Estruturas de Dados Básicas I

Aula 04 - Análise empírica

Motivação e objetivos

- Motivação
  - Para um mesmo problema, é possível ter diferentes soluções.
  - Porém, apenas uma deve ser implementada...

    ...de preferência a melhor de acordo com o contexto dos dados.
  - Como saber qual é a melhor???

- Objetivos
  - Apresentar uma possível técnica para comparar soluções algorítmicas
  - Identificar as vantagens e os inconvenientes desta técnica

Análise empírica de um algoritmo

- Eficiência

  Considerando que dois algoritmos (A e B) resolvem corretamente um problema, qual deles usa menos recursos?
- Solução simplista (óbvia): para comparar dois algoritmos, basta implementá-los e analisar suas execuções !!! :-)
- Principais comparações
  - Tempo (algoritmo A acha a solução primeiro que B)
  - Uso de memória (algoritmo A requer menos memória que B)
- Também conhecida como profiling ou análise de desempenho
- É possível comparar diferentes implementações do mesmo algoritmo

Como realizar a análise

- Normalmente, o número de recursos utilizados (tempo / memória) dependem dos dados de entrada
- Não basta comparar as execuções para uma única entrada
- Bateria de testes com:
  - Dados aleatórios (bem distribuídos no universo de possibilidades)
  - Tamanhos variados
- Definir um critério de comparação
  - Tempo (contagem de operações básicas e/ou tempo gasto)
  - Memória (tamanho máximo de memória alocada)

Passos (metodologia): prob. busca

I. Escolha dos algoritmos/implementações a serem analisados
   - Busca sequencial, Busca binária (iterativa e recursiva)
II. Definição dos critérios a serem avaliados
   - Tempo: número de passos e tempo de execução
III. Definição da base de testes
   - Tamanhos das amostras: 50 a 10000 (a cada 50)
   - Média de 100 amostras aleatórias para cada tamanho
IV. Executar os algoritmos para todas amostras
V. Compará-los
   - Gráficos (tendências), derivadas, análises estatísticas, etc

Critério de número de passos

- Contagem do número de "instruções básicas"
- Ex: número de execuções de um loop

```
int buscaSeq(int v[], int N, int x, int *count) {
  int i;
  for (i=0; i < N; i++) {
    (*count)++;   /* incrementa contador a cada iteração */
    if (x == v[i])
      return i;
  }
  return -1;
};
```

Critério de número de passos

- Contagem na recursão

```
int buscaBin(int v[], int N, int x, int *count) {
  int i = N/2;
  int y = v[i];
  (*count)++;   /* incrementa contador a cada chamada recursiva */

  if (N == 0)
    return -1;
  else if (x == y)
    return i;
  else if (x < y)
    return buscaBin(v, i, x, count);    /* passa o contador */
  else
    return buscaBin(&v[i+1], N-(i+1), x, count);
}
```

Critério de tempo de execução

- A biblioteca padrão do C fornece algumas funções (ex: `clock()`)
- Porém, a precisão é baixa (~0,1 segundos)
- Ler o tempo com alta-precisão, depende do S.O.
- É necessário uma biblioteca específica para cada um. Exemplo:
  - Windows: <windows.h>
  - Linux: <sys/time.h>
- Procedimento
  - Registra o tempo inicial (ou o clock da CPU)
  - chama o procedimento a ser testado
  - Registra o tempo final e calcula a diferença

Critério de tempo

- Solução no windows

```
#include <windows.h>
    ...
    /* registra o clock inicial e a freq. da CPU */
    LARGE_INTEGER lFreq, lStart, lEnd, ldiff;
    QueryPerformanceFrequency(&lFreq);
    QueryPerformanceCounter(&lStart);

    < operacao >

    /* registra o clock final e calcula a diferença entre os dois */
    QueryPerformanceCounter(&lEnd);
    ldiff = lEnd.QuadPart - lStart.QuadPart;    /* em tick da CPU */
    t = (double)ldiff / (double)lFreq.QuadPart; /* em microsegundos */
```

