

**Alef Ribeiro de Moura, F09ADC   
João Victor Molico Villella, N346868**

**João Victor dos Santos Peres, D8738J7**

**Paulo Christian Mariano, F02BHG3**

**DESENVOLVIMENTO DE SISTEMA PARA ANÁLISE DE PERFORMANCE DE ALGORITMOS DE ORDENAÇÃO DE DADOS**

**São Paulo**

**2020**

**Alef Ribeiro de Moura F09ADC   
João Victor Molico Villella, N346868**

**João Victor dos Santos Peres, D8738J7**

**Paulo Christian Mariano, F02BHG3**

**DESENVOLVIMENTO DE SISTEMA PARA ANÁLISE DE PERFORMANCE DE ALGORITMOS DE ORDENAÇÃO DE DADOS**

Trabalho de Atividade Prática Supervisionada apresentada à Universidade Paulista (UNIP), como exigência para finalização do 3°/4° semestre de Ciência da Computação.

**São PAULO**

**2020**

**Alef Ribeiro de Moura F09ADC  
 João Victor Molico Villella, N346868**

**João Victor dos Santos Peres, D8738J7**

**Paulo Christian Mariano, F02BHG3**

**DESENVOLVIMENTO DE SISTEMA PARA ANÁLISE DE PERFORMANCE DE ALGORITMOS DE ORDENAÇÃO DE DADOS**

Trabalho d Atividade Prática Supervisionada apresentada à Universidade Paulista (UNIP), como exigência para finalização do 3°/4° semestre de Ciência da Computação.

Orientador: Jorge Pirolla

**São PAULO**

**2020**

**UNIVERSIDADE PAULISTA (UNIP)**

Curso de Bacharelado em Ciência da Computação

**Coordenadora: Profª Dr. Eliane Santiago**

**São PAULO**

**2020**

Sumário

1. CAPA 1  
2. SUMÁRIO 5

[3. INTRODUÇÃO 6](#_Toc56535184)

[3.1 Contextualização 6](#_Toc56535185)

[3.2 Problemáticas 6](#_Toc56535186)

[3.3 Objetivo 7](#_Toc56535187)

[3.4 Motivação 7](#_Toc56535188)

[3.5 Organização do Relatório 7](#_Toc56535189)

[4.7 Sobre Educação Ambiental 8](#_Toc56535190)

[5. DESENVOLVIMENTO 8](#_Toc56535191)

[5.1 Ferramentas 10](#_Toc56535192)

[6. REGRAS E FUNCIONAMENTOS DO JOGO 10](#_Toc56535193)

[6.1 Interfaces gráficas em Java 10](#_Toc56535194)

[6.2 Componentes 11](#_Toc56535195)

[6.3 Tratamento de Eventos 13](#_Toc56535196)

[7. CÓDIGO COMENTADO 18](#_Toc56535197)

[8. RESULTADOS OU PROTOTIPAÇÃO 25](#_Toc56535198)

[9. CONCLUSÃO 29](#_Toc56535199)

[11. BIBLIOGRAFIA 30](#_Toc56535200)

[12. FICHAS APS 31](#_Toc56535201)

# 3. INTRODUÇÃO

## 3.1 Contextualização

A linguagem C#, que foi selecionada para a APS, suporta Estruturas (Structure), Sobrecarga de operadores e diretivas de pré-processadores. A linguagem **Java** não suporta esses recursos. Na linguagem **C#** podemos facilmente chamar funções API do Windows e acessar componentes COM. Na linguagem **Java** essas tarefas são muito mais difíceis de realizar.

Atualmente C# é a linguagem de programação mais utilizada para desenvolvimento de APIs, com isso vimos a oportunidade de nos aprofundarmos mais em uma linguagem nova que pode ampliar nossos horizontes e perspectivas perante o mercado de trabalho.

## 3.2 Problemáticas

Para sanar a questão da organização e ordenação de dados capturados de diversos lugares da forma mais eficiente possível, criamos um estudo centrado nos principais métodos de ordenação para algoritmos. Dentre estes podemos citar

O quicksort que adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores". Em seguida o quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada.

Bubble Sort que é um algoritmo de ordenação que pode ser aplicado em Arrays e Listas dinâmicas. Se o objetivo é ordenar os valores em forma decrescente, então, a posição atual é comparada com a próxima posição e, se a posição atual for maior que a posição posterior, é realizada a troca dos valores nessa posição.

E Selection sort que é baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o segundo menor valor para a segunda posição e assim sucessivamente, até os últimos dois elementos.

## 3.3 Objetivo

Desenvolver um software que realize ordenação utilizando algoritmos utilizando técnicas de ordenação para organizar dados obtidos a partir de dados gerados pelo geoprocessamento de imagens da floresta amazônica que permite a fiscalização de ações de crimes ambientais. Com isso conseguiremos obter dados concretos da forma mais eficiente possível.

