Trabalho de Algoritmo e Estrutura de Dados I

Aluno: Munir Xavier Wanis
Matrícula: 201320488411

Introdução:

O trabalho tem o objetivo de mostrar a complexidade computacional dos algoritmos de inserção, busca e exclusão das estruturas de lista sequencial ordenada e lista sequencial encadeada em função do número de elementos totais.

Para a realização dos experimentos foram feitos alguns programas para a automatização da execução, foi criado um arquivo “gerador.c” onde ele gera uma lista (dentro de um arquivo “entrada.txt”) aleatória com Matrícula, Idade e Nome, com um tamanho variável, onde o usuário pode mudá-lo.
O arquivo “entrada.txt” vai ser utilizado para inserir os seus elementos na lista encadeada e na lista sequencial.
Mais dois shell scripts para chamar os programas principais e automatizar o processo de geração de arquivo “entrada.txt” e geração de saída da database utilizada para plotar os gráficos.

Os programas de lista ordeanda e encadeada foram executados com dados de até trinta mil alunos (somados de cem em cem), para que fosse gerado arquivos com tempos consideráveis para se traçar um gráfico.

Para executar os programas, abra o terminal vá até o caminho das pastas e dentro da pasta encadeada digite “./autoseqenc.sh” (sem aspas), e na pasta ordenada digite “./autoseqord.sh”.