Critério de tempo

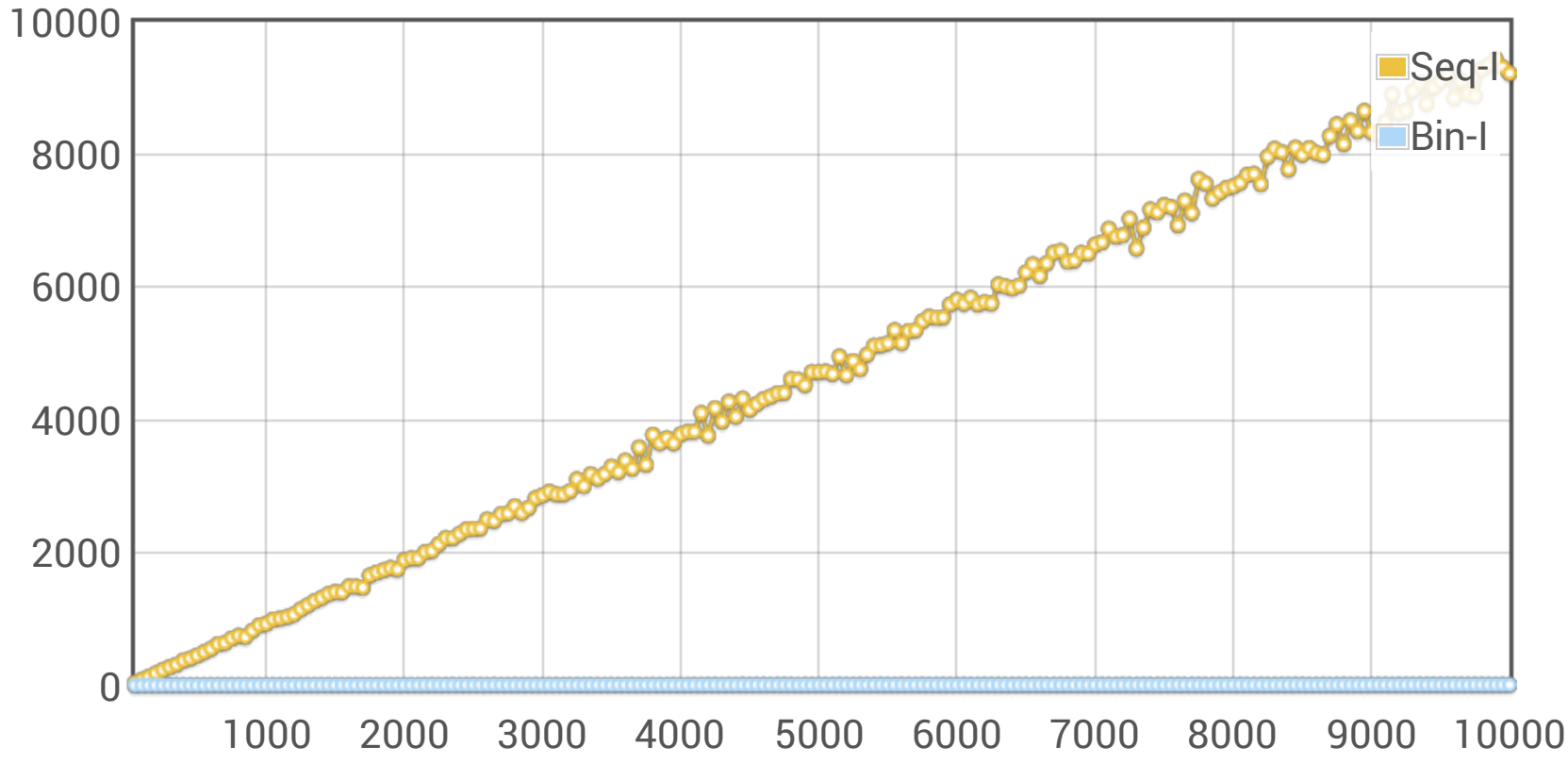
- Solução no linux

```
#include <sys/time.h>
    ...
    /* registra a hora inicial */
    struct timeval t0, t1;
    gettimeofday(&t0, NULL);

    < operacao >

    /* registra a hora final e calcula a diferença entre as duas */
    gettimeofday(&t1, NULL);
    t  = (t1.tv_sec - t0.tv_sec) * 1000000;  /* diferença */
    t += t1.tv_usec - t0.tv_usec             /* em microsegundos */
```

