## **📄 1️⃣ Como ordenar pedidos considerando múltiplos critérios**

### **🎯 Objetivo:**

Calcular uma **prioridade composta** para cada pedido com base em priorityScore, dispatchWindow e sizeCategory, e inseri-los em um Min-Heap para garantir que o mais prioritário esteja no topo.



📊 **1️⃣ Diagrama — Como ordenar pedidos considerando múltiplos critérios**

****

## 

## 

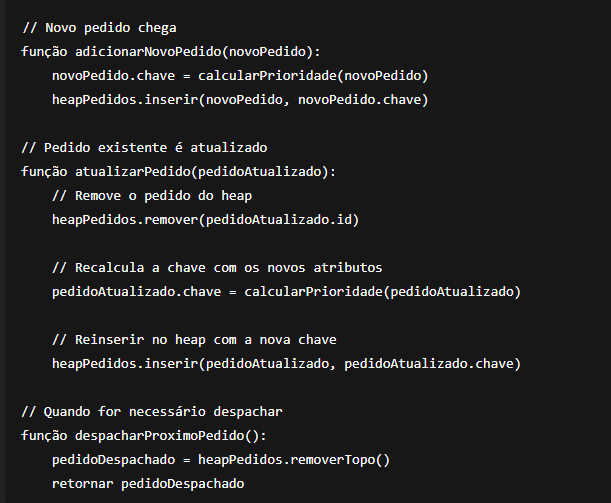
## 

## **📄 2️⃣ Como re‑enfileirar pedidos à medida que novos chegam ou prazos mudam**

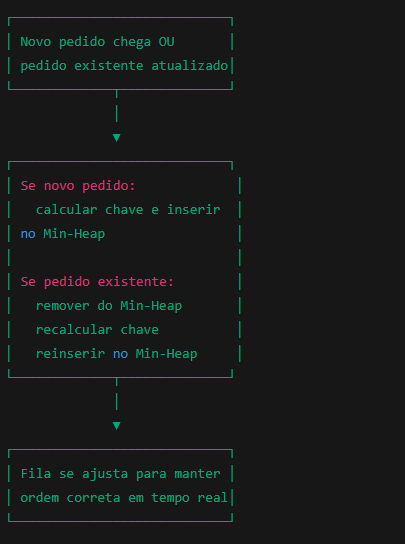
### **🎯 Objetivo:**

Atualizar a fila quando:

* Um novo pedido chega.
* Um pedido existente muda de atributos (por exemplo, porque o tempo restante diminuiu ou prioridade mudou).



# **📊 2️⃣ Diagrama — Como re‑enfileirar pedidos à medida que novos chegam ou prazos mudam**



Descrição textual do algoritmo/estrutura escolhida (pseudocódigo opcional).

Requisitos mínimos da solução

Descrição textual do algoritmo/estrutura escolhida (pseudocódigo opcional).

Diagrama (heap, fluxo, ou gráfico de Gantt) explicando o processo de inclusão, remoção e repriorização.

Justificativa dos critérios de ordenação e efeitos sobre métricas logísticas (p.ex. atraso médio, throughput/hora).

Mapa de trade‑offs (simplicidade × desempenho × manutenibilidade).

Considere o modelo abaixo que descreve as sessões que devem ter nenhum documento a ser entregue como solução para o projeto.

Seção Guia de conteúdo

Problema Situação real retomada em 3–4 linhas; dados de contexto.

Algoritmo / Estrutura Nome, motivação e breve pseudocódigo.

Diagrama Imagem autoexplicativa; use legenda e alto contraste.

Compensações 3 prós e 3 contras; justificativa de escolha.

Próximos passos Ideias futuras (ex.: incorporar aprendizagem online).

