12 - Resolução:

- A implementação inicial (algoritmo base), que não emprega melhorias, levou aproximadamente:

  Tempo de execução (Algoritmo base): **2.2265 segundos**

- Já a versão otimizada (com ordenação e cache) apresentou desempenho significativamente superior:

  Tempo de execução (Algoritmo melhorado): **0.0975996 segundos**

```
lucas-roseno@lucas-roseno-IdeaPad-Gaming-3-15IHU6:~/CEFET/3 periodo/AEDS/Exercicio12_Pratica1$ g++ -std=c++17 -O2 base.cpp -o base
lucas-roseno@lucas-roseno-IdeaPad-Gaming-3-15IHU6:~/CEFET/3 periodo/AEDS/Exercicio12_Pratica1$ ./base
Digite o número de processos (N): 5
Tempo de execução (Algoritimo base): 2.2265 segundos
lucas-roseno@lucas-roseno-IdeaPad-Gaming-3-15IHU6:~/CEFET/3 periodo/AEDS/Exercicio12_Pratica1$ g++ -std=c++17 -O2 melhorias.cpp -o
melhorias
lucas-roseno@lucas-roseno-IdeaPad-Gaming-3-15IHU6:~/CEFET/3 periodo/AEDS/Exercicio12_Pratica1$ ./melhorias
Digite o número de processos (N): 5
Tempo de execução (Algoritimo melhorado): 0.0975996 segundos
```

O que mostra uma diferença de tempo considerável. Se levarmos em consideração apenas a velocidade, e desconsiderar o espaço ocupado, as melhorias de cache e ordenação de leitura são muito expressivas, representadas por:

$$\text{Melhoria} = ((\text{Tbase} - \text{Tmelhorado}) / \text{Tbase}) \times 100$$

$$\text{Melhoria} = ((2.22652.2265 - 0.0975996) / 2.22652.2265) \times 100 \approx 95.62\%$$

Importante: os tempos apresentados não são precisos, pois foram medidos apenas uma vez e sem controle de variáveis externas, como o número de processos sendo executados simultaneamente no sistema operacional.

Além disso, o programa está configurado para funcionar corretamente apenas quando o usuário informa o valor 5, já que o gerador de arquivos foi construído especificamente para criar P1.txt a P5.txt e os arquivos de dados 01.txt a 05.txt.

Ama

1. Uso de Cache

- **Custo computacional:**
  Muito leve. O acesso à estrutura std::unordered_map (tabela hash) é praticamente O(1), mesmo com muitos elementos.
- **Benefício:**
  Evita a releitura e o recálculo da soma das raízes quadradas de arquivos com 100.000 valores, sempre que o mesmo arquivo é referenciado em múltiplas linhas.

Conclusão:
Altíssimo benefício em relação ao custo. O uso de cache é uma otimização de alta efetividade e baixo custo, válida em praticamente qualquer cenário.

2. Ordenação dos Arquivos por Linha

- **Custo adicional:**
  Cada linha executa um std::sort, que tem custo médio de O(M log M), onde M é o número de arquivos referenciados naquela linha.
- **Benefício:**
  A ordenação pode otimizar o acesso sequencial ao disco (especialmente útil em discos rígidos mecânicos - HDDs), e também aumenta a chance de reuso de dados já carregados em cache, ao agrupar arquivos iguais consecutivamente.

Conclusão:
O benefício é moderado, mas o custo é baixo. A ordenação é uma melhoria válida, especialmente em contextos com arquivos grandes ou sistemas de armazenamento mais lentos.