**Pilha (Stack)**

Utilizando um array como estrutura de armazenamento:

* **push(T elemento)**
  + **Tempo:** O(1)  
    Basta incrementar o índice do topo e inserir o elemento; não há iterações.
  + **Espaço:** O(1) por operação, considerando que o array já foi alocado com capacidade fixa (alocação total O(n)).
* **pop()**
  + **Tempo:** O(1)  
    Apenas acessa o elemento do topo e decrementa o índice.
  + **Espaço:** O(1) por operação (a estrutura principal já foi alocada previamente).

**Fila (Queue)**

Implementação com array circular:

* **enqueue(int elemento)**
  + **Tempo:** O(1)  
    Calcula a posição circular e insere o elemento.
  + **Espaço:** O(1) por operação, com a alocação total do array sendo O(n).
* **dequeue()**
  + **Tempo:** O(1)  
    Remove o elemento da frente e atualiza o índice.
  + **Espaço:** O(1) por operação.
* **front()**
  + **Tempo:** O(1)  
    Retorna o elemento na posição do índice frontal.
  + **Espaço:** O(1).
* **rear()**
  + **Tempo:** O(1)  
    Retorna o elemento na posição do índice traseiro.
  + **Espaço:** O(1).

**Lista Encadeada**

Cada nó é alocado dinamicamente e as operações podem variar conforme a posição:

* **push(Node node)**
  + **Tempo:** O(1)  
    Insere o nó no início da lista, alterando o ponteiro do head.
  + **Espaço:** O(1) por operação (alocação de um novo nó, total O(n) para n nós).
* **pop()**
  + **Tempo:** O(1)  
    Remove o nó do início e atualiza o head.
  + **Espaço:** O(1).
* **insert(int index, Node node)**
  + **Tempo:** O(n)  
    No pior caso, é necessário percorrer a lista até a posição *index - 1*.  
    (Caso o índice seja 0, é O(1), mas no pior caso, inserir no final, percorre toda a lista.)
  + **Espaço:** O(1) por operação (alocação de um novo nó).
* **remove(int index)**
  + **Tempo:** O(n)  
    No pior caso, percorre a lista até o nó anterior ao que será removido.
  + **Espaço:** O(1).
* **elementAt(int index)**
  + **Tempo:** O(n)  
    Percorre a lista do início até o índice desejado.
  + **Espaço:** O(1).