O sistema computacional está sendo desenvolvido utilizando a linguagem de programação Microsoft Visual C#.

## 3.4 Motivação

Além da oportunidade de desenvolver um projeto com C# que faz parte do NET Core. Com ela nós poderemos escrever e criar aplicações desktop, web, serviços backend, aplicações mobiles multiplataformas (nativa) sem precisar conhecer muito de outros recursos a não ser um pouco de JavaScript e TypeScript.

Logo. Será um grande alavancador para o crescimento de desenvolvimento pessoal e profissional do grupo.

## 3.5 Organização do Relatório

Durante as pesquisas e elaboração do projeto, comentários periódicos foram adicionados e testes para checar as funcionalidades em cada etapa, a fim de acumular conteúdo para a documentação e elaboração do relatório.

# REFERENCIAL TEÓRICO

Nestes trabalhos utilizamos os seguintes métodos:

## QUICKSORT

O Algoritmo Quicksort, criado por *C. A. R. Hoare* em 1960, é o método de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.

O Quicksort é o algoritmo mais eficiente na ordenação por comparação. Nele se escolhe um elemento chamado de pivô, a partir disto é organizada a lista para que todos os números anteriores a ele sejam menores que ele, e todos os números posteriores a ele sejam maiores que ele. Ao final desse processo o número pivô já está em sua posição final. Os dois grupos desordenados recursivamente sofreram o mesmo processo até que a lista esteja ordenada.   
  
Determina-se um elemento pivô. O pivô é posicionado dentro do vetor de tal forma que, todos à esquerda do pivô são menores que ele e, todos à direita do pivô são maiores. O pivô "divide" o vetor em dois subvetores. Recursivamente o quick sort é realizado na primeira metade do vetor e na segunda metade.

Algoritmo: Seja **x** o vetor a ser ordenado e **n** o número de elementos de **x**. Seja **a** um elemento de **x** escolhido ao acaso (por exemplo, **a=x[0]**). Suponha que os elementos de **x** estão divididos de tal forma que **a** é colocado na posição **j** e as seguintes condições são verdadeiras:

Todos os elementos nas posições de **0** a **j-1** são menores que **a**.

Todos os elementos nas posições de **j+1** a **n-1** são maiores ou iguais a **a**.

Então **a** está na posição correta no vetor. Se este processo for repetido para os sub-vetores **x[0]** a **x[j-1]** e **x[j+1]** a **x[n-1]**, o resultado é o vetor ordenado.

Observe a figura: Vetor original com elemento desordenados.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 57 | 86 | 48 | 37 | 24 | 92 | 12 |

O pivô (primeiro elemento do vetor) é escolhido.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 57 | 86 | 48 | 37 | 24 | 92 | 12 |
| pivô |  |  |  |  |  |  |  |

A busca por elementos maiores que o pivô se inicia à esquerda do trecho do vetor que está sendo ordenado, e a busca por elementos menores que o pivô se inicia à direita.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 57 | 86 | 48 | 37 | 24 | 92 | 12 |
| esq |  |  |  |  |  |  | dir |

25 não é maior que o pivô (25). A busca por um elemento maior que o pivô continua à esquerda.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 57 | 86 | 48 | 37 | 24 | 92 | 12 |
| pivô | esq |  |  |  |  |  | dir |

57 é encontrado. A busca por um elemento menor que o pivô à direita encontra o 12.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 57 | 86 | 48 | 37 | 24 | 92 | 12 |
| pivô | esq |  |  |  |  |  | dir |

Troca o elemento da esquerda com o da direita.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 12 | 86 | 48 | 37 | 24 | 92 | 57 |
| pivô | esq |  |  |  |  |  | dir |

Continua a busca por elementos maiores que o pivô (25) a partir da esquerda e por elementos menores a partir da direita.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 12 | 86 | 48 | 37 | 24 | 92 | 57 |
|  |  | esq |  |  | dir |  |  |

Troca.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 12 | 24 | 48 | 37 | 86 | 92 | 57 |
| pivô |  | esq |  |  | dir |  |  |

O processo continua.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 25 | 12 | 24 | 48 | 37 | 86 | 92 | 57 |
| pivô |  | dir | esq |  |  |  |  |

Ops! Ocorreu um cruzamento da posição esquerda com a posição direita. Troca o pivô com o elemento da posição direita.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 24 | 12 | 25 | 48 | 37 | 86 | 92 | 57 |

O subvetor à esquerda do pivô contém os elementos menores que o pivô e o subvetor à direita do pivô contém os elementos maiores que o pivô. O processo se repete para o subvetor da esquerda do vetor e para o subvetor da direita.

