Universidade Federal de Pernambuco
UFPE

Relatório Projeto 3

Professor Renato Vimieiro
IF969 - Algoritmos e Estrutura de Dados
Discente: Emerson Victor Ferreira da Luz

Recife, 2017

Relatório da análise de complexidade

1. Conta somas

Implementação anterior

- Tempo de execução(em segundos) com:
  - 50 elementos → 0,029133271891624
  - 100 elementos → 0,232199794147163
  - 250 elementos → 3,756465839687730
  - 500 elementos → 30,24794884817670
  - 1000 elementos →251,6228153402910
  - 1500 elementos → 1816,433610686100
- Complexidade:
  - Tempo: O(N^3)
  - Espaço: O(N)

Implementação atual

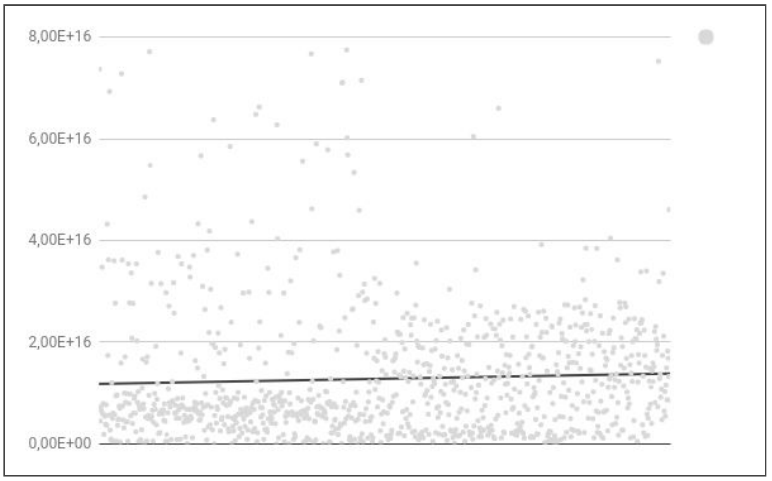
- Tempo de execução com:
  - 50 elementos → 0.004701291996752843
  - 100 elementos → 0.02331280199723551
  - 250 elementos → 0.1355139709994546
  - 500 elementos → 0.6529539900002419
  - 1000 elementos → 2.911640048994741
  - 1500 elementos → 7.018828832995496
- Complexidade:
  - Tempo: O(N^2 log N)
  - Espaço: O(N)

