UNIP
UNIVERSIDADE PAULISTA

4º Semestre Ciência da Computação (CC)

Atividade Prática Supervisionada

Tema: "DESENVOLVIMENTO DE SISTEMA PARA ANÁLISE DE PERFORMANCE DE ALGORITMOS DE ORDENAÇÃO DE DADOS."

Alunos participantes:

Filipi Yukio Iwakami Itoyama, RA – N4453J-1

William Rossi do Carmo Ruiz, RA - N473GF-8

Luiz Gabriel Zeferino Duarte, RA – N454CD-8

Índice

1. Objetivo do Trabalho

Têm-se como objetivo deste trabalho, gerar um vetor de números aleatórios e ordená-los utilizando-se de 3 métodos de ordenação e computando o tempo em que cada método gasta no processamento. Cada método tem que ordenar o vetor aleatório gerado de forma crescente a partir do vetor aleatório gerado, ordenar o vetor em forma decrescente e utilizar esse vetor decrescente para ordenar em forma crescente, e por fim, ordenar o vetor que já está crescente para a forma crescente novamente. Isso é para computar o tempo em uma hipótese comum (aleatório para crescente), na pior das hipóteses (decrescente para crescente) e na melhor das hipóteses (crescente para crescente), respectivamente.

2. Geração de números aleatórios

Primeiramente, para podermos começar o nosso trabalho de análise, precisamos de um vetor linear, com 100.000 elementos, mas gerados de forma totalmente aleatória, ou seja, para isso, precisaremos criar um pequeno e simples algoritmo para essa finalidade.

Abaixo, encontra-se o algoritmo para a geração de números aleatórios, em linguagem JAVA:

```
import java.util.Arrays;

import java.util.Random;

Random geraNum = new Random();

int[] arrayAle = new int[100000];

for (int i = 0; i < arrayAle.length; i++) {

    arrayAle[i] = geraNum.nextInt(200000);

}

System.out.println(Arrays.toString(arrayAle));
```

3. Bubble Sort

O primeiro estilo de ordenação que iremos apresentar é o Bubble Sort, nela a ordenação acontece de forma em que o vetor é percorrido por diversas vezes, e cada vez que ela percorre o vetor, ela pega o elemento comparador (o que está perseguindo o vetor inteiro) e compara sempre com o próximo elemento do vetor, passando o elemento maior sempre para o final. Esse caso citado acime é para o ordenamento crescente, para o ordenamento decrescente é só fazer o contrário, para que o elemento MENOR vá para o final. Enquanto isso, o algoritmo de ordenamento somente para metade do vetor, na hora de se declarar o “for” para definir a quantia de vezes que o algoritmo vai atravessar, é só dividir por 2, para obter a metade de quantidade de vezes que vai percorrer.

A seguir apresentamos o código para a ordenação bubble sort para ordenação crescente:

```
private static int[] bubbleSortCrescente(int[] a) {
        int[] arrayNovo = new int[10000];
        for (int i = 0; i < a.length; i++) {
            arrayNovo[i] = a[i];
        }
        for(int i = 0; i < arrayNovo.length; i++) {
            for(int j = 0; j < arrayNovo.length - 1 - i; j++) {
                if(arrayNovo[j] > arrayNovo[j + 1]) {
                    int aux = arrayNovo[j];
                    arrayNovo[j] = arrayNovo[j + 1];
                    arrayNovo[j + 1] = aux;
                }
            }
        }
```

```
        int[] arrayNovoCrescente = new int[10000];
        for (int i = 0; i < arrayNovo.length; i++) {
            arrayNovoCrescente[i] = arrayNovo[i];
        }
        return arrayNovoCrescente;
    }
```

