Aluno: João Vitor Bessa Lacerda

PRIMEIRA PARTE

ESPECIFICAÇÕES DO SISTEMA

- Modelo do processador e velocidade: Intel i7-7700 3.6GHz
- Quantidade de memória principal / RAM: 16GB
- Sistema operacional utilizado: Windows 10 Pro

➔CÓDIGO EM JAVA

```java
import java.util.Random;

public class App {

    private static void quicksort(int[] array, int esq, int dir) {

        int part;
        if (esq < dir) {
            part = particao(array, esq, dir);
            quicksort(array, esq, part - 1);
            quicksort(array, part + 1, dir);
        }
    }

    private static int particao(int[] array, int inicio, int fim) {

        int pivot = array[fim];
        int part = inicio - 1;
        for (int i = inicio; i < fim; i++) {
            if (array[i] < pivot) {
                part++;
                swap(array, part, i);
            }
        }
        part++;
        swap(array, part, fim);
        return (part);
    }

    private static void swap(int[] array, int i, int j) {
```

```
        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    ////////////////////////////////////////////////////////////////

    private static void bubblesort(int[] array) {
        for (int i = array.length - 1; i > 0; i--) {

            for (int w = 0; w < i; w++) {

                if (array[w] > array[w + 1]) {
                    int aux = array[w];
                    array[w] = array[w + 1];
                    array[w + 1] = aux;
                }
            }
        }
    }

    ////////////////////////////////////////////////////////////////

    private static void randomFillArray(int[] array, int seed) {

        Random randomComSeed = new Random(seed);

        for (int i = 0; i < array.length; i++) {
            array[i] = randomComSeed.nextInt(10);
        }
    }

    ////////////////////////////////////////////////////////////////

    public static void main(String[] args) throws Exception {

        int[] vetor625 = new int[62500];

        long[] temposQuick = new long[50];
        long[] temposBubble = new long[50];

        long inicio = 0;
        long fim = 0;
```

```
        for (int i = 0; i < 50; i++) {

            randomFillArray(vetor625, i);
            inicio = System.currentTimeMillis();
            quicksort(vetor625, 0, vetor625.length - 1);
            fim = System.currentTimeMillis();
            temposQuick[i] = fim - inicio;

            inicio = System.currentTimeMillis();
            bubblesort(vetor625);
            fim = System.currentTimeMillis();
            temposBubble[i] = fim - inicio;
        }

        double media2 = 0.0;
        double media1 = 0.0;

        for (int i = 0; i < temposQuick.length; i++) {
            media1 += temposQuick[i];
            media2 += temposBubble[i];

        }
        System.out.println("Média Quicksort [62500]: " + media1 / 50);
        System.out.println("Média Bubblesort [62500]: " + media2 / 50);

        ////////////////////////////////////////////////////////////////
        System.out.println("-=-=-=-=-=-=-=-=-=-=-");
        ////////////////////////////////////////////////////////////////

        int[] vetor125 = new int[125000];

        for (int i = 0; i < 50; i++) {

            randomFillArray(vetor125, i);
            inicio = System.currentTimeMillis();
            quicksort(vetor125, 0, vetor125.length - 1);
            fim = System.currentTimeMillis();
            temposQuick[i] = fim - inicio;

            inicio = System.currentTimeMillis();
            bubblesort(vetor125);
            fim = System.currentTimeMillis();
```

```
            temposBubble[i] = fim - inicio;
        }

        media2 = 0.0;
        media1 = 0.0;

        for (int i = 0; i < temposQuick.length; i++) {
            media1 += temposQuick[i];
            media2 += temposBubble[i];

        }
        System.out.println("Média Quicksort [125000]: " + media1 / 50);
        System.out.println("Média Bubblesort [125000]: " + media2 /
50);

        ////////////////////////////////////////////////////////////////
        System.out.println("-=-=-=-=-=-=-=-=-=-");
        ////////////////////////////////////////////////////////////////

        int[] vetor250 = new int[250000];

        for (int i = 0; i < 50; i++) {

        randomFillArray(vetor250, i);
        inicio = System.currentTimeMillis();
        quicksort(vetor250, 0, vetor250.length - 1);
        fim = System.currentTimeMillis();
        temposQuick[i] = fim - inicio;

        inicio = System.currentTimeMillis();
        bubblesort(vetor250);
        fim = System.currentTimeMillis();
        temposBubble[i] = fim - inicio;
        }

        media2 = 0.0;
        media1 = 0.0;

        for (int i = 0; i < temposQuick.length; i++) {
        media1 += temposQuick[i];
        media2 += temposBubble[i];

        }
```

```
        System.out.println("Média Quicksort [250000]: " + media1 / 50);
        System.out.println("Média Bubblesort [250000]: " + media2 /
50);
    }
}
```