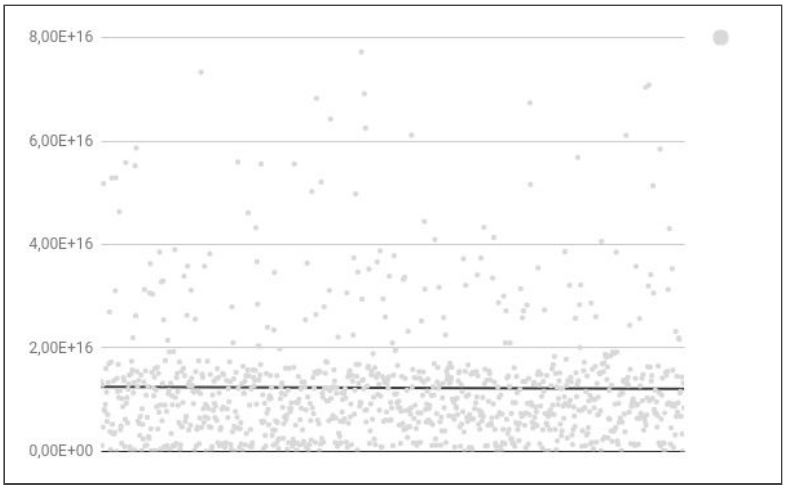
Conclusão

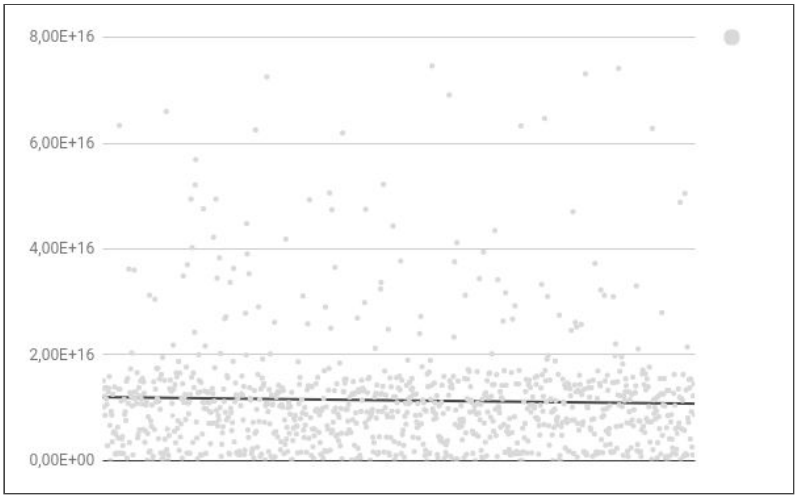
O tempo de execução em vetores grande tornou possível observar o complexidade que o novo algoritmo traz, reduzindo de cerca de1816s para 7s em vetores com 1500 elementos.

2. Lista duplamente encadeada

- **Tempo médio de pesquisa (em segundos):**
  - Não ordenado: 8,43E+15
  - Ordenado crescentemente: 9,71E+15
  - Ordenado Decrescentemente: 9,74E+15



