**Benchmark entre Estruturas de Fila com Prioridade (TypeScript)**

**1. Introdução**

Este documento apresenta uma análise comparativa entre duas implementações de estruturas de dados do tipo fila com prioridade, escritas em TypeScript.

O primeiro código (denominado github.ts) é uma implementação genérica e didática, encontrada no GitHub.

O segundo código está inserido no contexto de uma API hospitalar (backend), com lógica orientada ao domínio da triagem médica.

A análise considera quatro aspectos principais: tamanho, complexidade estrutural, eficiência e aplicação.

**2. Tamanho dos Códigos**

Código 1 - github.ts:

Aproximadamente 91 linhas de código.

Contido em um único arquivo.

Ocupa cerca de 2.2 KB.

Código 2 - backend (foco na lógica de filas):

Aproximadamente 140 linhas relevantes (excluindo controladores e endpoints).

Distribuído em dois arquivos principais: Queue.ts (classe abstrata + filha) e createNode.ts (nó personalizado).

Ocupa cerca de 3.7 KB.

**3. Complexidade Estrutural**

Código 1 - github.ts:

Usa uma única classe PriorityQueue<T> com encadeamento de nós (Node<T>).

Foco genérico, implementação clara e direta.

Estrutura linear e fácil de entender para estudos e testes.

Código 2 - backend:

Aplica herança: Queue<T> é abstrata e ConsultQueue implementa a lógica concreta.

Estrutura baseada em triagem médica real, com inserção customizada segundo níveis de urgência.

O nó (NodeConsult) carrega atributos clínicos como nome, tempo, categoria, e deadline.

Maior acoplamento entre os dados da fila e a lógica hospitalar.

**4. Eficiência (Complexidade Assintótica)**

Ambas as implementações seguem a lógica típica de uma fila com prioridade baseada em lista encadeada:

Inserção (enqueue ou insertQueue):

Complexidade O(n), já que percorre a lista para encontrar a posição correta.

Remoção (dequeue ou callNext):

Complexidade O(1), removendo diretamente o primeiro elemento.

Diferencial:

No github.ts, o método size() percorre toda a fila (O(n)).

No backend, a quantidade é controlada por um contador (qtyPatients), permitindo consulta em O(1).

**5. Foco e Aplicação**

Código 1 - github.ts:

Excelente para fins educacionais, prototipagem e testes.

Tipagem genérica e uso simples.

Código 2 - backend:

Estrutura robusta e aderente à realidade hospitalar.

Lógica de triagem avançada (com categoria, tempo limite e prioridade máxima).

Útil para sistemas reais de gestão de pacientes em filas médicas.

**6. Conclusão**

A comparação evidencia dois tipos distintos de abordagens:

O código do GitHub oferece simplicidade, leveza e clareza, ideal para aprendizado ou usos genéricos.

O código do Backend hospitalar traz complexidade controlada e realismo, adequado para sistemas que exigem fidelidade a regras de negócio.

**Benchmark Comparativo: github.ts vs Backend/API**

1. **Tamanho**

|  |  |  |
| --- | --- | --- |
| **Critério** | **github.ts**  **(Canditado 1)** | **Backend**  **(Canditado 2)** |
| Linhas | ~91 | ~140 (com NodeConsult) |
| Arquivos | 1 só | 2 (Queue + Node) |
| Bytes | ~2.2 KB | ~3.7 KB |

1. **Complexidade Estrutural**

|  |  |  |
| --- | --- | --- |
| **Critério** | **github.ts** | **Backend** |
| Organização | Classe única  (PriorityQueue) | Herança com classe abstrata |
| Abstração | Moderada | Alta (extends,  encapsulamento) |
| Flexibilidade | Alta (genérico ) | Alta (orientado ao domínio real) |
| Dependência  de contexto | Nenhuma | Usa enums,datas e regras de negócio |
| Clareza de leitura | Muito Alta | Média (mais verboso e especifico) |
| Modularidade | Sim | Sim |

1. **Eficiência (Big-O)**

|  |  |  |
| --- | --- | --- |
| **Operação** | **github.ts** | **Backend** |
| enqueue() | O(n) (linear scan) | O(n) (scan + inserção lógica) |
| dequeue() | O(1) | O(1) |
| peek() | O(1) | getFirst() = O(1) |
| size() | O(n) | getQty() = O(1) |

1. **Foco e Aplicação**

|  |  |  |
| --- | --- | --- |
| **Critério** | **github.ts** | **Backend** |
| Propósito | Didático e genérico | Aplicação real (fila hospitalar com triagem) |
| Suporte a prioridade  crítica | Sim | Sim (com  maxPriority) |
| Customização da lógica | Não | Sim (com regras de triagem,datas,etc.) |
| Facilidade de manutenção | Alta | Média(mais lógica acoplada) |

1. **Conclusão e Veredito**

|  |  |
| --- | --- |
| **Critério** | **Melhor Código** |
| Simplicidade | github.ts |
| Eficiência geral | Empate (cada um com vantagens distintas) |
| Clareza e didática | github.ts |
| Modelagem real | Backend (usa categorias,tempos e prioridades clinicas) |
| Escalabilidade lógica | Backend (mais extensível a novos critérios |
| Tamanho/leveza | github.ts |