**Exemplo AppWeb: To-Do List com Ranking**

**1. Informações Gerais**

* **Centro Universitário**: Wyden UniRuy
* **Docente**: Professor MSc Eng Heleno Cardoso
* **Aluno(s):** Julia Cardoso
* **Disciplina:** Algoritmos e Complexidade 2025.2
* **Projeto Web:** To-Do List com Ranking
* **Repositório GitHub:** [github.com/joaosilva/todolist-ranking](https://github.com/joaosilva/todolist-ranking)
* **Link de Deployment:** <https://todolist-ranking.vercel.app>

**2. Descrição do Projeto**

* **Objetivo da aplicação:**

Permitir que múltiplos usuários criem, editem e concluam tarefas, e gerar um ranking com os usuários que completaram mais tarefas.

* **Funcionalidades principais:**
  1. Cadastro e login de usuários.
  2. CRUD de tarefas (criar, editar, concluir, excluir).
  3. Visualização do ranking de usuários por tarefas concluídas.
  4. Busca de tarefas por palavra-chave.
* **Tecnologias utilizadas:**
  1. **Frontend**: React.js + Tailwind CSS
  2. **Backend**: Node.js + Express
  3. **Banco de dados**: MongoDB
  4. **Deployment**: Vercel (frontend) + Render (backend)

**3. Estruturas de Dados**

| **Rotina/Função** | **Est. Dados** | **Justificativa do Uso** |
| --- | --- | --- |
| Cadastro de usuários | **Hash Table** | **Busca rápida** pelo ID do usuário |
| Armazenamento de tarefas | **Array/List** | **Ordenação** e iteração simples das tarefas |
| Ranking de usuários | **Heap/Min-Heap** | Permite **manter ranking** top N de forma eficiente |
| Busca de tarefas | **Array/List** | Iteração simples e **filtragem com palavra-chave** |

**4. Algoritmos Implementados**

* **Ordenação:**
  + QuickSort para ordenar usuários por tarefas concluídas no ranking.
  + Complexidade:
    - Melhor caso: O(n log n)
    - Caso médio: O(n log n)
    - Pior caso: O(n²)
* **Busca:**
  + Busca linear nas tarefas por palavra-chave.
  + Complexidade: O(n), pois precisa percorrer todas as tarefas do usuário.
* **Outros algoritmos relevantes:**
  + Atualização do ranking usando heap para inserir/remover usuários com eficiência O(log n).

**5. Recursividade e Equações de Recorrência**

| **Rotina Recursiva** | **Objetivo** | **Equação de Recorrência** | **Complexidade** |
| --- | --- | --- | --- |
| QuickSort | Ordenar ranking de usuários | **T(n)** = **2T(n/2) + O(n)** | O(**n log n**) |

A função **QuickSort** foi implementada para ordenar usuários por tarefas concluídas. A recursão divide o **array** de usuários em **subarrays** até atingir tamanho 1 e combina os resultados.

**6. Técnicas de Programação Aplicadas**

* **Divisão e Conquista:**
  + Usada no **QuickSort** para ordenar o **ranking** de usuários.
* **Estruturas Avançadas (Heap):**
  + **Min-Heap** utilizada para manter ranking top N eficiente.

Não foram utilizadas programação dinâmica ou algoritmos gulosos neste projeto.

**7. Complexidade Assintótica**

| **Rotina/Função** | **Caso Melhor** | **Caso Médio** | **Caso Pior** | **Notação Big O/Θ/Ω** |
| --- | --- | --- | --- | --- |
| Cadastro usuário | O(1) | O(1) | O(1) | O(1) |
| Criação de tarefa | O(1) | O(1) | O(1) | O(1) |
| Conclusão de tarefa | O(1) | O(1) | O(1) | O(1) |
| Busca de tarefa | O(n) | O(n) | O(n) | O(n) |
| Ordenação ranking (QuickSort) | O(n log n) | O(n log n) | O(n²) | O(n log n) |
| Atualização ranking (Heap) | O(log n) | O(log n) | O(log n) | O(log n) |

**Observações**: A maior parte das rotinas CRUD tem complexidade O(1), enquanto a ordenação do ranking representa o principal custo computacional quando há muitos usuários.

**8. Observações Finais: Desafios/Melhorias**

* O uso de **Min-Heap** no **ranking** permitiu manter eficiência mesmo com crescimento do número de usuários.
* **QuickSort** provê ordenação rápida, mas em casos extremos (**arrays quase ordenados**) o tempo pode degradar.
* **Futuras melhorias**: Implementar **busca indexada para tarefas**, adicionar filtros por data ou prioridade, e usar cache para **ranking**.