Este basicamente é o método utilizando-se do estilo Bubble Sort para ordenar o vetor de forma crescente, e dentro do método main, acontecerá a chamada do método, junto à computação do tempo gasto:

```
System.out.println(Arrays.toString(arrayAle));
        long inicio = System.currentTimeMillis();
        int[] arrayCrescente = bubbleSortCrescente(arrayAle);
        long termino = System.currentTimeMillis();
        double tempo = (termino - inicio) / 1000.0;
        System.out.println(Arrays.toString(arrayCrescente));
        System.out.println("Tempo de execução (crescente): " + tempo + " segundos");
```

A seguir, apresentaremos o método que ordena o vetor em ordem decrescente:

```
private static int[] bubbleSortDecrescente(int[] a) {
        int[] arrayNovo = new int[10000];
        for (int i = 0; i < a.length; i++) {
            arrayNovo[i] = a[i];
        }
        for(int i = 0; i < arrayNovo.length; i++) {
```

```
            for(int j = 0; j < arrayNovo.length - 1 - i; j++) {
                if(arrayNovo[j] < arrayNovo[j + 1]) {
                    int aux = arrayNovo[j];
                    arrayNovo[j] = arrayNovo[j + 1];
                    arrayNovo[j + 1] = aux;
                }
            }
        }
        int[] arrayNovoDecrescente = new int[10000];
        for (int i = 0; i < arrayNovo.length; i++) {
            arrayNovoDecrescente[i] = arrayNovo[i];
        }
        return arrayNovoDecrescente;
    }
```

E logo após, no método main, a chamada do método, junto à computação do tempo:

```
int[] arrayDecrescente = bubbleSortDecrescente(arrayAle);
        System.out.println(Arrays.toString(arrayDecrescente));
        long inicio2 = System.currentTimeMillis();
        int[] arrayDecrescenteCrescente = bubbleSortCrescente(arrayDecrescente);
        long termino2 = System.currentTimeMillis();
        double tempo2 = (termino2 - inicio2) / 1000.0;
        System.out.println(Arrays.toString(arrayDecrescenteCrescente));
```

System.out.println("Tempo de execução (decrescente - crescente): " + tempo2 + " segundos");

E por fim, para finalizarmos o bubble sort, utilizaremos o vetor já ordenado em forma crescente feito acima, para Reordenarmos novamente chamando o método de ordenação de forma crescente:

System.out.println(Arrays.toString(arrayCrescente));

long inicio1 = System.currentTimeMillis();

int[] arrayCrescenteCrescente = bubbleSortCrescente(arrayCrescente);

long termino1 = System.currentTimeMillis();

double tempo1 = (termino1 - inicio1) / 1000.0;

System.out.println(Arrays.toString(arrayCrescenteCrescente));

System.out.println("Tempo de execução (crescente-crescente): " + tempo1 + " segundos");

O resultado os códigos acima (com 100 mil números aleatórios gerados), é:

```
run:
[1609, 65494, 154054, 23043, 129354, 99643, 131387, 156874, 19499
[0, 2, 4, 6, 6, 10, 15, 16, 20, 30, 32, 33, 35, 37, 38, 39, 39, 4
Tempo de execução (crescente): 15.58 segundos
[199997, 199997, 199997, 199997, 199994, 199994, 199992, 199991,
[0, 2, 4, 6, 6, 10, 15, 16, 20, 30, 32, 33, 35, 37, 38, 39, 39, 4
Tempo de execução (decrescente - crescente): 7.835 segundos
[0, 2, 4, 6, 6, 10, 15, 16, 20, 30, 32, 33, 35, 37, 38, 39, 39, 4
[0, 2, 4, 6, 6, 10, 15, 16, 20, 30, 32, 33, 35, 37, 38, 39, 39, 4
Tempo de execução (crescente-crescente): 1.653 segundos
CONSTRUÍDO COM SUCESSO (tempo total: 43 segundos)
```

4. Selection Sort

O segundo estilo de ordenação que iremos apresentar é o Selection Sort, no qual é um estilo de ordenação um pouco mais eficiente do que o bubble sort apresentado anteriormente. Basicamente dizendo, esse estilo de ordenação vai ordenar os elementos dentro de um vetor, passando por todos os elementos e sempre pegando o elemento menor e passando ele para o início do vetor, ou o contrário, passando o maior elemento para o início, dependendo do requerimento da ordenação

