School of Computer Science and AI, SR University

# Advanced Data Structure and Algorithm using python/Java

**Course Code: 25MSC402PC501**

## Unit I
## Abstract Data Type

**Assignment**


**Submitted by:**
**Name: Elijah M. Flomo**
**Enrol No. 2506B09602**
**Course: Msc Computer Science**

**Submitted to:**
**Name: Dr. Rahul Mishra**
**email: rahul.mishra@sru.edu.in**
**Title: Course Instructor**


**Date: 24 Oct 2025**

**Q 1: Find the time and space complexity for the following loops**

```
( a ) cnt = 0
f o r  i  i n range ( 1 , n+1)
f o r j i n range ( 1 , n+1)
cnt + = 1
```

```
( b ) cnt = 0
i =1
w h i l e ( i <=n )
f o r j i n range ( 1 , i +1);
cnt + =1
 i *=2
```

**Answer:**

(a)

```
cnt = 0
for i in range(1, n+1):
    for j in range(1, n+1):
        cnt += 1
```

Step 1: Time Complexity

- Outer loop runs from i = 1 to i = n → n iterations
- Inner loop runs from j = 1 to j = n → n iterations for each i
- Total operations: n * n = $n^2$
- Time Complexity (T(n)) = $O(n^2)$

Step 2: Space Complexity

- We are using:

  cnt → 1 variable

  i and j → loop variables

- No extra data structures are used.
- Space Complexity (S(n)) = O(1) (constant space)

**Answer (a):**

- Time Complexity = $O(n^2)$
- Space Complexity = O(1)

**(b)**

```
cnt = 0
i = 1
while (i <= n):
   for j in range(1, i+1):
      cnt += 1
   i *= 2
```

Step 1: Analyze Loops

- Outer while loop: i = 1, 2, 4, 8, ..., n
  → i doubles each time → number of iterations ≈ $\log_2(n)$
- Inner for loop: runs from 1 to i → i iterations

Step 2: Count Total Operations

Iterations of cnt += 1:

i = 1 → 1 iteration
i = 2 → 2 iterations
i = 4 → 4 iterations
i = 8 → 8 iterations
...
i ≤ n

- Total = 1 + 2 + 4 + 8 + ... + ≤ n

  This is a geometric series: 1 + 2 + 4 + ... + n ≈ 2n − 1

- So, Time Complexity (T(n)) = O(n)

Step 3: Space Complexity

- Variables used: cnt, i, j
- No extra data structures.
- Space Complexity (S(n)) = O(1)

Answer (b):

- Time Complexity = O(n)
- Space Complexity = O(1)

**Q 2: What is the worst case time complexity of inserting n elements into an empty linked list, if the linked list needs to be maintained in sorted order?**

**Answer:**

Problem

- We have an empty linked list.
- We want to insert n elements one by one.
- The linked list must remain sorted after each insertion.
- We need to find worst-case time complexity.

Step 1: Understand insertion in a sorted linked list

- For a singly linked list, inserting an element in sorted order involves:

    1. Traversing the list to find the correct position.
       In the worst case, the element might need to go at the end.

       Traversal takes O(k) time, where k = current number of elements in the list.

    2. Inserting the element (changing pointers) → O(1)

Step 2: Count operations for all n elements

- 1st element → 0 comparisons (list is empty) → O(1)

- 2nd element → compare with 1 element → O(1)
- 3rd element → compare with 2 elements → O(2)
- nth element → compare with n-1 elements → O(n-1)

Total operations:

$0 + 1 + 2 + ... + (n-1) = n(n-1)/2$

Step 3: Determine worst-case time complexity

- $n(n-1)/2 \approx O(n^2)$
- Space Complexity:

  Each node requires O(1) space, so total = O(n) for n nodes.

Answer

- Worst-case Time Complexity = $O(n^2)$
- Space Complexity = O(n)

**Q 3: Explain the trade-offs between singly, doubly, and circular linked lists in terms of:**

**Memory usage**

**Insertion and deletion complexity**

**Cache performance**

Answer:

1. Memory Usage

| Type of Linked List | Memory Usage per Node | Explanation |
|---|---|---|
| Singly Linked List | 1 pointer (next) | Each node stores only the address of the next node. Least memory usage. |
| Doubly Linked List | 2 pointers (prev + next) | Stores both previous and next pointers → about twice the pointer memory of singly linked list. |
| Circular Linked List | 1 pointer (singly) or 2 pointers (doubly) | Same as singly or doubly, but the last node points back to the first node. Slight overhead in logic, not in memory. |

Trade-off:

- Singly uses the least memory.
- Doubly uses more memory but allows easier backward traversal.
- Circular adds logical complexity but no extra memory for singly; if doubly circular, memory usage is same as doubly.

2. Insertion and Deletion Complexity

| Operation | Singly Linked List | Doubly Linked List | Circular Linked List |
|---|---|---|---|
| Insert at head | O(1) | O(1) | O(1) |
| Insert at tail | O(n) (unless tail pointer exists) | O(1) (if tail pointer exists) | O(1) if tail pointer maintained, else O(n) |
| Delete at head | O(1) | O(1) | O(1) |
| Delete at tail | O(n) | O(1) | O(1) if tail pointer maintained, else O(n) |
| Insert/Delete at middle | O(n) | O(n) | O(n) |

Trade-off:

- Singly: Insertion/deletion is simple at head but costly at tail or middle.
- Doubly: Faster insertion/deletion anywhere because we have a prev pointer.
- Circular: Useful for applications where we need to loop infinitely, like round-robin scheduling. Insertion/deletion similar to singly/doubly depending on type.

