,...]
22. #print(sequence)
23. start = time.time() # start timer
24. **print**(minmax(sequence)) # execute the function with the sequence
25. **print**("Input size=", input\_size, " Time taken=", time.time()-start)

28. # Now, we make input size larger, 2k, 10k,50k, 200k,1000k
30. k = 1000;
31. measure\_time(2\*k)
32. measure\_time(10\*k)
33. measure\_time(50\*k)
34. measure\_time(200\*k)
35. measure\_time(1000\*k)
37. # Now, we plot in Excel. The plot looks linear? This is O(n) because
38. # the for loop in line 12.

# Appendix A.2 - Proposed solution 2

# Appendix A.3 – Optimised solution

# Appendix B - Further evidence of team contribution if necessary

Such as chat log, meeting log …