(crescente/decrescente). O algoritmo, sempre vai passar pelo vetor, alocando o primeiro elemento da passagem em uma variável auxiliar, e a partir disso, comparando com os próximos, e sempre que for menor, ou maior, ele aloca o valor nessa variável auxiliar, e fazendo isso pelo vetor inteiro, no final, alocando o valor que está nessa variável no início do vetor, juntamente, trocando o valor que estava no local (primeiro elemento) com o index armazenado por uma outra variável auxiliar que armazenou o index, junto ao valor de tal elemento.

Para melhor compreensão, mostraremos o código a seguir para a ordenação de forma crescente:

```
private static int[] selectionSortCrescente(int[] a){
    int[] arrayNovo = new int[100000];
    for (int i = 0; i < a.length; i++) {
        arrayNovo[i] = a[i];
    }
    for (int i = 0; i < arrayNovo.length - 1; i++){
        int index = i;
        for (int j = i + 1; j < arrayNovo.length; j++){
            if (arrayNovo[j] < arrayNovo[index]){
                index = j;
            }
        }
        int smallerNumber = arrayNovo[index];
        arrayNovo[index] = arrayNovo[i];
        arrayNovo[i] = smallerNumber;
    }
```

```
        int[] arrayNovoCrescente = new int[100000];
        for (int i = 0; i < arrayNovo.length; i++) {
            arrayNovoCrescente[i] = arrayNovo[i];
        }
        return arrayNovoCrescente;
    }
```

Logo após, faremos o mesmo processo que foi feito para a chamada do método e a computação do tempo gasto que foi feito para o bubble sort, só que, trocando somente de bubble sort para o selection sort:

```
System.out.println(Arrays.toString(arrayAle));
        long inicio4 = System.currentTimeMillis();
        int[] arrayCrescente1 = selectionSortCrescente(arrayAle);
        long termino4 = System.currentTimeMillis();
        double tempo4 = (termino4 - inicio4) / 1000.0;
        System.out.println(Arrays.toString(arrayCrescente1));
        System.out.println("Tempo de execução (crescente): " + tempo4 + " segundos");
```

A seguir, o método de selection sort para a ordenação decrescente:

```
private static int[] selectionSortDecrescente(int[] a){
        int[] arrayNovo = new int[100000];
        for (int i = 0; i < a.length; i++) {
            arrayNovo[i] = a[i];
        }
        for (int i = 0; i < arrayNovo.length - 1; i++){
            int index = i;
```

```
            for (int j = i + 1; j < arrayNovo.length; j++){
                if (arrayNovo[j] > arrayNovo[index]){
                    index = j;
                }
            }
            int smallerNumber = arrayNovo[index];
            arrayNovo[index] = arrayNovo[i];
            arrayNovo[i] = smallerNumber;
        }
        int[] arrayNovoDecrescente = new int[100000];
        for (int i = 0; i < arrayNovo.length; i++) {
            arrayNovoDecrescente[i] = arrayNovo[i];
        }
        return arrayNovoDecrescente;
    }
```

E novamente, faremos o mesmo processo para a chamada do método e a computação do tempo:

```
int[] arrayDecrescente1 = selectionSortDecrescente(arrayAle);
        System.out.println(Arrays.toString(arrayDecrescente1));
        long inicio5 = System.currentTimeMillis();
        int[] arrayDecrescenteCrescente1 = selectionSortCrescente(arrayDecrescente1);
        long termino5 = System.currentTimeMillis();
        double tempo5 = (termino5 - inicio5) / 1000.0;
```

System.out.println(Arrays.toString(arrayDecrescenteCrescente1));

System.out.println("Tempo de execução (decrescente - crescente): " + tempo5 + " segundos");