**Problema**

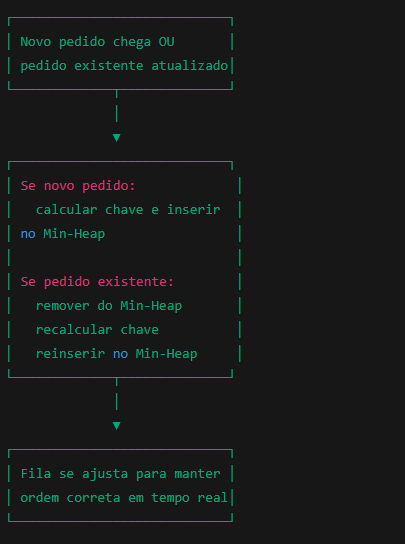
A VelozMart lançou uma funcionalidade para exibir em tempo real os pedidos aguardando expedição. Cada pedido tem um PriorityScore (0–100), tempo restante até expiração do prazo (dispatchWindow) e tamanho (P, M, G). O desafio é ordenar dinamicamente os pedidos para reduzir atrasos e otimizar a operação logística.

**Algoritmo / Estrutura**

Min-Heap. A estrutura central é um heap mínimo que ordena os pedidos com base em uma função de custo composta, considerando:

custo(pedido) = α × (1 / dispatchWindow) + β × (1 / PriorityScore) + γ × sizePenalty

**Diagrama**



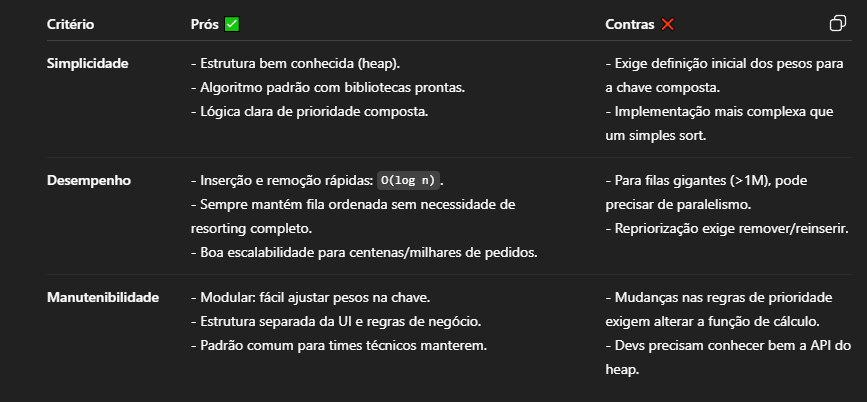
**Trade-offs**

**Prós:**

* Suporta múltiplos critérios de forma flexível (basta ajustar os pesos).
* Permite inserção e remoção rápidas de pedidos em tempo real (O(log n)).
* Escala bem para grandes volumes de pedidos sem perder desempenho.

**Contras:**

* Não é in‑place: utiliza memória adicional para manter a estrutura do heap.
* Não é estável por padrão (empates podem alterar a ordem original).
* Requer calibrar bem os pesos para refletir corretamente as prioridades logísticas.



## **Como ordenar pedidos considerando múltiplos critérios?**

Para ordenar pedidos levando em conta múltiplos critérios como **priorityScore**, **dispatchWindow** e **sizeCategory**, utilizamos um conceito chamado **Priority Key**, que transforma esses fatores em um único número de prioridade ponderada.  
 Esse número é calculado atribuindo pesos aos critérios, por exemplo:

priorityKey = α × (100 - priorityScore) + β × dispatchWindow + γ × sizeWeight

Depois, os pedidos são gerenciados dinamicamente em uma **fila de prioridade implementada com Min‑Heap**, garantindo que o pedido mais urgente esteja sempre no topo da fila para expedição.

