Question 1: Why is a ring buffer useful and/or when should it be used?

A ring buffer, also known as a circular buffer, is a data structure that has various practical applications, especially in scenarios where performance and efficient memory usage are crucial. Ring buffers are particularly useful in the following situations:

1. **Data Streaming and I/O Buffers:** Ring buffers are generally used when in data streaming applications, for example during audio and video processing. Data is continuously generated and consumed in a circular fashion in this process, and they provide an efficient way to store and process the incoming data. Ring buffers can efficiently store and process real-time audio data.

2. **Communication and Networking:** Ring buffers are useful for managing incoming and outgoing data packets in networking protocols and communication systems. It allows data to be continuously processed even when the buffer reaches its capacity. This is especially useful when data loss is not tolerated since ring buffers know how to handle overflow conditions.

3. **High-Performance Systems:** High-performance computing and embedded systems highly depend on ring buffers when low latency is critical. They're used for tasks such as managing task queues, storing sensor data in real-time, and when peripherals need buffered input/output data. Here, ring buffers reduce the amount of overhead associated with dynamic memory allocation and deallocation.

Pros of using a ring buffer:

- Constant-time complexity for both enqueue and dequeue operations.
- Efficient memory utilization, as the buffer size remains fixed.
- Most suitable for real-time and high-performance applications.
- Doesn't use dynamic memory which means no memory leaks.
- Conserves memory as we only store up to our capacity (opposed to a queue which could continue to grow if input outpaces output.)
- Simple Implementation, which makes it easier to trust and test.
- Never has to reorganize / copy data around.

Cons of using Ring buffer:

- Limited capacity: Buffer size is fixed, can lead to data loss if not managed properly.
- Complexity in handling variable-sized data: Ring buffers work best when dealing with fixed-sized data.

Complexity:

The complexity of ring buffer is O(1). As all operations occur in constant time. The enqueue and deque operations take constant time.

Question 2: Why is a stack useful and/or when should it be used?

Stacks are fundamental data structures that follow the Last-In, First-Out (LIFO) principle, which is useful in various scenarios:

1. **Function Call Management:** Stacks are generally used in managing function calls. When you call a function, the local variables and return addresses are pushed to the stack. If the function exits, the stack is 'popped', and the program can return to the previous state. Stack management is absolutely crucial to track program execution flow.
2. **Expression Evaluation:** Mathematical expression evaluation also uses stacks, particularly parentheses and operators with different precedence levels. Operand stacks keep track of these intermediate results, which allow efficient expression parsing and evaluation.
3. **Memory Allocation:** Memory allocation in computer programs also employ stacks. In C, local variables are generally allocated on the stack, which provide fast and deterministic memory allocation and deallocation. This contrasts to heap, which usually involves more overhead and can cause memory leaks if improperly managed.
4. **Backtracking and Undo Operations:** Stacks are also used in backtracking algorithms and undo operations in text editors, games, etc. During the program's execution, the stack stores the state at various points which allow for easy recovery and exploration of several different paths.

Pros of using a stack:

- **Easy implementation:** Stack data structure is easy to implement using arrays or linked lists, and its operations are simple to understand and implement.
- **Efficient memory utilization**: Stack uses a contiguous block of memory, making it more efficient in memory utilization as compared to other data structures.
- **Fast access time:** Stack data structure provides fast access time for adding and removing elements as the elements are added and removed from the top of the stack.
- **Helps in function calls:** Stack data structure is used to store function calls and their states, which helps in the efficient implementation of recursive function calls.
- **Supports backtracking:** Stack data structure supports backtracking algorithms, which are used in problem-solving to explore all possible solutions by storing the previous states.
- **Used in Compiler Design:** Stack data structure is used in compiler design for parsing and syntax analysis of programming languages.
- **Enables undo/redo operations**: Stack data structure is used to enable undo and redo operations in various applications like text editors, graphic design tools, and software development environments.

Cons of using a stack:

- **Not suitable for certain applications:** Stack data structure is not suitable for applications that require accessing elements in the middle of the stack, like searching or sorting algorithms.
- **Stack overflow and underflow**: Stack data structure can result in stack overflow if too many elements are pushed onto the stack, and it can result in stack underflow if too many elements are popped from the stack.
- **Recursive function calls limitations:** While stack data structure supports recursive function calls, too many recursive function calls can lead to stack overflow, resulting in the termination of the program.

Complexity:

All operations of stack also take O(1) time complexity. Which means all operations are done in constant time.

Citations:

[1]
B. Ward, "Circular Queue or Ring Buffer," *Medium*, May 01, 2020. https://towardsdatascience.com/circular-queue-or-ring-buffer-92c7b0193326

[2]
"Applications, Advantages and Disadvantages of Stack," *GeeksforGeeks*, May 16, 2022. https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-stack/