E por fim, utilizando-se do mesmo método de ordenação crescente e do mesmo vetor ordenado de forma crescente criado anteriormente, Reordenaremos crescentemente o vetor já crescente:

System.out.println(Arrays.toString(arrayCrescente1));

long inicio6 = System.currentTimeMillis();

int[] arrayCrescenteCrescente1 = selectionSortCrescente(arrayCrescente1);

long termino6 = System.currentTimeMillis();

double tempo6 = (termino6 - inicio6) / 1000.0;

System.out.println(Arrays.toString(arrayCrescenteCrescente1));

System.out.println("Tempo de execução (crescente-crescente): " + tempo6 + " segundos");

O resultado a respeito do tempo é o seguinte:

```
run:
[64403, 156647, 33998, 139718, 127729, 56101, 12304, 132414, 182591, 1
[1, 2, 2, 3, 3, 4, 4, 6, 7, 8, 8, 9, 13, 15, 16, 17, 19, 20, 21, 22, 2
Tempo de execução (crescente): 3.724 segundos
[199998, 199995, 199993, 199992, 199989, 199988, 199987, 199986, 19998
[1, 2, 2, 3, 3, 4, 4, 6, 7, 8, 8, 9, 13, 15, 16, 17, 19, 20, 21, 22, 2
Tempo de execução (decrescente - crescente): 6.622 segundos
[1, 2, 2, 3, 3, 4, 4, 6, 7, 8, 8, 9, 13, 15, 16, 17, 19, 20, 21, 22, 2
[1, 2, 2, 3, 3, 4, 4, 6, 7, 8, 8, 9, 13, 15, 16, 17, 19, 20, 21, 22, 2
Tempo de execução (crescente-crescente): 2.41 segundos
CONSTRUÍDO COM SUCESSO (tempo total: 18 segundos)
```

Por ser um método de ordenação mais eficiente, pode ser visto que o tempo gasto, em qualquer um dos processos foi bem mais rápido do que o bubble sort.

5. **Insertion Sort**

   O terceiro e o último modelo de ordenação que apresentaremos é referente ao Insertion Sort. Esse modelo de ordenação ordena de forma em que, começa a

passagem pelo vetor sempre do segundo elemento, pois ele sempre compara dois elementos do vetor, da DIREITA PARA A ESQUERDA, isso quer dizer que, ele compara dois elementos. Ele é bem mais rápido do que os outros estilos de ordenação por conta de que a alteração é feita diretamente ENQUANTO o algoritmo passa pelo vetor.

Abaixo iremos apresentar o código para a ordenação de forma crescente:

```
private static int[] insertionSortCrescente(int a[]) {
        int[] arrayNovo = new int[100000];
        for (int i = 0; i < a.length; i++) {
            arrayNovo[i] = a[i];
        }
        for (int i = 1; i < arrayNovo.length; i++) {
            int aux = arrayNovo[i];
            int j = i;
            while ((j > 0) && (arrayNovo[j-1] > aux)){
                arrayNovo[j] = arrayNovo[j-1];
                j -= 1;
            }
            arrayNovo[j] = aux;
          }
        int[] arrayNovoCrescente = new int[100000];
        for (int i = 0; i < arrayNovo.length; i++) {
            arrayNovoCrescente[i] = arrayNovo[i];
        }
        return arrayNovoCrescente;
    }
```

Novamente, utilizando-se do mesmo processo, chamamos o método e computamos o tempo:

```
System.out.println(Arrays.toString(arrayAle));
        long inicio7 = System.currentTimeMillis();
        int[] arrayCrescente2 = insertionSortCrescente(arrayAle);
        long termino7 = System.currentTimeMillis();
        double tempo7 = (termino7 - inicio7) / 1000.0;
```

```
        System.out.println(Arrays.toString(arrayCrescente2));
        System.out.println("Tempo de execução (crescente): " + tempo7 + "
segundos");
```

