Report on Collections

1. ArrayList

Structure

- ArrayList is a non-generic collection that can hold items of any type (object).
- It resizes dynamically as elements are added or removed.

Time Complexity

- **Access:** O(1)
- **Search:** O(n)
- **Insert/Delete:** O(n)

Business Case

ArrayList is suitable for scenarios where the types of elements are not known at compile-time or when a collection of objects needs to be managed without type safety.

Example

```
ArrayList arrayList = new ArrayList();
arrayList.Add(1);
arrayList.Add("Somaya");
arrayList.Add(5.5);
for (int i = 0; i < arrayList.Count; i++)

    Console.WriteLine(arrayList[i]);
```

2. List<T>

Structure

- List<T> is a generic collection that stores elements of a specified type.
- It resizes dynamically like ArrayList, but provides type safety.

Time Complexity

- **Access:** O(1)
- **Search:** O(n)
- **Insert/Delete:** O(n)

Business Case

List<T> is ideal for scenarios where type safety is required, and the size of the collection may change.

Example

```
List<int> list = new List<int> { 1, 2, 3 };
list.Add(4);
// list.Add("Somaya"); // Invalid

Console.WriteLine(list[3]);
```

3. LinkedList<T>

Structure

- LinkedList<T> is a doubly linked list, where each element (node) contains a reference to the next and previous node.
- It allows for efficient insertions and deletions at any position.

Time Complexity

- **Access:** O(n)
- **Search:** O(n)
- **Insert/Delete:** O(1)

Business Case

LinkedList<T> is suitable for scenarios where frequent insertions and deletions are required.

Example

```
LinkedList<int> linkedList = new LinkedList<int>();
linkedList.AddFirst(1);
linkedList.AddFirst(2);
linkedList.AddLast(3);
LinkedListNode<int> node = linkedList.Find(2);
linkedList.AddAfter(node, 7);
foreach (int item in linkedList)

    Console.WriteLine(item);
```

4. Stack<T>

Structure

- Stack<T> is a last-in-first-out (LIFO) collection.
- Elements are added and removed from the same end.

Time Complexity

- **Access:** O(n)
- **Search:** O(n)
- **Insert/Delete:** O(1)

Business Case

Stack<T> is useful for scenarios like backtracking algorithms, expression evaluation, or maintaining a history of actions (undo functionality).

Example

```
Stack<int> stack = new Stack<int>();
stack.Push(1);
stack.Push(2);
stack.Push(3);
stack.Push(4);

Console.WriteLine(stack.Peek());

Console.WriteLine(stack.Pop());
```

5. Queue<T>

Structure

- Queue<T> is a first-in-first-out (FIFO) collection.
- Elements are added at one end and removed from the other.

Time Complexity

- **Access:** O(n)
- **Search:** O(n)
- **Insert/Delete:** O(1)

Business Case

Queue<T> is suitable for scenarios like task scheduling, where order of processing is important, such as printing tasks or handling requests in a web server.

Example

```
Queue<int> queue = new Queue<int>();
queue.Enqueue(1);
queue.Enqueue(2);
queue.Enqueue(3);
queue.Enqueue(4);
Console.WriteLine(queue.Peek());

Console.WriteLine(queue.Dequeue());
```