Análise de complexidade do algoritmo Lista Sequencial Ordenada

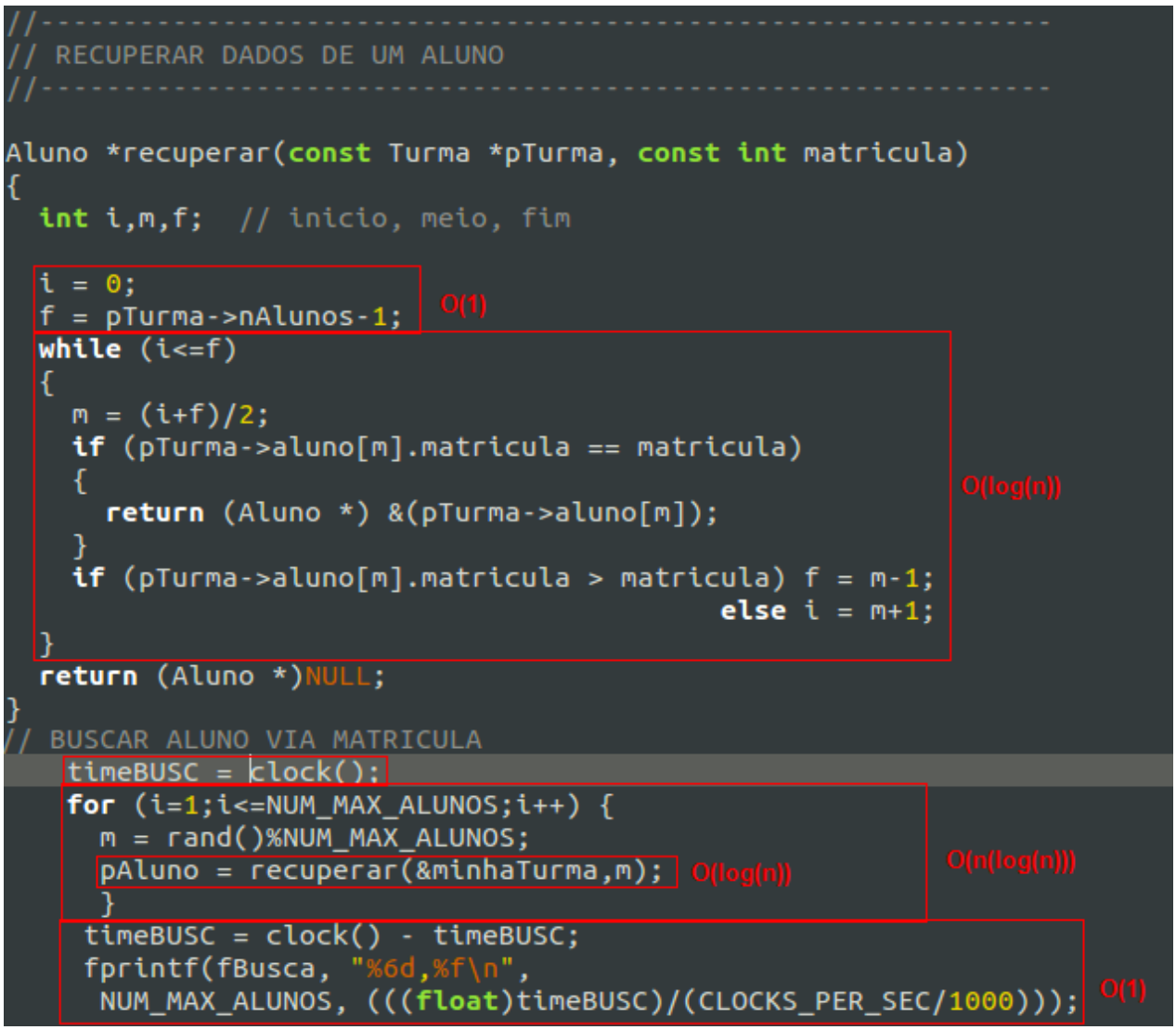
- Algoritmo de Inserção:

```
//---------------------------------------------------------------
// INSERIR UM NOVO ALUNO
//---------------------------------------------------------------
int inserir(Turma *pTurma, const Aluno *pNovoAluno)
{
  int i;
  if (pTurma->nAlunos == NUM_MAX_ALUNOS)
  {
    return 1;  // ERRO 1: Lista Cheia
  }
  i = pTurma->nAlunos - 1;
  while (i >= 0 && pTurma->aluno[i].matricula > pNovoAluno->matricula)
  {
    pTurma->aluno[i+1] = pTurma->aluno[i];
    i--;
  }
  pTurma->aluno[i+1] = *pNovoAluno;
  pTurma->nAlunos++;
  return 0;

}
// INCLUIR ALUNO
    timeINC = clock();
    for (i=1;i<=NUM_MAX_ALUNOS;i++)
        {
          scanf("%d",&(meuAluno.matricula));
          scanf("%d",&(meuAluno.idade));
          scanf("%s",meuAluno.nome);
          resultado = inserir(&minhaTurma,&meuAluno);
        }
    timeINC = clock() - timeINC;
    fprintf(fInsert, "%6d,%f\n",
     NUM_MAX_ALUNOS, (((float)timeINC)/(CLOCKS_PER_SEC/1000)));
```

O(1)
O(n)
O(1)
O(1)
O(1)
O(n²)
O(n)
O(1)