A seguir, o código para a ordenação de forma decrescente:

```
private static int[] insertionSortDecrescente(int a[]) {
        int[] arrayNovo = new int[100000];
        for (int i = 0; i < a.length; i++) {
            arrayNovo[i] = a[i];
        }
        for (int i = 1; i < arrayNovo.length; i++) {
            int aux = arrayNovo[i];
            int j = i;
            while ((j > 0) && (arrayNovo[j-1] < aux)){
                arrayNovo[j] = arrayNovo[j-1];
                j -= 1;
            }
            arrayNovo[j] = aux;
          }
        int[] arrayNovoDecrescente = new int[100000];
        for (int i = 0; i < arrayNovo.length; i++) {
            arrayNovoDecrescente[i] = arrayNovo[i];
        }
        return arrayNovoDecrescente;
    }
```

Chamando o método, e computando o tempo:

```
int[] arrayDecrescente2 = insertionSortDecrescente(arrayAle);
        System.out.println(Arrays.toString(arrayDecrescente2));
        long inicio8 = System.currentTimeMillis();
        int[]                    arrayDecrescenteCrescente2                    =
insertionSortCrescente(arrayDecrescente2);
        long termino8 = System.currentTimeMillis();
        double tempo8 = (termino8 - inicio8) / 1000.0;
        System.out.println(Arrays.toString(arrayDecrescenteCrescente2));
```

System.out.println("Tempo de execução (decrescente - crescente): " + tempo8 + " segundos");

E por fim, do mesmo modo que os outros métodos, chamaremos a função crescente novamente e passaremos como parâmetro o vetor já ordenado em forma crescente:

```
System.out.println(Arrays.toString(arrayCrescente2));
        long inicio9 = System.currentTimeMillis();
        int[] arrayCrescenteCrescente2 = insertionSortCrescente(arrayCrescente2);
        long termino9 = System.currentTimeMillis();
        double tempo9 = (termino9 - inicio9) / 1000.0;
        System.out.println(Arrays.toString(arrayCrescenteCrescente2));
        System.out.println("Tempo de execução (crescente-crescente): " + tempo9 + " segundos");
```

O resultados deste método foi o seguinte:

```
run:
[63956, 185462, 127022, 105028, 188316, 33014, 26300, 71389, 122275
[0, 5, 5, 6, 7, 11, 13, 15, 18, 20, 20, 25, 26, 28, 29, 31, 32, 36,
Tempo de execução (crescente): 0.902 segundos
[199998, 199997, 199990, 199983, 199979, 199979, 199975, 199975, 19
[0, 5, 5, 6, 7, 11, 13, 15, 18, 20, 20, 25, 26, 28, 29, 31, 32, 36,
Tempo de execução (decrescente - crescente): 1.655 segundos
[0, 5, 5, 6, 7, 11, 13, 15, 18, 20, 20, 25, 26, 28, 29, 31, 32, 36,
[0, 5, 5, 6, 7, 11, 13, 15, 18, 20, 20, 25, 26, 28, 29, 31, 32, 36,
Tempo de execução (crescente-crescente): 0.002 segundos
CONSTRUÍDO COM SUCESSO (tempo total: 5 segundos)
```

Como pode ser visto na foto acima, este método é o mais rápido entre todos os outros citados anteriormente.

6. Considerações Finais

Neste tópico, apresentaremos as considerações finais sobre o que levamos como aprendizado neste trabalho. No caso, percebemos o QUÃO importante é ter uma boa seleção de estilos de códigos, e modelos de otimização, pois isso pode sacrificar as vezes até mesmo mais de 90% do desempenho de um código. Por conta disso, um bom estilo de otimização, nunca deve ser desprezada na hora de criar um software, ou até mesmo um simples algoritmo. Isso foi essencial para o nosso aprendizado nessa disciplina, pois mostra o

tamanho da importância da disciplina de estrutura de dados para a criação e desenvolvimento de uma aplicação.