RESULTADOS

JAVA (em ms)	QUICKSORT	BUBBLESORT
62.500	89.3	375.58
125.000	370.32	1459.22
250.000	Erro	Erro
375.000	Erro	Erro

*Erro: StackOverflowError

CONCLUSÃO:

Após a extração dos resultados, é possível notar a disparidade do tempo gasto para a execução dos *sortings* em cada algoritmo. Essa diferença faz jus aos respectivos cálculos de complexidade [O(n logn) e O(n^2)].

Devido a uma limitação padrão de chamadas recursivas das IDE's utilizadas nos testes (VSCode e IntelliJ), a partir do teste "250.000" houve uma inviabilidade de retornar resultados: o erro "StackOverflowError" foi lançado. Este erro é uma exceção em tempo de execução que ocorre quando a pilha de chamadas de um programa Java atinge um tamanho muito grande, ultrapassando a capacidade máxima permitida.

Por isso, realizei o mesmo teste, trocando apenas o algoritmo bubblesort pelo selection sort e desta vez na linguagem C, que por sua vez não limitou o tamanho da pilha de chamadas, sem retornar erro.

➔CÓDIGO EM C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define ARR_SIZE_62   62500
#define ARR_SIZE_125 125000
```

```c
#define ARR_SIZE_250 250000
#define ARR_SIZE_375 375000

void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

int partition(int* arr, int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr + i, arr + j);
        }
    }
    swap(arr + (i + 1), arr + high);
    return (i + 1);
}

void quickSort(int* arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

void selectionSort(int* arr, int n) {
    int i, j, min_idx;

    for (i = 0; i < n-1; i++) {
        min_idx = i;
        for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        swap(arr + min_idx, arr + i);
    }
```

```
}

void randomFillArray(int* arr, int seed, int arr_size){
    srand(seed);
    for(int i = 0 ; i < arr_size; i++){
        arr[i] = rand() % 100;
    }
}

double media(double* arr, int arr_size){
    double media = 0;
    if(arr != NULL && arr_size >= 0){
        for(int i = 0; i < arr_size; i++){
            media += arr[i];
        }
    }
    else{
        printf("ERRO");
        media = -1;
    }
    return media/arr_size;
}

int main(){

    clock_t inicio, fim;
    double tempo;

    double* tempos_quick_62 = malloc(sizeof(double)*50);

    double* tempos_quick_125 = malloc(sizeof(double)*50);

    double* tempos_quick_250 = malloc(sizeof(double)*50);

    double* tempos_quick_375 = malloc(sizeof(double)*50);

    double* tempos_selec_62 = malloc(sizeof(double)*50);

    double* tempos_selec_125 = malloc(sizeof(double)*50);

    double* tempos_selec_250 = malloc(sizeof(double)*50);

    double* tempos_selec_375 = malloc(sizeof(double)*50);
```

```
    //Bloco 62.500

    int* arrQuick = malloc(sizeof(int)*ARR_SIZE_62);

    int* arrSelec = malloc(sizeof(int)*ARR_SIZE_62);

    for(int i = 0; i < 50; i++){

        randomFillArray(arrQuick,i,ARR_SIZE_62);

        inicio = clock();
        quickSort(arrQuick, 0, ARR_SIZE_62);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
        tempos_quick_62[i] = tempo;

    }

    printf("Media de tempo para QuickSort em array de tamanho 62.500:
%lf segundos\n",media(tempos_quick_62,50));

    printf("Seleção 62.500: 0 porcento concluido\n");

    for(int i = 0; i < 50; i++){

        randomFillArray(arrSelec,i,ARR_SIZE_62);

        inicio = clock();
        selectionSort(arrSelec,ARR_SIZE_62);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
        tempos_selec_62[i] = tempo;

        printf("Seleção 62.500: %d porcento concluido\n",(i+1)*2);
        fflush(stdout);

    }
```

```
    printf("Media de tempo para SelectionSort em array de tamanho
62.500: %lf segundos\n",media(tempos_selec_62,50));

    free(arrQuick);
    arrQuick = NULL;

    free(arrSelec);
    arrSelec = NULL;

    //Bloco 125.000


    arrQuick = malloc(sizeof(int)*ARR_SIZE_125);

    arrSelec = malloc(sizeof(int)*ARR_SIZE_125);

    for(int i = 0; i < 50; i++){

        randomFillArray(arrQuick,i,ARR_SIZE_125);

        inicio = clock();
        quickSort(arrQuick, 0, ARR_SIZE_125);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
        tempos_quick_125[i] = tempo;

    }

    printf("Media de tempo para QuickSort em array de tamanho 125.000:
%lf segundos\n",media(tempos_quick_125,50));

    printf("Seleção 125.000: 0 porcento concluido\n");

    for(int i = 0; i < 50; i++){

        randomFillArray(arrSelec,i,ARR_SIZE_125);

        inicio = clock();
        selectionSort(arrSelec,ARR_SIZE_125);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
```

```
        tempos_selec_125[i] = tempo;

        printf("Seleção 125.000: %d porcento concluido\n",(i+1)*2);
        fflush(stdout);

    }

    printf("Media de tempo para SelectionSort em array de tamanho
125.000: %lf segundos\n",media(tempos_selec_125,50));

    free(arrQuick);
    arrQuick = NULL;

    free(arrSelec);
    arrSelec = NULL;


    //Bloco 250.000

    arrQuick = malloc(sizeof(int)*ARR_SIZE_250);

    arrSelec = malloc(sizeof(int)*ARR_SIZE_250);

    for(int i = 0; i < 50; i++){

        randomFillArray(arrQuick,i,ARR_SIZE_250);

        inicio = clock();
        quickSort(arrQuick, 0, ARR_SIZE_250);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
        tempos_quick_250[i] = tempo;

    }

    printf("Media de tempo para QuickSort em array de tamanho 250.00:
%lf segundos\n",media(tempos_quick_250,50));

    printf("Seleção 250.000: 0 porcento concluido\n");

    for(int i = 0; i < 50; i++){
```

```
        randomFillArray(arrSelec,i,ARR_SIZE_250);

        inicio = clock();
        selectionSort(arrSelec,ARR_SIZE_250);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
        tempos_selec_250[i] = tempo;

        printf("Seleção 250.000: %d porcento concluido\n",(i+1)*2);
        fflush(stdout);

    }

    printf("Media de tempo para SelectionSort em array de tamanho
250.000: %lf segundos\n",media(tempos_selec_250,50));

    free(arrQuick);
    arrQuick = NULL;

    free(arrSelec);
    arrSelec = NULL;


    //Bloco 375.000

    arrQuick = malloc(sizeof(int)*ARR_SIZE_375);

    arrSelec = malloc(sizeof(int)*ARR_SIZE_375);

    for(int i = 0; i < 50; i++){

        randomFillArray(arrQuick,i,ARR_SIZE_375);

        inicio = clock();
        quickSort(arrQuick, 0, ARR_SIZE_375);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
        tempos_quick_375[i] = tempo;

    }
```

```
    printf("Media de tempo para QuickSort em array de tamanho 375.000:
%lf segundos\n",media(tempos_quick_375,50));

    printf("Seleção 375.000: 0 porcento concluido\n");

    for(int i = 0; i < 50; i++){

        randomFillArray(arrSelec,i,ARR_SIZE_375);

        inicio = clock();
        selectionSort(arrSelec,ARR_SIZE_375);
        fim = clock();

        tempo = ((double)(fim-inicio)) / CLOCKS_PER_SEC;
        tempos_selec_375[i] = tempo;

        printf("Seleção 375.000: %d porcento concluido\n",(i+1)*2);
        fflush(stdout);