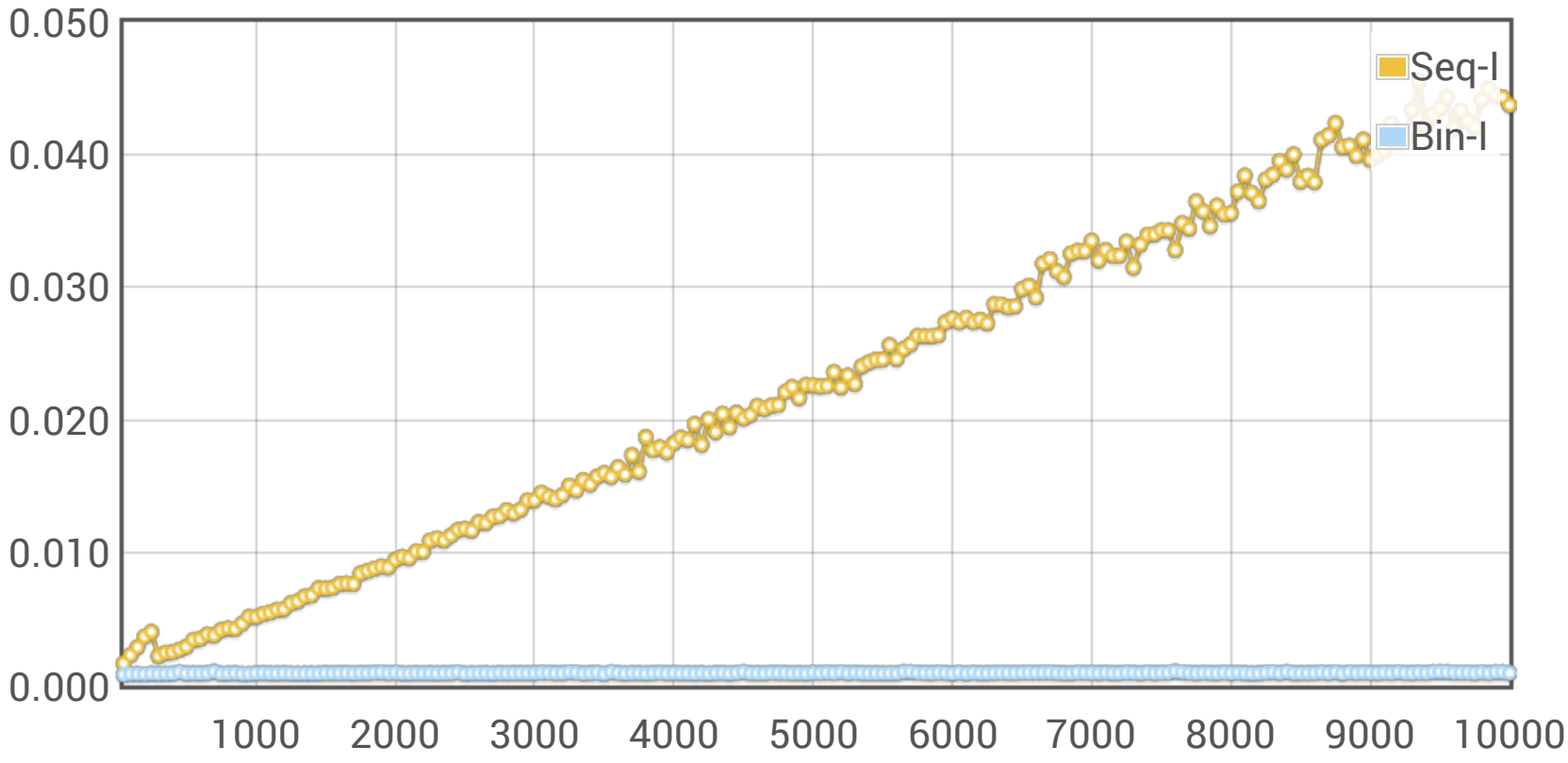
Análise dos resultados

- Critério de comparação
  - Número de passos
  - Tempo de execução
- Comparação gráfica
  - Procedimento iterativo: busca sequencial vs recursiva
  - Procedimento recursivo: (similar)
  - Busca sequencial: iterativa vs recursiva
  - Busca binária: iterativa vs recursiva