---

7. Bibliografia

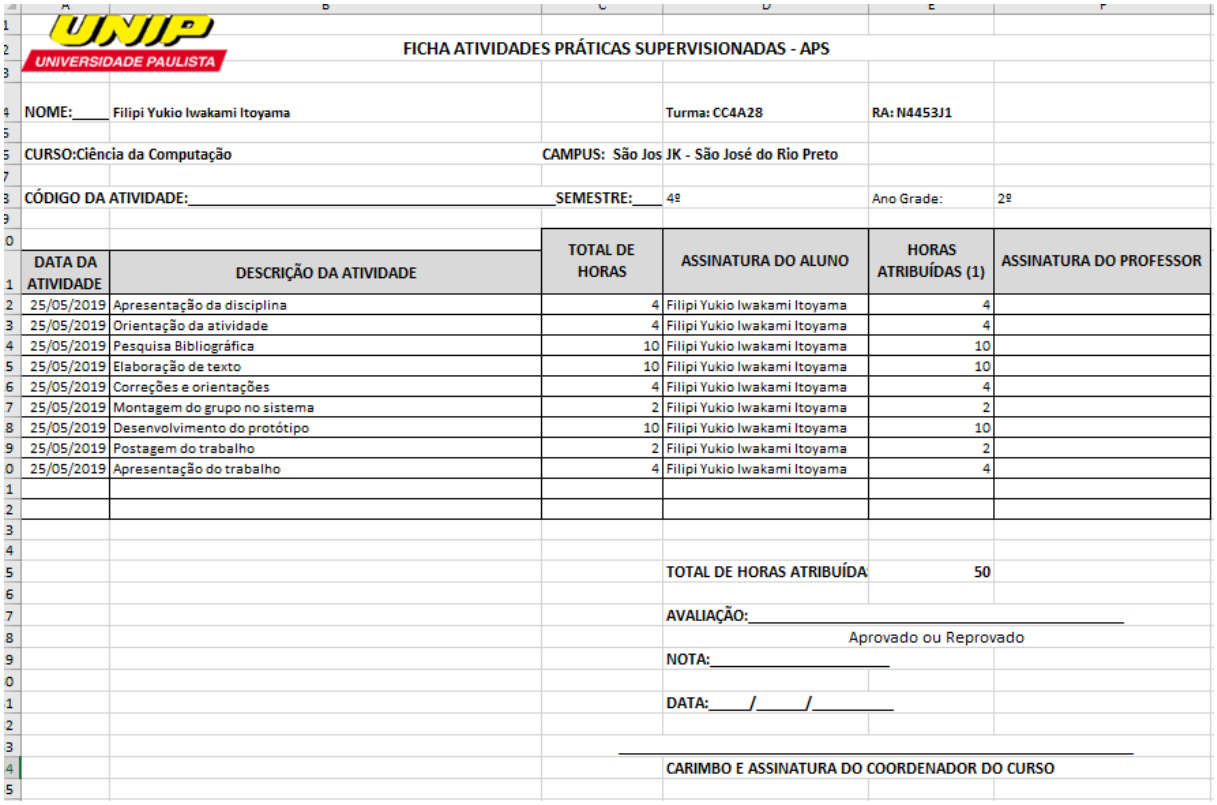
https://www.devmedia.com.br/entendendo-o-algoritmo-bubble-sort-em-java/24812

https://pt.wikipedia.org/wiki/Bubble_sort

https://pt.wikipedia.org/wiki/Selection_sort

https://pt.wikipedia.org/wiki/Insertion_sort

8. Fichas de Atividades Práticas Supervisionadas



UNIP UNIVERSIDADE PAULISTA

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME:____ Filipi Yukio Iwakami Itoyama — Turma: CC4A28 — RA: N4453J1