    }

    printf("Media de tempo para SelectionSort em array de tamanho
375.000: %lf segundos\n",media(tempos_selec_375,50));

    free(arrQuick);
    arrQuick = NULL;

    free(arrSelec);
    arrSelec = NULL;

    return 0;
}
```

RESULTADOS

C (em s)	QUICKSORT	SELECTION SORT
62.500	0.065353	5.840493
125.000	0.248581	23.35706
250.000	0.973069	72.821701

375.000	2.130369	187.770264

ANÁLISE:

Analisando a porcentagem de crescimento de tempo de execução (divisão do valor de tempo do próximo tamanho de vetor pelo anterior, por exemplo: $23.35706 / 5.840493 \cong 3,99$), podemos observar que a taxa de crescimento é menor no algoritmo quicksort comparado ao algoritmo selection sort.

CONCLUSÃO:
Agora, foi possível concluir com sucesso o teste. E assim podemos afirmar mais uma vez que a discrepância corresponde com a teoria de complexidade [O(n logn) e O(n^2)]. A taxa de aumento de tempo de execução para o quicksort é bem menor que a do selection sort.

SEGUNDA PARTE

→CÓDIGO JAVA

```java
import java.util.Random;

public class App {

    private static void quicksort(int[] array, int esq, int dir) {

        int part;
        if (esq < dir){
            part = particao(array, esq, dir);
            quicksort(array, esq, part - 1);
            quicksort(array, part + 1, dir);
        }
    }

    private static int particao(int[] array, int inicio, int fim) {

        int pivot = array[fim];
        int part = inicio - 1;
        for (int i = inicio; i < fim; i++) {
            if (array[i] < pivot) {
                part++;
                swap(array, part, i);
            }
        }
```

```
        part++;
        swap(array, part, fim);
        return (part);
    }

    private static void swap(int[] array, int i, int j) {

        int temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }


    private static void randomFillArray(int[] array){

        Random randomComSeed = new Random(1);

        for (int i = 0; i < array.length; i++){
            array[i] = randomComSeed.nextInt(10);
        }
    }

    private static void fillArray(int[] array){

        for (int i = 0; i < array.length; i++){
            array[i] = i+1;
        }
    }
    public static void main(String[] args) throws Exception {

        int[] vetor = new int[10000];
        int[] vetor2 = new int[10000];

        long[] tempos = new long[10];
        long[] temposOrdenados = new long[10];


        long inicio = 0;
        long fim = 0;

        for (int i = 0; i < 10; i++){
            randomFillArray(vetor);
            inicio = System.currentTimeMillis();
```

```
            quicksort(vetor, 0, vetor.length - 1);
            fim = System.currentTimeMillis();
            tempos[i] = fim - inicio;
        }

        for (int i = 0; i < 10; i++){
            fillArray(vetor2);
            inicio = System.currentTimeMillis();
            quicksort(vetor2, 0, vetor2.length - 1);
            fim = System.currentTimeMillis();
            temposOrdenados[i] = fim - inicio;
        }

        double media1 = 0.0;

        for(int i=0; i < tempos.length; i++){
            media1 += tempos[i];
        }

        System.out.println("Média dos aleatórios: " + media1/10);

        double media2 = 0.0;

        for(int i=0; i < temposOrdenados.length; i++){
            media2 += temposOrdenados[i];
        }

        System.out.println("Média dos ordenados: " + media2/10);
    }
}
```

RESULTADOS

(TESTE EM ms)	QUICKSORT
ALEATÓRIO	2.7
ORDENADO	72.3

ANÁLISE

É possível concluir com os testes registrados na tabela, a grande contraditória eficiência do quicksort quando executado para organizar um vetor aleatório e quando para um vetor já ordenado.
Isso ocorre porque quando o vetor já está ordenado, o pivô escolhido no início do algoritmo sempre será o menor ou o maior elemento, dependendo se a ordem é ascendente ou descendente. Isso significa que, em cada iteração, o pivô dividirá o vetor em uma parte com apenas um elemento e outra com todos os outros elementos. Portanto, o Quicksort terá que percorrer todo o vetor várias vezes, tornando o desempenho muito pior do que em um cenário aleatório.

CONCLUSÃO

O Algoritmo Quicksort tem tempo de execução menor quando ordena um vetor aleatório (pior caso O $= n^2$) do que quando ordena um vetor já ordenado(melhor caso O = $(n \log n)$).