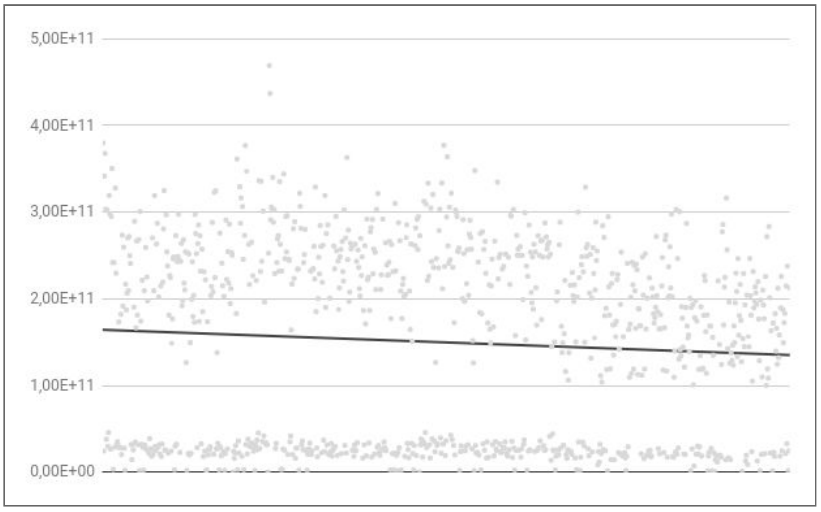


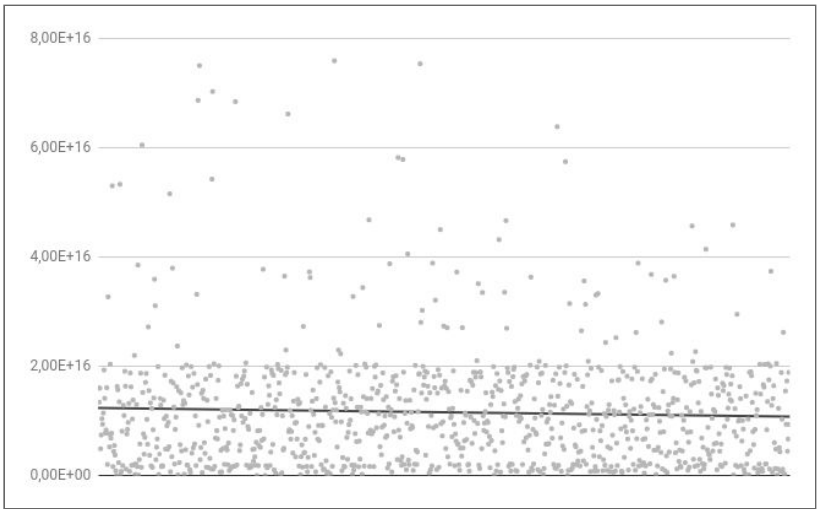


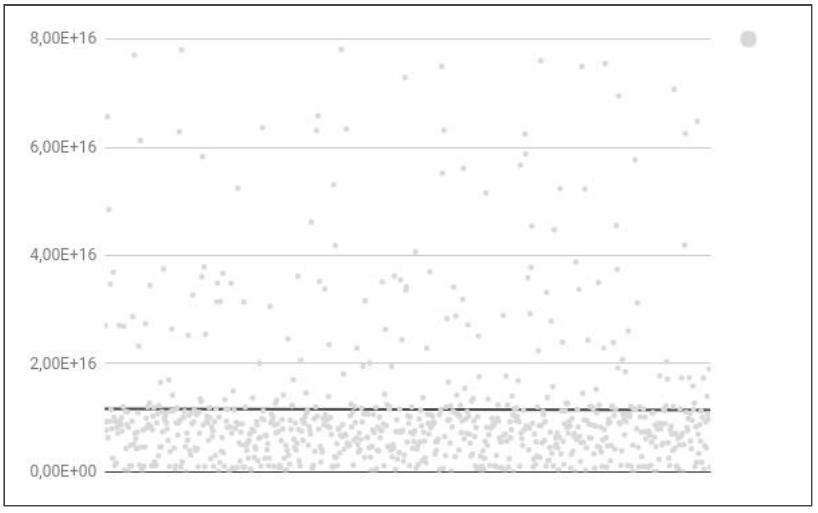
3. Árvore binária

4. Tempo médio de pesquisa (em segundos):

a. Não ordenado: *1,74E+11*
b. Ordenado crescentemente: *1,00E+16*
c. Ordenado Decrescentemente: *8,20E+15*







Conclusões referente a análise da árvore binária e da lista duplamente encadeada

Analisando os gráficos e resultados obtidos nos testes, pode-se perceber um melhor desempenho tanto da lista quanto da árvore com objetos desordenados. Assim como visto teoricamente, a árvore binária possui um melhor desempenho podendo atingir cerca de menos 75% do tempo de execução da lista.

5. Quicksort

QS1 → Refere-se ao método de quebra e concatenação de lista
QS2 → Refere-se ao método de partição

- Teste realizado com 20.000 veículos
    - QS1 = Duração com elementos não ordenados: 0.48259147299904726
    - QS2 = Duração com elementos não ordenados: 0.30996449400117854

    - QS1 = Duração com elementos ordenados crescentemente: -1
    - QS2 = Duração com elementos ordenados crescentemente: -1

    - QS1 = Duração com elementos ordenados decrescentemente: -1
    - QS2 = Duração com elementos ordenados decrescentemente: -1

- Teste realizado com 15.000 veículos
    - QS1 = Duração com elementos não ordenados: 0.38754480000352487
    - QS2 = Duração com elementos não ordenados: 0.24144237299697124

    - QS1 = Duração com elementos ordenados crescentemente: -1
    - QS2 = Duração com elementos ordenados crescentemente: -1

    - QS1 = Duração com elementos ordenados decrescentemente: -1
    - QS2 = Duração com elementos ordenados decrescentemente: -1

- Teste realizado com 10.000 veículos
    - QS1 = Duração com elementos não ordenados: 0.3886835940029414
    - QS2 = Duração com elementos não ordenados: 0.24242694299755385

    - QS1 = Duração com elementos ordenados crescentemente: -1
    - QS2 = Duração com elementos ordenados crescentemente: -1

    - QS1 = Duração com elementos ordenados decrescentemente: -1
    - QS2 = Duração com elementos ordenados decrescentemente: -1

Conclusões:

Apesar do python não realizar tamanha quantidade de recursão em caso da lista está ordenada(crescente ou decrescentemente), pode-se ver uma melhora significativa no desempenho do algoritmo quicksort nos dois métodos apresentados. Os testes foram realizados dez vezes e os resultados apresentados apresentam a média dos valores totais. A diferença em grandes arquivos torna mais perceptível o desempenho do algoritmo. Outro ponto importante a ser analisado é a quantidade de memória adicional que o QS1 utiliza, tornando-se uma escolha inviável entre os algoritmos apresentados.