CURSO:Ciência da Computação — CAMPUS: São Jos JK - São José do Rio Preto

CÓDIGO DA ATIVIDADE:_____ SEMESTRE:___ 4º — Ano Grade: 2º

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
25/05/2019	Apresentação da disciplina	4	Filipi Yukio Iwakami Itoyama	4	
25/05/2019	Orientação da atividade	4	Filipi Yukio Iwakami Itoyama	4	
25/05/2019	Pesquisa Bibliográfica	10	Filipi Yukio Iwakami Itoyama	10	
25/05/2019	Elaboração de texto	10	Filipi Yukio Iwakami Itoyama	10	
25/05/2019	Correções e orientações	4	Filipi Yukio Iwakami Itoyama	4	
25/05/2019	Montagem do grupo no sistema	2	Filipi Yukio Iwakami Itoyama	2	
25/05/2019	Desenvolvimento do protótipo	10	Filipi Yukio Iwakami Itoyama	10	
25/05/2019	Postagem do trabalho	2	Filipi Yukio Iwakami Itoyama	2	
25/05/2019	Apresentação do trabalho	4	Filipi Yukio Iwakami Itoyama	4	

TOTAL DE HORAS ATRIBUÍDA 50

AVALIAÇÃO:_____

Aprovado ou Reprovado

NOTA:_____

DATA:____/_____/_____

_____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIVERSIDADE PAULISTA

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME:____ Luiz Gabriel Zeferino Duarte Turma: CC4A28 RA:N454CD8

CURSO:Ciência da Computação CAMPUS: São Jos JK - São José do Rio Preto

CÓDIGO DA ATIVIDADE:_____SEMESTRE:___ 4º Ano Grade: 2º

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
25/05/2019	Apresentação da disciplina	4	Luiz Gabriel Zeferino Duarte	4	
25/05/2019	Orientação da atividade	4	Luiz Gabriel Zeferino Duarte	4	
25/05/2019	Pesquisa Bibliográfica	10	Luiz Gabriel Zeferino Duarte	10	
25/05/2019	Elaboração de texto	10	Luiz Gabriel Zeferino Duarte	10	
25/05/2019	Correções e orientações	4	Luiz Gabriel Zeferino Duarte	4	
25/05/2019	Montagem do grupo no sistema	2	Luiz Gabriel Zeferino Duarte	2	
25/05/2019	Desenvolvimento do protótipo	10	Luiz Gabriel Zeferino Duarte	10	
25/05/2019	Postagem do trabalho	2	Luiz Gabriel Zeferino Duarte	2	
25/05/2019	Apresentação do trabalho	4	Luiz Gabriel Zeferino Duarte	4	

TOTAL DE HORAS ATRIBUÍDA 50

AVALIAÇÃO:_____

Aprovado ou Reprovado

NOTA:_____

DATA:_____/_____/_____

_____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

UNIVERSIDADE PAULISTA

FICHA ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME:____ William Rossi do Carmo Ruiz Turma: CC4A28 RA:n473gf8

CURSO:Ciência da Computação CAMPUS: São Jos JK - São José do Rio Preto

CÓDIGO DA ATIVIDADE:_____SEMESTRE:___ 4º Ano Grade: 2º

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
25/05/2019	Apresentação da disciplina	4	William Rossi do Carmo Ruiz	4	
25/05/2019	Orientação da atividade	4	William Rossi do Carmo Ruiz	4	
25/05/2019	Pesquisa Bibliográfica	10	William Rossi do Carmo Ruiz	10	
25/05/2019	Elaboração de texto	10	William Rossi do Carmo Ruiz	10	
25/05/2019	Correções e orientações	4	William Rossi do Carmo Ruiz	4	
25/05/2019	Montagem do grupo no sistema	2	William Rossi do Carmo Ruiz	2	
25/05/2019	Desenvolvimento do protótipo	10	William Rossi do Carmo Ruiz	10	
25/05/2019	Postagem do trabalho	2	William Rossi do Carmo Ruiz	2	
25/05/2019	Apresentação do trabalho	4	William Rossi do Carmo Ruiz	4	

TOTAL DE HORAS ATRIBUÍDA 50

AVALIAÇÃO:_____

Aprovado ou Reprovado

NOTA:_____

DATA:_____/_____/_____

_____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO