**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 *HeapSort*, *QuickSort*, *MergeSort* e *RadixSort* variando o tamanho (n) da entrada de dados

**Aluno:**

Allyson Bruno Campos Barros Vilela

**Natal/RN, Maio de 2017**

## Implementação dos Algoritmos

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

### HeapSort

```
function trocar_posicao(lista, i, j) {
    var aux = lista[i];
    lista[i] = lista[j];
    lista[j] = aux;
}

function regenerar_arvore(lista, i, tamanho) {
    while (true) {
        var esquerda = i*2 + 1;
        var direita = i*2 + 2;
        var maior_valor = i;

        if (esquerda < tamanho && lista[esquerda] > lista[maior_valor]) {
            maior_valor = esquerda;
        }

        if (direita < tamanho && lista[direita] > lista[maior_valor]) {
            maior_valor = direita;
        }

        if (i == maior_valor) {
            break;
        }

        trocar_posicao(lista, i, maior_valor);
        i = maior_valor;
    }
}

function ordenar(lista, tamanho) {
    for (var i = Math.floor(tamanho/2); i >= 0; i--) {
        regenerar_arvore(lista, i, tamanho);
    }
}

exports.heap_sort = function(lista) {
    lista_ordenada = Array.from(lista);
    ordenar(lista_ordenada, lista_ordenada.length);

    for (var i = lista_ordenada.length - 1; i > 0; i--) {
        trocar_posicao(lista_ordenada, i, 0);
        regenerar_arvore(lista_ordenada, 0, i-1);
```

```
    }

    return lista_ordenada;
}
```

**Complexidade:** $n \log n$

## QuickSort

```
function trocar_posicao(lista, i, j) {
    var aux = lista[i];
    lista[i] = lista[j];
    lista[j] = aux;
}

function particionar_lista(lista, salto, esquerda, direita) {
    var index = esquerda;
    var valor = lista[salto];

    trocar_posicao(lista, salto, direita);

    for(var v = esquerda; v < direita; v++) {
      if(lista[v] < valor) {
        trocar_posicao(lista, v, index);
        index++;
      }
    }

    trocar_posicao(lista, direita, index);
    return index;
}

exports.quick_sort = function(lista, esquerda=0, direita=0) {
    var lista_ordenada = Array.from(lista);
    var salto = null;

    if (esquerda < direita) {
        salto = esquerda + Math.ceil((direita - esquerda) * 0.5)
        novo_salto = particionar_lista(lista_ordenada, salto, esquerda, direita);

        sort(lista_ordenada, esquerda, novo_salto - 1);
        sort(lista_ordenada, novo_salto + 1, direita);
    }

    return lista_ordenada;
}
```

**Complexidade:** $n^2$

# MergeSort

```
function unir_listas(esquerda, direita) {
    var resultado = [];

    while(esquerda.length || direita.length) {
      if(esquerda.length && direita.length) {
        if(esquerda[0] < direita[0]) {
          resultado.push(esquerda.shift());
        } else {
          resultado.push(direita.shift());
        }
      } else if (esquerda.length) {
        resultado.push(esquerda.shift());
      } else {
        resultado.push(direita.shift());
      }
    }

    return resultado;
}

function merge_sort(lista) {
    lista_ordenada = Array.from(lista);

    var tamanho = lista_ordenada.length;
    var meio = Math.floor(tamanho * 0.5);
    var esquerda = lista_ordenada.slice(0, meio);
    var direita = lista_ordenada.slice(meio, tamanho);

    if(tamanho === 1) {
      return lista_ordenada;
    }

    return unir_listas(merge_sort(esquerda), merge_sort(direita));
}

exports.merge_sort = merge_sort;
```

**Complexidade:** $n \log n$

## RadixSort

```
function get_digito(numero, index) {
  var digito = 0;
  while(index--){
    digito = numero % 10
    numero = Math.floor((numero - digito) / 10)
  }
  return digito
}

exports.radix_sort = function(lista) {
    var lista_ordenada = Array.from(lista);
    console.log(lista_ordenada);
    var maximo = Math.floor(Math.log10(Math.max.apply(Math, lista_ordenada)));
    var array_digitos = [];
    var idx = 0;

    for(var i = 0;i < maximo + 1; i++) {
        array_digitos = []

        for(var j = 0; j < lista_ordenada.length; j++) {
            var digito = get_digito(lista_ordenada[j], i+1);
            array_digitos[digito] = array_digitos[digito] || [];
            array_digitos[digito].push(lista_ordenada[j]);
        }

        idx = 0
        for(var t = 0; t < array_digitos.length; t++){
            if(array_digitos[t] && array_digitos[t].length > 0) {
                for(j = 0; j < array_digitos[t].length; j++) {
                    lista_ordenada[idx++] = array_digitos[t][j];
                }
            }
        }
    }

    return lista_ordenada;
}
```

**Complexidade:** $n(k)$

## 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 mais eficientes.
const heap_sort = require('./heap_sort');
const quick_sort = require('./quick_sort');
const merge_sort = require('./merge_sort');
const radix_sort = require('./radix_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('heap_sort');
    resultado = heap_sort.heap_sort(lista);
    console.timeEnd('heap_sort');

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

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

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

## Resultado da Execução dos Testes

### Algoritmos Eficientes

| | HeapSort | QuickSort | MergeSort | RadixSort |
|---|---|---|---|---|
| **64 elementos** | *0,882ms* | ***0,203ms*** | ***1,378ms*** | *0,632ms* |
| **5.000 elementos** | *2,125ms* | ***0,996ms*** | ***45,293ms*** | *4,551ms* |
| **10.000 elementos** | *3,091ms* | ***1,517ms*** | ***80,920ms*** | *4,567ms* |
| **100.000 elementos** | *36,468ms* | ***26,734ms*** | ***885,915ms*** | *36,564ms* |

### Algoritmos Ineficientes

| | BubbleSort | InsertionSort | SelectionSort | ShellSort |
|---|---|---|---|---|
| **64 elementos** | ***0,713ms*** | ***0,249ms*** | *0,578ms* | *0,384ms* |
| **5.000 elementos** | ***45,231ms*** | *8,034ms* | *20,646ms* | ***2,379ms*** |
| **10.000 elementos** | ***204,810ms*** | *29,094ms* | *50,524ms* | ***3,739ms*** |
| **100.000 elementos** | *24547,505ms* | *3474,242ms* | ***6729,415ms*** | ***136,526ms*** |

Conforme ilustrado na tabela acima, o algoritmo **QuickSort** foi o que apresentou o melhor desempenho enquanto que o algoritmo **MergeSort** teve o pior desempenho. O **RadixSort** teve o segundo melhor desempenho apenas na primeira lista enquanto que o **HeapSort** teve o segundo melhor desempenho nas outras três listas. Comparando os resultados da execução dos algoritmos ineficientes com os mais eficientes, é possível ver a grande diferença de tempo de execução entre os dois algoritmos com mais desempenho.