Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital
Programa de Pós-Graduação em Engenharia de Software - PPGSW

Relatório de Análise do comportamento dos algoritmos *BubbleSort*, *InsertionSort*, *SelecionSort* e *ShellSort* variando o tamanho (n) da entrada de dados

Aluno:
Allyson Bruno Campos Barros Vilela

Natal/RN, Abril de 2017

Implementação dos Algoritmos

Os algoritmos foram implementados utilizando a linguagem *Javascript* e se encontram disponíveis no Github através da URL: https://github.com/allysonbarros/MPES0021_trabalho_2

BubbleSort

```
function(lista) {
    lista_ordenada = Array.from(lista);
    var trocado;
    do {
        trocado = false;
        for (var i = 0; i < lista_ordenada.length - 1; i++) {
            if (lista_ordenada[i] > lista_ordenada[i+1]) {
                var aux = lista_ordenada[i];
                lista_ordenada[i] = lista_ordenada[i+1];
                lista_ordenada[i+1] = aux;
                trocado = true;
            }
        }
    } while (trocado);

    return lista_ordenada;
}
```

Complexidade: $O(n^2)$

InsertionSort

```
function(lista) {
    lista_ordenada = Array.from(lista);
    lista_ordenada.forEach(function(valor, i) {
        while ((i > 0) && (lista_ordenada[i-1] > valor)) {
            lista_ordenada[i] = lista_ordenada[i-1];
            i -= 1;
        }

        lista_ordenada[i] = valor;
    });

    return lista_ordenada;
}
```

Complexidade: $O(n^2)$

SelectionSort

```
function(lista) {
    lista_ordenada = Array.from(lista);
    var menor_valor;
    var menor_indice;

    for (var i = 0; i < lista_ordenada.length; i++) {
        menor_valor = lista_ordenada[i];
        menor_indice = i;

        for (var j = i+1; j < lista_ordenada.length; j++) {
            if (lista_ordenada[j] < menor_valor) {
                menor_valor = lista_ordenada[j]
                menor_indice = j;
            }
        }

        lista_ordenada[menor_indice] = lista_ordenada[i];
        lista_ordenada[i] = menor_valor;
    }

    return lista_ordenada;
}
```

Complexidade: $O(n^2)$

ShellSort

```
function(lista) {
    lista_ordenada = Array.from(lista);
    for (var h = lista_ordenada.length; h > 0; h = parseInt(h / Math.floor(Math.random() * 10) + 1)) {
        for (var i = h; i < lista_ordenada.length; i++) {
            var k = lista_ordenada[i];

            for (var j = i; j >= h && k < lista_ordenada[j - h]; j -= h) {
                lista_ordenada[j] = lista_ordenada[j - h];
            }

            lista_ordenada[j] = k;
        }
    }

    return lista_ordenada;
}
```

Complexidade: $O(nlog_2 n)$

Comparativo entre os Algoritmos

Os testes foram executados utilizando o framework **node.js**, que permite a execução de códigos *javascripts* a partir do terminal do sistema operacional. Como entrada de dados, foram utilizadas quatro listas de tamanhos diferentes com números ordenados de forma aleatória:

- a primeira com **64 elementos** (*https://github.com/allysonbarros/MPES0021_trabalho_2/blob/master/variaveis.js#L1*);
- a segunda com **5.000 elementos** (*https://github.com/allysonbarros/MPES0021_trabalho_2/blob/master/variaveis.js#L3*);
- a terceira com **10.000 elementos** (*https://github.com/allysonbarros/MPES0021_trabalho_2/blob/master/variaveis.js#L5*);
- a última com **100.000 elementos** (*https://github.com/allysonbarros/MPES0021_trabalho_2/blob/master/variaveis.js#L7*).

Código Fonte

```
// Importando os algoritmos de ordenação
const bubble_sort = require('./bubble_sort');
const insertion_sort = require('./insertion_sort');
const selection_sort = require('./selection_sort');
const shell_sort = require('./shell_sort');

// Importando e declarando as listas de valores utilizados pelos testes.
const variaveis = require('./variaveis');
var listas = [variaveis.lista_1, variaveis.lista_2, variaveis.lista_3, variaveis.lista_4];

// Executando a comparação entre os algoritmos;
listas.forEach(function(lista, index) {
    console.log('\n');
    console.log('--- Iniciando os testes com a ' + parseInt(index+1) + 'ª lista que contém
' + lista.length + ' elementos.');

    console.time('bubble_sort');
    var resultado = bubble_sort.bubble_sort(lista);
    console.timeEnd('bubble_sort');
```

```
    console.time('insertion_sort');
    var resultado = insertion_sort.insertion_sort(lista);
    console.timeEnd('insertion_sort');

    console.time('selection_sort');
    resultado = selection_sort.selection_sort(lista);
    console.timeEnd('selection_sort');

    console.time('shell_sort');
    resultado = shell_sort.shell_sort(lista);
    console.timeEnd('shell_sort');
});
console.log('\n');
```

Resultado da Execução dos Testes

	BubbleSort	InsertionSort	SelectionSort	ShellSort
64 elementos	***0,696ms***	***0,280ms***	*0,611ms*	*0,332ms*
5.000 elementos	***43,982ms***	*7,409ms*	*15,511ms*	***4,258ms***
10.000 elementos	***211,732ms***	*34,217ms*	*54,570ms*	***3,185ms***
100.000 elementos	*23084,472ms*	*2746,073ms*	***5451,472ms***	***45,227ms***

Conforme ilustrado na tabela acima, o algoritmo **shellsort** foi o que executou em menor tempo para uma entrada de dados maior que mil elementos. Já para uma entrada de dados menor que mil elementos, o **Insertionsort** obteve um melhor desempenho. O algoritmo **Bubblesort** teve o pior desempenho nas três primeiras listas. Já o algoritmo **Selectionsort** teve um desempenho ruim nas três primeiras listas e foi o com pior desempenho na última lista.