## 

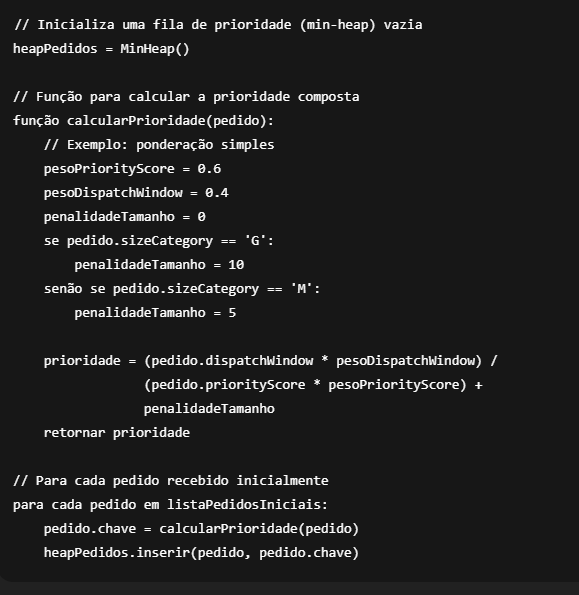
## 

## **Complexidade Assintótica (Big O)**

* Inserção de pedido: **O(log n)**
* Remoção do pedido mais prioritário: **O(log n)**
* Atualização de prioridade: **O(log n)**

## **In‑Place?**

* **Não é in‑place**, pois usa uma estrutura de dados adicional (o heap).





## **Como re‑enfileirar pedidos à medida que novos chegam ou prazos mudam?**

A estrutura de dados **Min‑Heap com Priority Key** facilita a re‑enfileiração dinâmica de pedidos.  
 Sempre que um novo pedido chega ou um pedido existente tem algum de seus critérios atualizados (como dispatchWindow ou priorityScore), recalculamos sua **priorityKey** e o re‑inserimos na fila com a nova prioridade.

### **Como funciona:**

* Para **novo pedido**:  
  + Calcula sua priorityKey com base nos critérios atuais.
  + Insere no Min‑Heap em O(log n).
* Para **pedido com prioridade alterada**:  
  + Remove o pedido do Min‑Heap em O(log n).
  + Recalcula sua priorityKey.
  + Insere novamente no Min‑Heap em O(log n).

Dessa forma, a fila é sempre mantida ordenada pela prioridade composta, refletindo o estado mais recente de cada pedido.

### **🔷 Pontos Positivos**

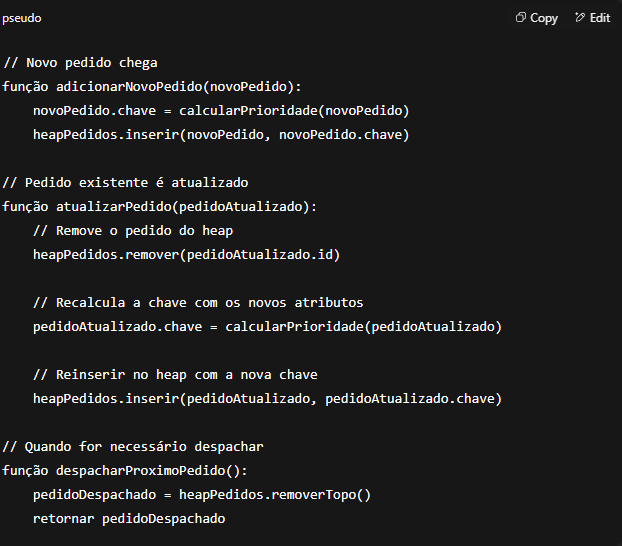
* ✅ Rápido e eficiente para atualizações frequentes.
* ✅ Não é necessário reordenar a lista inteira.
* ✅ Suporta alta taxa de chegadas e mudanças em tempo real.

### **🔷 Pontos Negativos**

* ❌ Requer atenção para evitar inconsistências (por exemplo, esquecer de remover antes de inserir novamente).
* ❌ Sobrecarga ligeiramente maior para sistemas muito pequenos.
* ❌ Complexidade de implementação maior que algoritmos simples.