3. Cache Performance

- Linked lists generally have poor cache performance because nodes are scattered in memory.
- Singly:
    - Slightly better cache usage than doubly because less memory per node → more nodes may fit in a cache line.
- Doubly:
    - Uses more memory → fewer nodes per cache line → slightly worse cache performance.
- Circular:
    - Cache performance similar to singly/doubly depending on pointer usage. Circularity does not improve caching; it only changes traversal logic.

Summary of Trade-offs

| Feature | Singly | Doubly | Circular |
|---|---|---|---|
| Memory Usage | Low | High | Low/High |
| Insertion/Deletion | Simple at head, slow elsewhere | Fast anywhere | Fast anywhere if tail pointer maintained |
| Traversal | Forward only | Forward & backward | Forward only (singly circular) / Both (doubly circular) |

**Q 4: Consider an unordered list of N distinct integers. What is the minimum number of**

**element comparisons required to find an integer in the list that is NOT the largest in**

**the list?**

**Answer:**

Problem Analysis

- We have an unordered list of N distinct integers.
- We want to find an integer that is NOT the largest.
- Question: Minimum number of comparisons required.

Step 1: Understand the worst-case scenario

- If the list is unordered, the largest element is unknown.
- To ensure we pick an element that is not the largest, we only need to avoid the largest element.
- Observation:
    - If we take any element except the largest, we succeed.
    - The smallest number of comparisons is not about sorting; it's about identifying one element that is guaranteed not to be the largest.

Step 2: Strategy to minimize comparisons

- The largest element is the only element we cannot pick.
- To find any element that is NOT the largest:
- Pick any two elements and compare them.
- The larger one might be the largest, but the smaller one cannot be the largest.
- Therefore, after 1 comparison, we know which of the two elements is definitely NOT the largest.
- Key idea: You don't need to check all N elements. Just pick any two and take the smaller one.

Step 3: Minimum number of comparisons

- Comparisons needed = 1 (compare any two elements and pick the smaller one).

Answer:

- Minimum number of element comparisons required = 1

**Q 5: Write a function to rotate a linked list by k nodes. For example:**

**Input: 10→20→30→40→50, k = 2**

**Output: 30→40→50→10→20**

**Answer:**

Problem Analysis

We have a singly linked list:

10 → 20 → 30 → 40 → 50

We want to rotate it by k nodes.

- k = 2
- Output should be:

30 → 40 → 50 → 10 → 20

Idea:

1. Traverse the list to find the kth node.
2. Break the list at the kth node.
3. The (k+1)th node becomes the new head.
4. Connect the original last node to the original head.

Implementation

Link:

https://drive.google.com/file/d/19IGMc72XffIDDB_Bs9G4GIIkQD9RgqfU/view?usp=sharing

**Q 6: Write a function to split a circular linked list into two halves. Analyze the boundary**

**cases when the number of nodes is odd or even.**

**Example 1: Odd number of nodes**

**Original Circular Linked List:**

1 → 2 → 3 → 4 → 5 → (back to head)

**First half:**

1 → 2 → 3 → (back to head)

**Second half:**

4 → 5 → (back to head)

**Example 2: Even number of nodes**

**Original Circular Linked List:**

10 → 20 → 30 → 40 → 50 → 60 → (back to head)

**First half:**

10 → 20 → 30 → (back to head)

**Second half:**

40 → 50 → 60 → (back to head

**Answer:**

Problem Analysis

- We have a circular singly linked list.
- We want to split it into two halves, both of which are circular.
- Key points:
    - If an odd number of nodes → first half has one extra node.
    - If even → halves are equal.
- We'll use the slow and fast pointer technique (similar to finding the middle of a linked list).

Implementation

Link:

https://drive.google.com/file/d/19IGMc72XffIDDB_Bs9G4GIIkQD9RgqfU/view?usp=sharing

**Q 7: Write a function to check if a linked list is a palindrome using: O(1) space complexity**

**and O(n) time complexity.**

**Answer:**

Problem Analysis

- Palindrome linked list: A linked list that reads the same forward and backward.
    - Example: 1 → 2 → 3 → 2 → 1 → palindrome
- Constraints:
    - Time Complexity: O(n)
    - Space Complexity: O(1) (no extra array or stack allowed)

Idea:

1. Find the middle of the linked list using slow and fast pointers.
2. Reverse the second half of the linked list in place.
3. Compare the first half and the reversed second half.
4. Restore the original linked list (optional, but good practice).
5. Return True if palindrome, else False.

Implantation

Link:

https://drive.google.com/file/d/1UqZ6gHcJoRWZcn5Cp3n2DzuBraomNLm3/view?usp=sharing

**Q 8: Write a simple airline ticket reservation program. The program should display a menu**

**with the following options: reserve a ticket, cancel a reservation, check whether a**

**The ticket is reserved for a particular person, and displays the passengers. The information**

**is maintained on an alphabetized linked list of names. In a simpler version of the**

**program, assume that the tickets are reserved only for one flight. In a fuller version,**

**place no limit on the number of flights. Create a linked list of flights with each node**

**including a reference to a linked list passengers**

**Answer:**

Link:

https://drive.google.com/file/d/1kFmiH8vBEUSeQJGLnvIAjCl1tJQA_AP7/view?usp=sharing