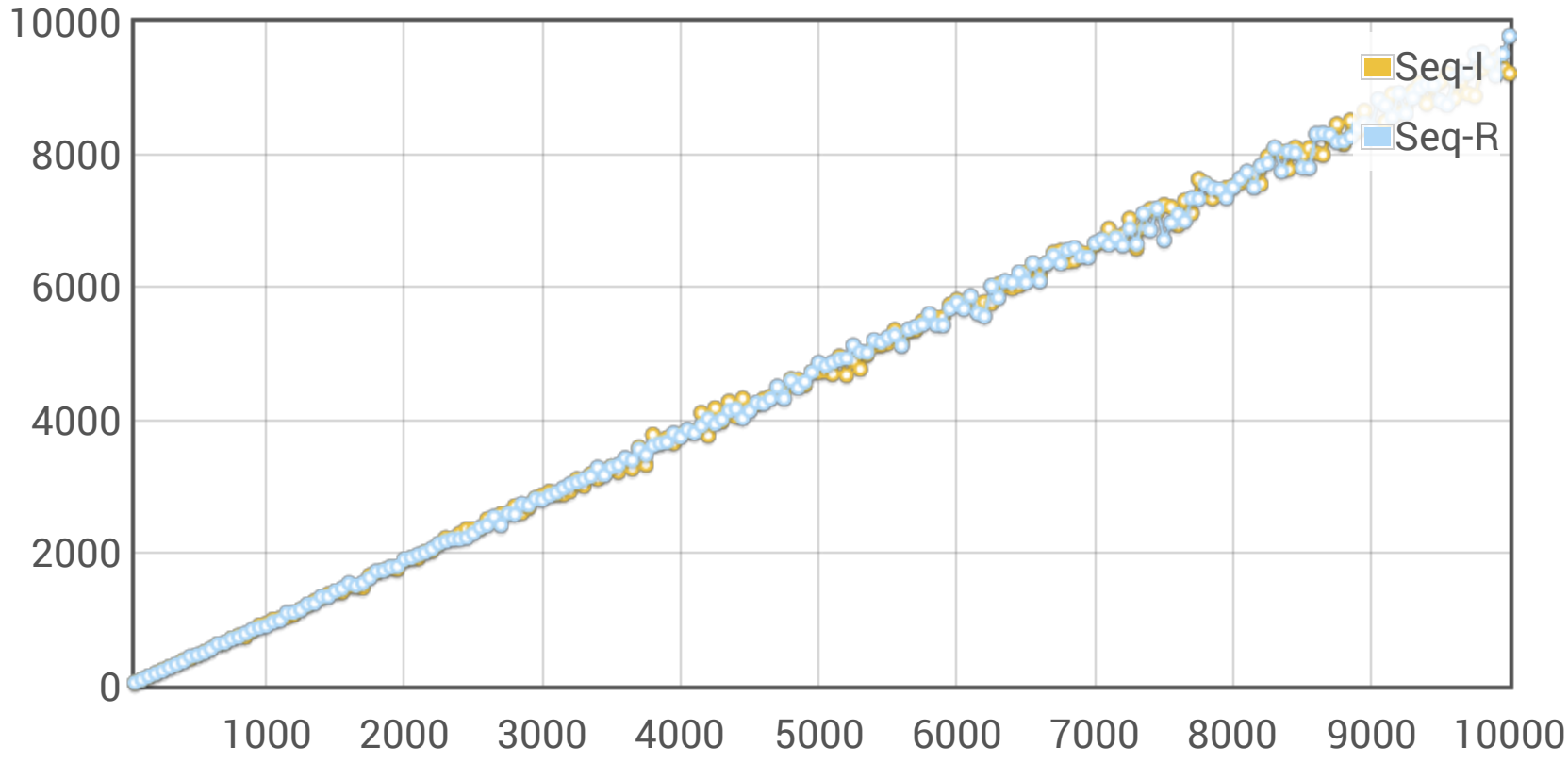
Resultado - num de passos



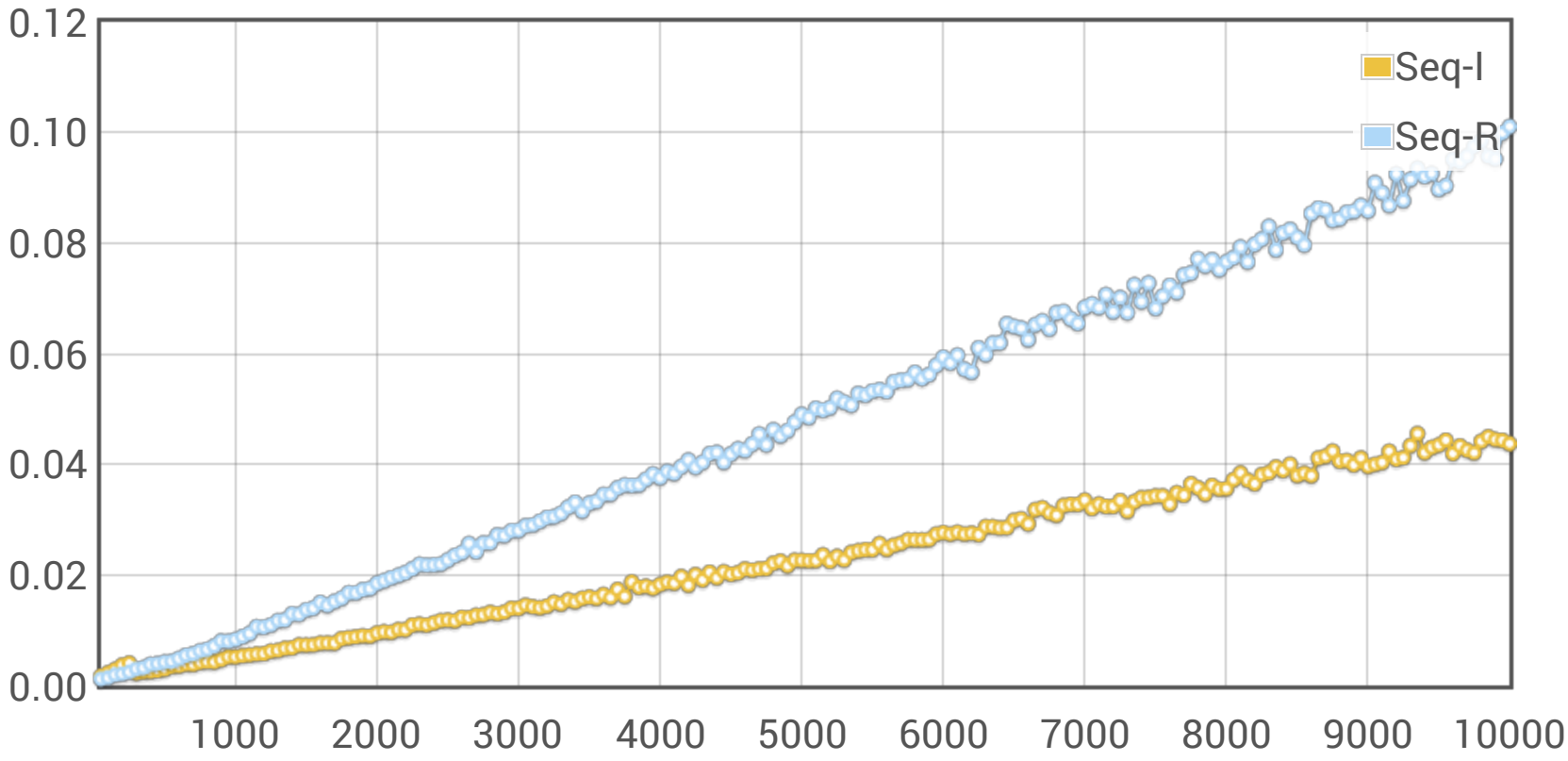
Resultado - tempo de execução



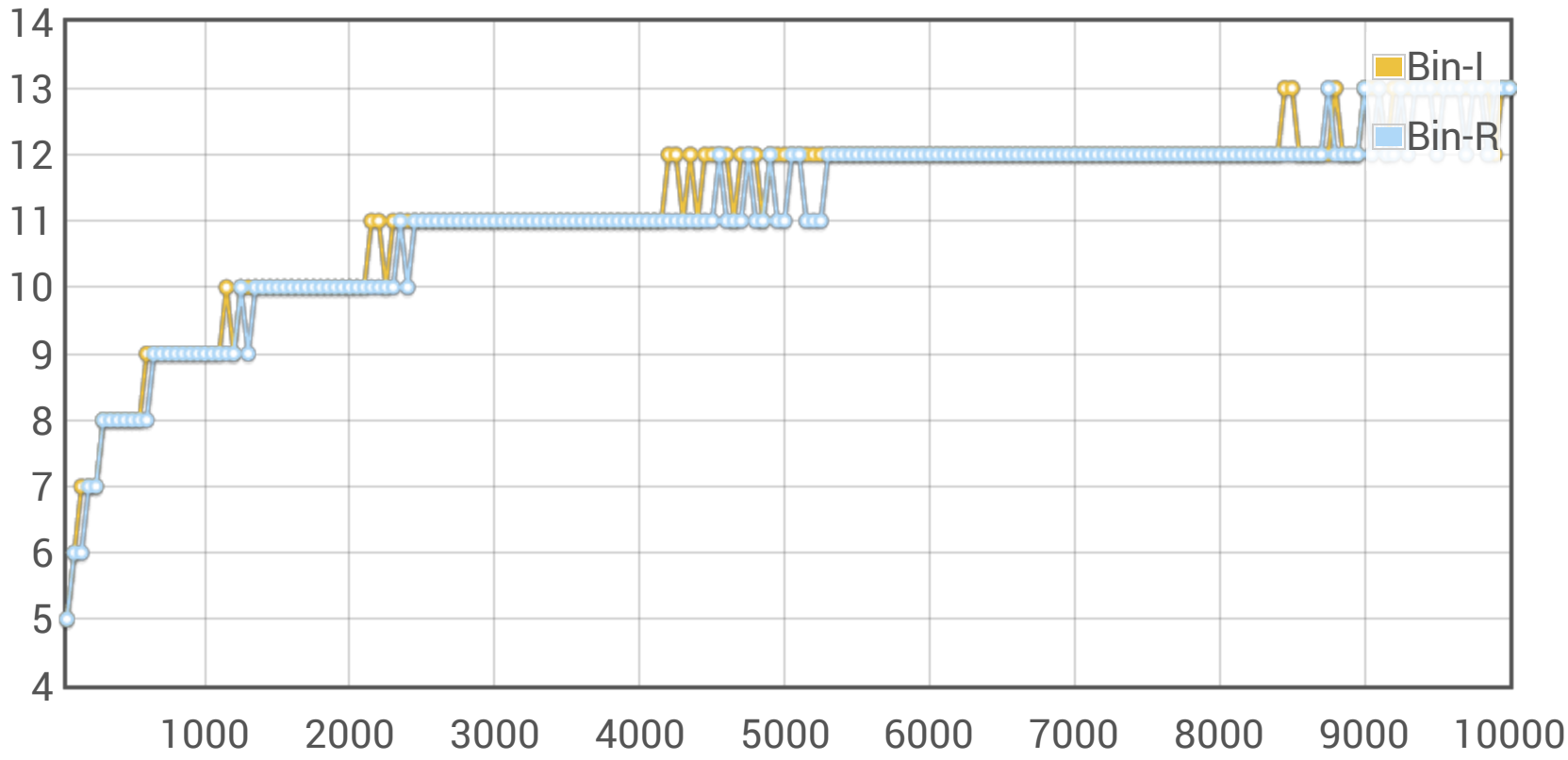
Resultado - num de passos



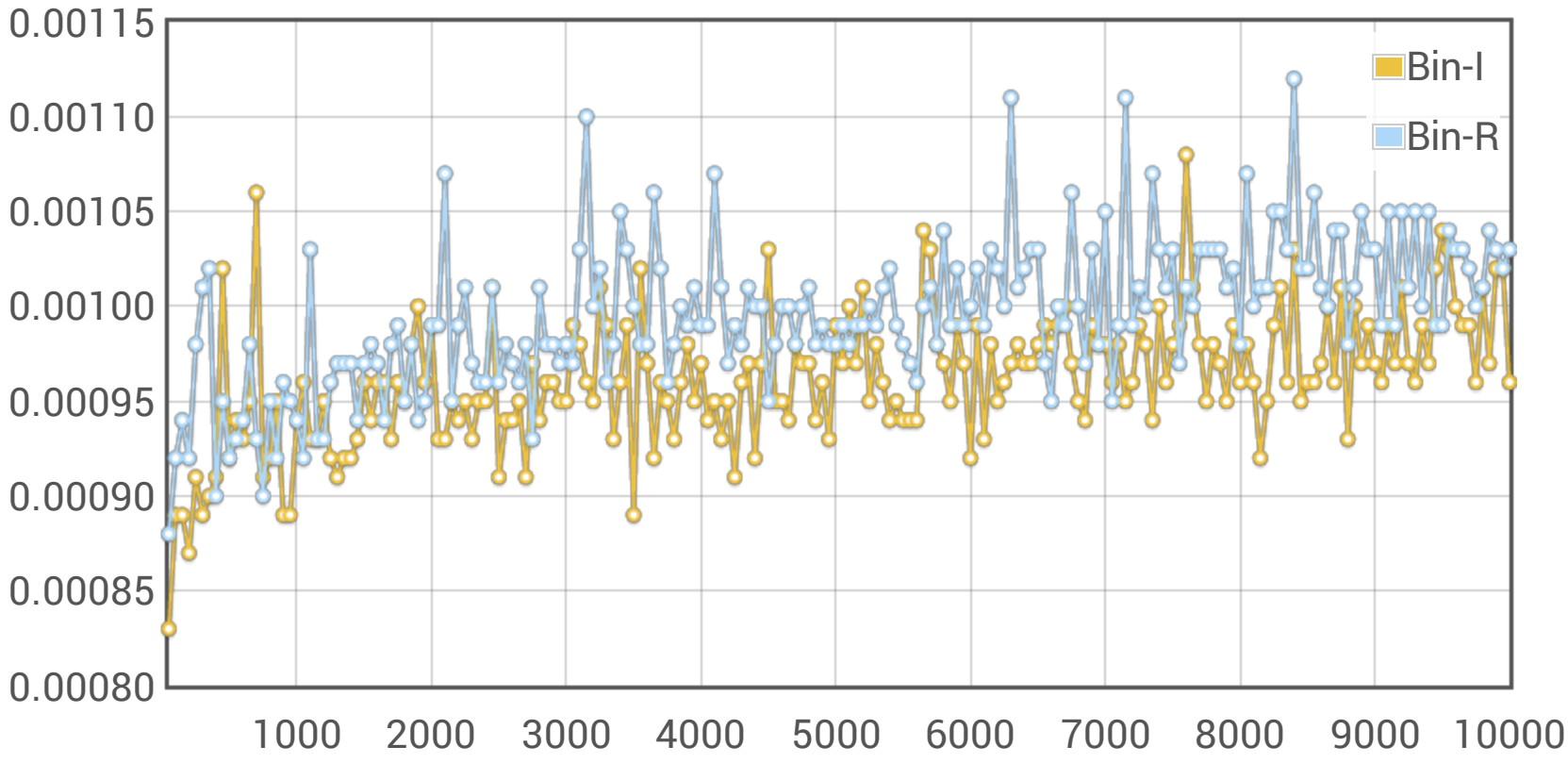
Resultado - tempo de execução