## **📊 Complexidade Assintótica (Big O)**

* Atualização ou inserção de pedido: **O(log n)**
* Remoção do mais prioritário: **O(log n)**

****

## **📌 Quais métricas comprovam que a estratégia é melhor que uma ordenação simples por tempo?**

A ordenação por múltiplos critérios usando **Min‑Heap com Priority Key** é superior à ordenação simples por dispatchWindow (tempo restante) porque considera **urgência real + impacto logístico + valor estratégico**.

Para comprovar essa superioridade, podemos analisar as seguintes métricas:

### **📊 1. Atraso médio (Average Delay)**

* Mede o tempo médio de atraso dos pedidos em relação ao SLA.
* 🔺 Estratégia simples (ordenar só por tempo) pode priorizar um pedido barato e leve antes de outro mais importante.
* ✅ Com Priority Key, pedidos de maior valor logístico são despachados antes, **reduzindo o atraso médio dos mais relevantes**.

### **📈 2. Throughput por hora (Pedidos/hora expedidos)**

* Avalia quantos pedidos são processados por hora.
* ✅ Ao evitar o acúmulo de pedidos grandes e difíceis no final da fila, a ordenação inteligente distribui melhor os volumes, **aumentando o ritmo geral de expedição**.

### **💰 3. Valor médio expedido por hora**

* Soma do valor total dos pedidos expedidos dividido pelo tempo.
* ✅ Ao considerar o priorityScore (que inclui valor e reputação), a estratégia garante que **pedidos mais valiosos sejam priorizados**, elevando o retorno logístico.

### **📦 4. Uso eficiente da doca (Balanceamento de tamanhos)**

* Ordenar apenas por tempo pode concentrar muitos pedidos grandes em sequência.
* ✅ Com ponderação por sizeCategory, a estratégia prioriza de forma balanceada, **evitando gargalos físicos na doca de embalagem**.

### **🧪 Como comprovar essas métricas?**

* Criar um teste A/B ou simulação com dois grupos:  
  + Grupo A: ordenação apenas por tempo.
  + Grupo B: ordenação por Min‑Heap com Priority Key.
* Coletar dados reais ou simulados por algumas horas/dias e comparar as métricas acima.

## **📄 Justificativa dos critérios de ordenação e efeitos sobre métricas logísticas**

Os critérios escolhidos — **priorityScore**, **dispatchWindow** e **sizeCategory** — refletem dimensões essenciais para a operação logística do VelozMart:

* **priorityScore**: já incorpora fatores como SLA, valor do item, distância e reputação do vendedor. Ele garante que pedidos de maior valor estratégico para o negócio tenham prioridade.
* **dispatchWindow**: representa a urgência temporal para evitar atrasos. Priorizar janelas mais curtas reduz o número de pedidos fora do SLA.
* **sizeCategory**: considera o impacto no espaço físico das docas. Penalizar pedidos muito grandes ajuda a manter um fluxo constante, evitando gargalos na operação.

Esses critérios combinados em uma chave composta equilibram eficiência comercial (prioridade estratégica) e operacional (minimizar atrasos e otimizar uso do espaço).

### **📊 Efeitos sobre métricas logísticas:**

* 📉 **Atraso médio**: Reduzido ao priorizar pedidos com janelas mais curtas e maior impacto estratégico.
* 🚀 **Throughput por hora**: Aumentado pela fluidez na ocupação das docas e menor tempo ocioso.
* 📈 **% de pedidos dentro do SLA**: Melhorado pelo foco em janelas críticas.
* 🔄 **Previsibilidade operacional**: A fila se adapta dinamicamente a mudanças nos pedidos sem necessidade de intervenção manual constante.

Essa abordagem é superior à ordenação simples por tempo, pois leva em conta múltiplos objetivos simultaneamente, maximizando resultados globais.

🔷 **Mapa de Trade-offs: Simplicidade × Desempenho × Manutenibilidade**