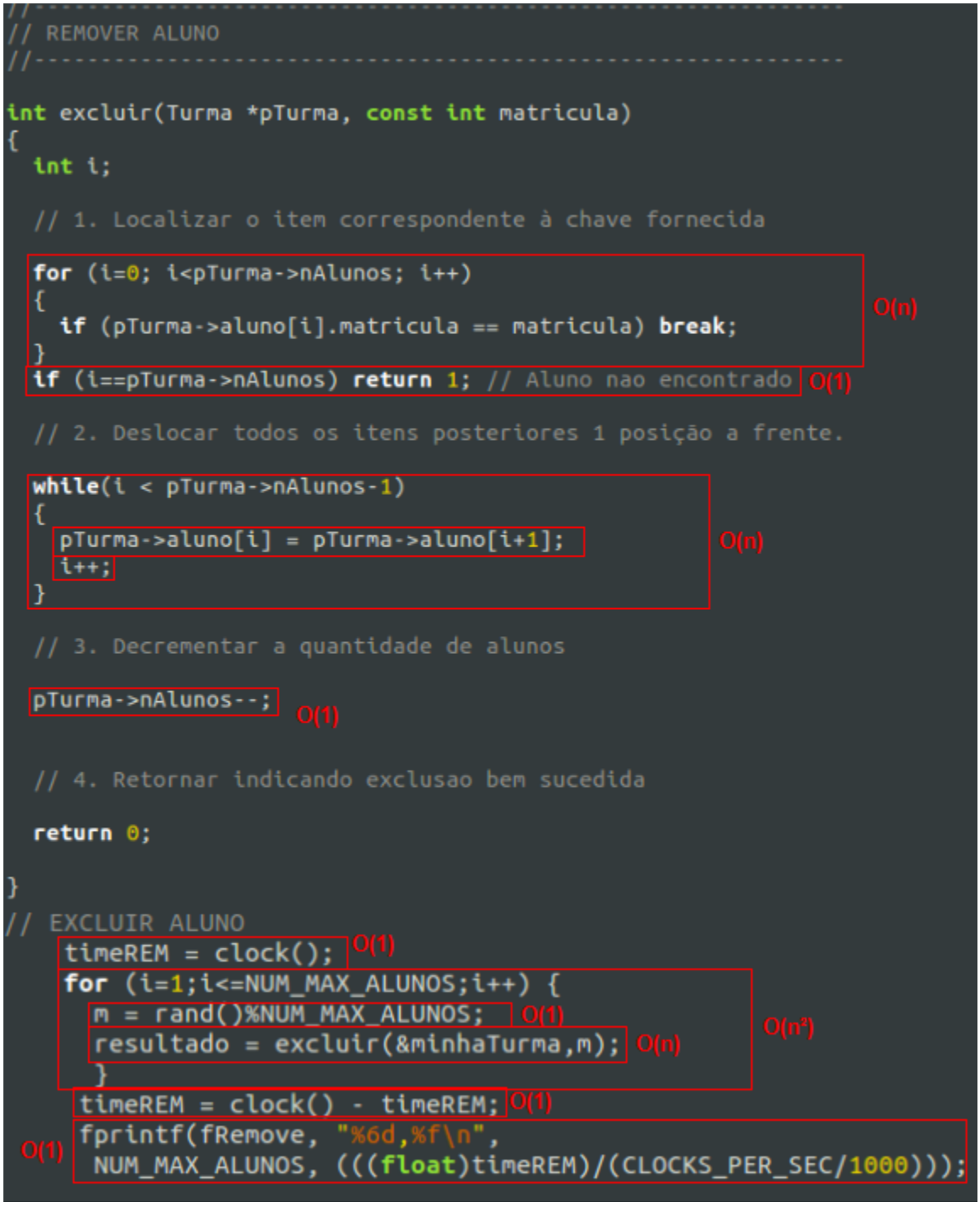
O resultado da análise de complexidade do algoritmo de inserção é $O(n^2)$.

- Algoritmo de Busca:



O resultado da análise de complexidade do algoritmo de busca é O(n(log(n))).

- Algoritmo de Remoção:



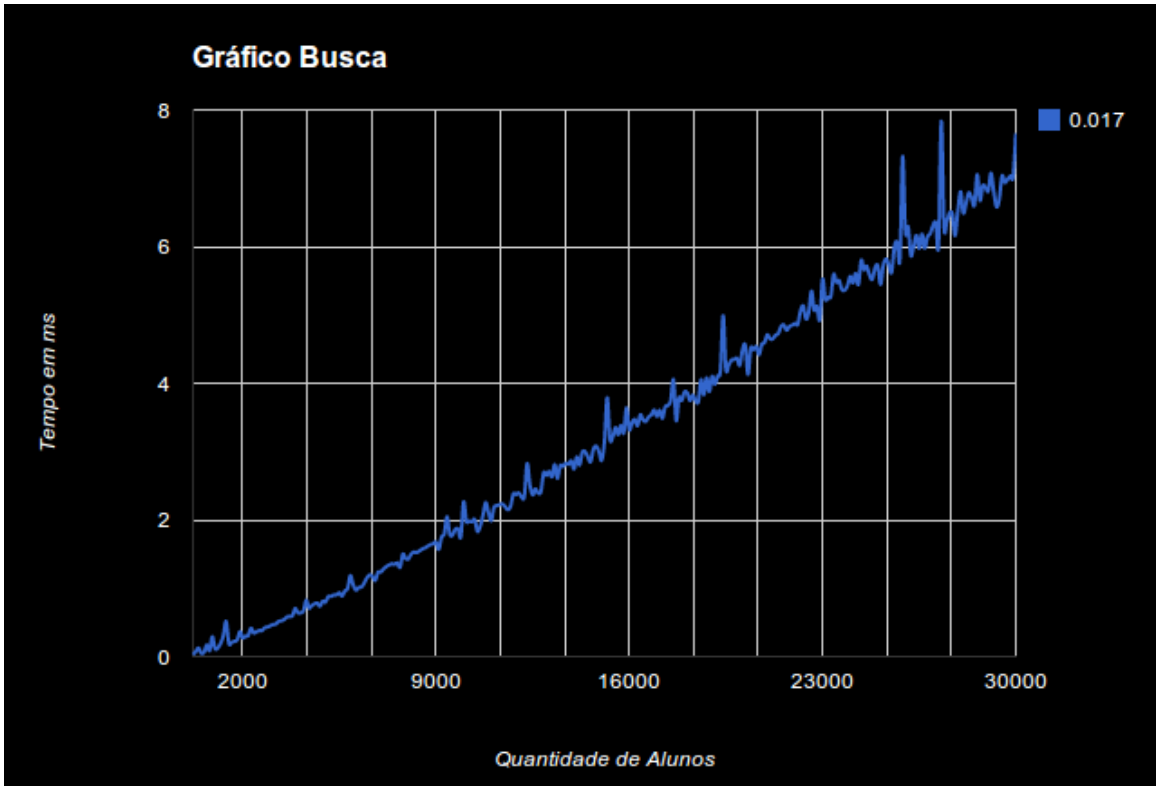
De acordo com a análise de complexidade do algoritmo de remoção, temos que ele é $O(n^2)$.

Gráficos Quantidade x Tempo do algoritmo Lista Sequencial Ordenada:

- Algoritmo de Inserção:

Percebe-se que o gráfico possui um comportamento quadrático.

- Algoritmo de Busca:



Como foi utilizado o algoritmo de busca binária, o tempo foi reduzido, pois o gráfico resultado possui um comportamento n(log(n)).

- Algoritmo de Remoção:

O gráfico de remoção também possui um comportamento quadrático.

Análise de complexidade do algoritmo Lista Encadeada

- Algoritmo de Inserção:

```
//---------------------------------------------------------------
// INSERIR UM NOVO ALUNO
//---------------------------------------------------------------

int inserir(Turma *pTurma, const Aluno *pNovoAluno)
{
  Celula *pNovaCelula;

  pNovaCelula = malloc(sizeof(Celula));                    O(1)

  if (!pNovaCelula) return 1;                              O(1)
  pNovaCelula->aluno = *pNovoAluno;                        O(1)
  pNovaCelula->pProximaCelula = pTurma->pPrimeiraCelula;   O(1)
  pTurma->pPrimeiraCelula = pNovaCelula;                   O(1)

  return 0;

}
// INCLUIR ALUNO
    timeINS = clock();                                     O(1)
    for (i=1;i<=NUM_MAX_ALUNOS;i++)                                  O(n)
        {
          scanf("%d",&(meuAluno.matricula));
          vetor[i] = meuAluno.matricula;                   O(1)
          scanf("%d",&(meuAluno.idade));
          scanf("%s",(meuAluno.nome));
          resultado = inserir(&minhaTurma,&meuAluno);      O(1)
        }

    timeINS = clock() - timeINS;                           O(1)
    fprintf(fInsert, "%6d,%f\n",
     NUM_MAX_ALUNOS, (((float)timeINS)/(CLOCKS_PER_SEC/1000)));   O(1)
```

No algoritmo de inserção a sua complexidade é O(n).

- Algoritmo de Busca:

```
//------------------------------------------------------------
// RECUPERAR DADOS DE UM ALUNO
//------------------------------------------------------------


Aluno *recuperar(const Turma *pTurma, const int matricula)
{
  Celula *p;

  for (p=pTurma->pPrimeiraCelula; p; p=p->pProximaCelula)
  {
    if (p->aluno.matricula == matricula)
    {
      return (Aluno *) &(p->aluno);
    }
  }
  return (Aluno *) NULL;
}
// BUSCAR ALUNO VIA MATRICULA
    timeBUSC = clock();
    for (i=1;i<=NUM_MAX_ALUNOS;i++) {
      m = rand()%NUM_MAX_ALUNOS;
      pAluno = recuperar(&minhaTurma,m);
      }
     timeBUSC = clock() - timeBUSC;
     fprintf(fBusca, "%6d,%f\n",
      NUM_MAX_ALUNOS, (((float)timeBUSC)/(CLOCKS_PER_SEC/1000)));
```

O(1) O(n) O(1) O(n) O(n²) O(1) O(1)

No algoritmo de busca a complexidade é O(n²).

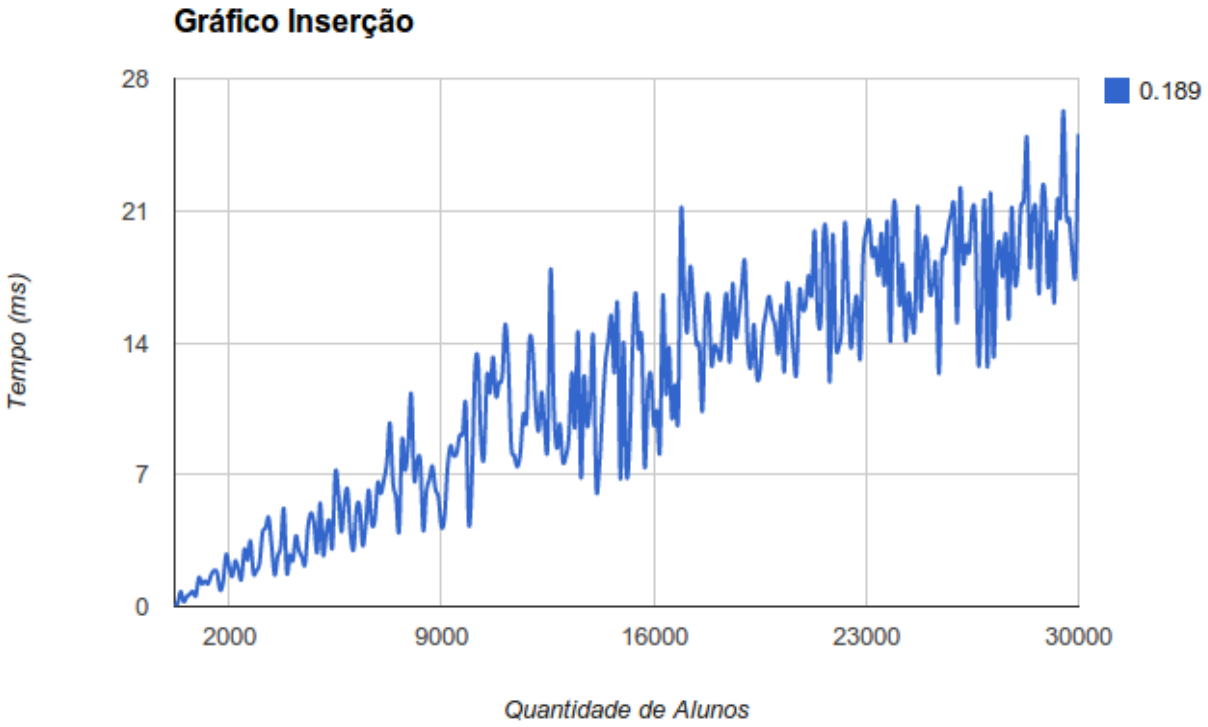
- Algoritmo de Remoção:

```
//------------------------------------------------------------
// REMOVER ALUNO
//------------------------------------------------------------
int excluir(Turma *pTurma, const int matricula)
{
  Celula *p, *pExcluir;
  int r; // ELEMENTO NAO ENCONTRADO
  if (pTurma->pPrimeiraCelula == NULL) return 2; // lista Vazia   O(1)
  if (pTurma->pPrimeiraCelula->aluno.matricula == matricula)
  {
    // Excluir primeiro item da lista
    pExcluir = pTurma->pPrimeiraCelula;
    pTurma->pPrimeiraCelula = pTurma->pPrimeiraCelula->pProximaCelula;   O(1)
    free(pExcluir);
    r=0;
  }                                                                      O(1)
  else
  {
    // Excluir item que nao e o primeiro da lista

    for (p=pTurma->pPrimeiraCelula; p!=NULL; p=p->pProximaCelula)
    {
      if (p->pProximaCelula->aluno.matricula == matricula)
      {
        pExcluir = p->pProximaCelula;
        p->pProximaCelula = p->pProximaCelula->pProximaCelula;           O(n)
        free(pExcluir);
        r=0; break;
      }
    }                                                                    O(n)
  }

// EXCLUIR ALUNO
    timeREM = clock();
    for (i=1;i<=NUM_MAX_ALUNOS;i++) {
      m = vetor[i];
      resultado = excluir(&minhaTurma,m);      O(n²)


      }
     timeREM = clock() - timeREM;
     fprintf(fRemove, "%6d,%f\n",
      NUM_MAX_ALUNOS, (((float)timeREM)/(CLOCKS_PER_SEC/1000)));   O(1)
```

Neste algoritmo de remoção pode-se ver que a complexidade é O(n²).

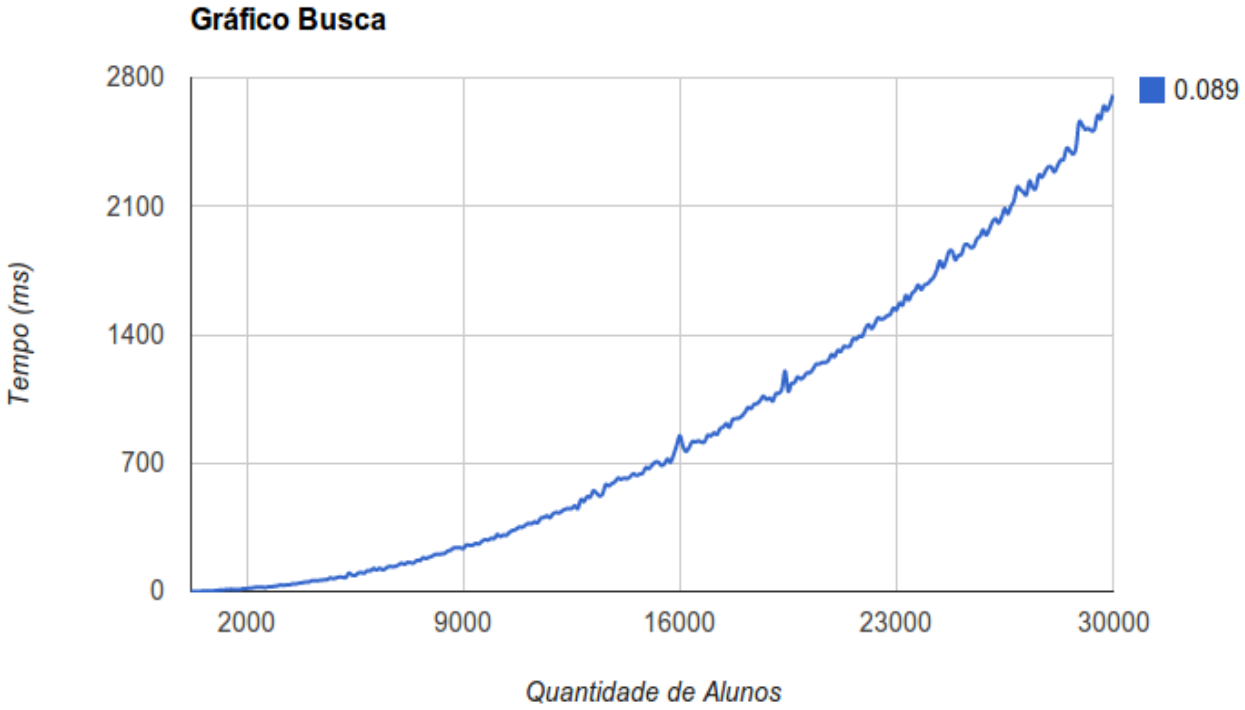
Gráficos Quantidade x Tempo do algoritmo Lista Encadeada:

- Algoritmo de Inserção:



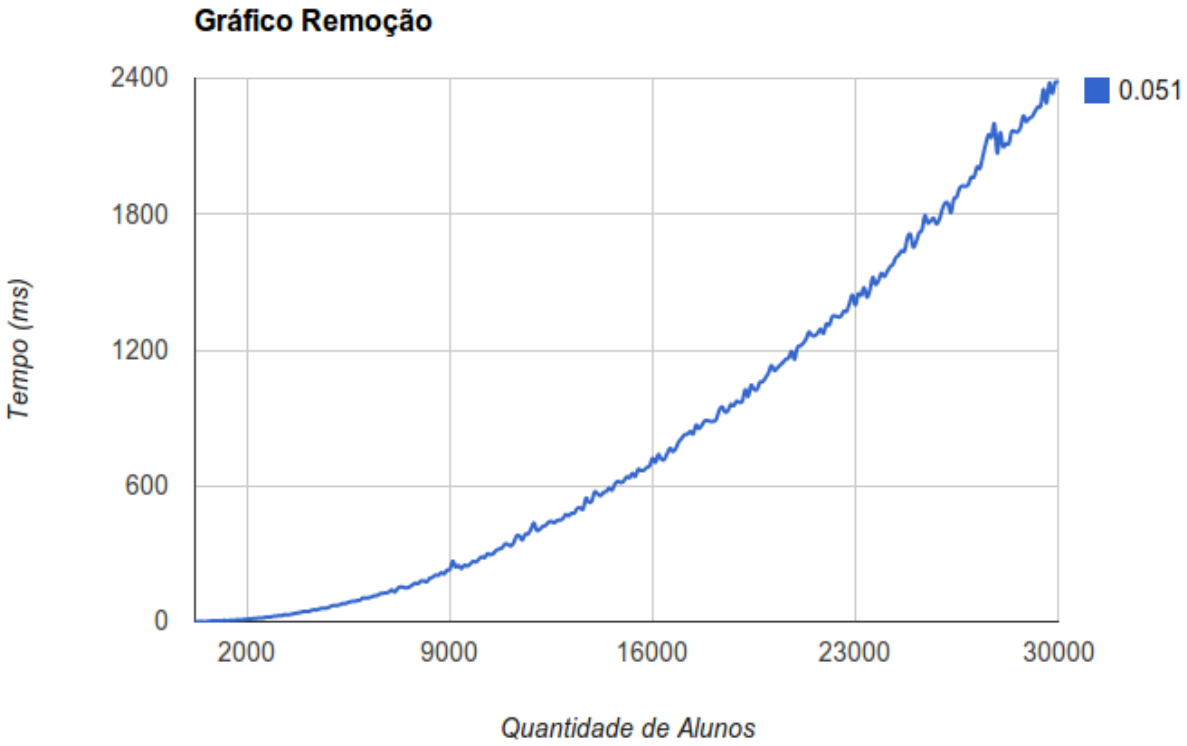
Traçando a média dos pontos, pode-se ver uma reta no gráfico de Inserção.

- Algoritmo de Busca:



O gráfico de busca possui um comportamento quadrático.

- Algoritmo de Remoção:



O gráfico da remoção também possui um comportamento quadrático.