Resultado - num de passos

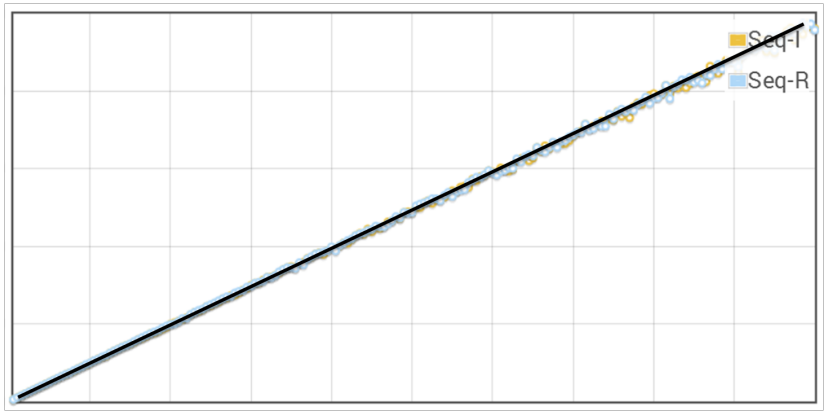


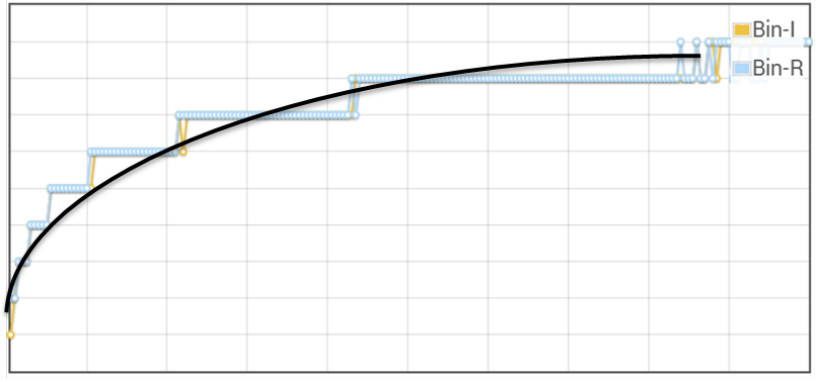
Resultado - tempo de execução



Discussão

- Os dados da avaliação de tempo são "perturbados" por outros processos da CPU
- A contagem de passos é mais precisa
- É possível extrair uma função que represente a curva sem executar todo o procedimento experimental???





Vantagens e desvantagens

- Vantagens
  - Aplicável em qualquer situação / problema / contexto ...
  - Consegue comparar implementações (e não algoritmos)
  - Permite ter uma noção do comportamento dos algortimos sem grandes dificuldades
- Desvantagens
  - Depende das características de onde os testes estão sendo realizados
  - Não permite comparações com outros algoritmos que foram testados em outros computadores

Considerações Finais

- Podemos comparar algoritmos sem precisar implementá-los? (comparando a ideia)
  - Aparentemente, há uma tendência de curva a medida que o tamanho do problema aumenta
  - Se conseguirmos identificar a função da curva, podemos comparar as funções de dois algoritmos diferentes

- Análise da complexidade de um algoritmo (próxima aula)