O 24 é pivô no subvetor à esquerda.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 24 | 12 | 25 | 48 | 37 | 86 | 92 | 57 |
| esq | dir |  |  |  |  |  |  |

E o processa continua no subvetor à esquerda.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 48 | 37 | 86 | 92 | 57 |
| esq | dir |  |  |  |  |  |  |

E a direita, onde o pivô é o 48.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 48 | 37 | 86 | 92 | 57 |
|  |  |  | esq |  |  |  | dir |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 48 | 37 | 86 | 92 | 57 |
|  |  |  |  | dir | esq |  |  |

Ops! Cruzamanto da posiçães direita e esquerda. Troca o pivô com o elemento da posição direita.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 37 | 48 | 86 | 92 | 57 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 37 | 48 | 86 | 92 | 57 |
|  |  |  |  |  | esq |  | dir |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 37 | 48 | 86 | 92 | 57 |
|  |  |  |  |  |  | esq | dir |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 37 | 48 | 86 | 57 | 92 |
|  |  |  |  |  |  | esq | dir |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 37 | 48 | 86 | 57 | 92 |
|  |  |  |  |  |  | dir | esq |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 37 | 48 | 57 | 86 | 92 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 12 | 24 | 25 | 37 | 48 | 57 | 86 | 92 |

O processo de ordenação termina!

Veja o algoritmo a seguir:

#include

void lerVet( int \*p, int t ){

int i;

for ( i=0; i<="" t;="" printf("\tposicao="" %d:="" %d\n",i,\*p);="" trocar="" (int="" \*pv,="" x,="" y="" aux="pv[x];" pv[x]="pv[y];" pv[y]="aux;" divide(="" \*v,="" inf,="" sup="" )="" {="" pivo,="" esq,="" dir;="" pivo="v[inf];" esq="inf;" dir="sup;" while="" v[esq]="" &&="" esq++;="" v[dir]=""> pivo )

dir--;

if ( esq < dir )

trocar(v,esq,dir);

}

v[inf] = v[dir];

v[dir] = pivo ;

return dir;

}

void quickSort( int \*p, int inf, int sup ) {

int posPivo; // posição do pivô

if ( inf >= sup )

return;

posPivo= divide(p,inf,sup);

quickSort(p,inf,posPivo-1);

quickSort(p,posPivo+1,sup);

}

void main(){

int \*p, tam;

printf("Quantidade de elementos do vetor?");

scanf("%d",&tam);

p = (int\*) malloc(tam \* sizeof(int));

printf("\nDigite o conteudo do vetor:\n");

lerVet(p, tam);

printf("\nConteudo digitado para o vetor:\n");

mostrarVet(p, tam);

printf("\nOrdenando o vetor...\n");

quickSort(p, 0, tam-1);

printf("\nConteudo do vetor ja ordenado:\n");

mostrarVet(p, tam);

free(p);

}

## Algoritmo Merge Sort

Criado em 1945 pelo matemático americano John Von Neumann o Mergesort é um exemplo de algoritmo de ordenação que faz uso da estratégia “dividir para conquistar” para resolver problemas. É um método estável e possui complexidade C(n) = O(n log n) para todos os casos.

Esse algoritmo divide o problema em pedaços menores, resolve cada pedaço e depois junta (merge) os resultados. O vetor será dividido em duas partes iguais, que serão cada uma divididas em duas partes, e assim até ficar um ou dois elementos cuja ordenação é trivial.

Para juntar as partes ordenadas os dois elementos de cada parte são separados e o menor deles é selecionado e retirado de sua parte. Em seguida os menores entre os restantes são comparados e assim se prossegue até juntar as partes.

A idéia básica do Merge Sort é criar uma sequência ordenada a partir de duas outras também ordenadas. Para isso, o algoritmo Merge Sort divide a sequência original em pares de dados, agrupa estes pares na ordem desejada; depois as agrupa as sequências de pares já ordenados, formando uma nova sequência ordenada de quatro elementos, e assim por diante, até ter toda a sequência ordenada.

Algoritmo:  
Os três passos úteis dos algoritmos dividir-para-conquistar, que se aplicam ao Merge Sort são:

Dividir: Dividir os dados em subsequências pequenas;  
Este passo é realizado recursivamente, iniciando com a divisão do vetor de n elementos em duas metades, cada uma das metades é novamente dividida em duas novas metades e assim por diante, até que não seja mais possível a divisão (ou seja, sobrem n vetores com um elemento cada).

Conquistar: Classificar as duas metades recursivamente aplicando o merge sort;

Combinar: Juntar as duas metades em um único conjunto já classificado.

Para completar a ordenação do vetor original de n elementos, faz-se o merge ou a fusão dos sub-vetores já ordenados.

A desvantagem do Merge Sort é que requer o dobro de memória, ou seja, precisa de um vetor com as mesmas dimensões do vetor que está sendo classificado.

Observe a figura: Vetor original com elemento desordenados.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 53 | 25 | 46 | 32 | 23 | 37 | 41 | 17 | 10 |

O vetor original é subdividido em dois vetores.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 53 | 25 | 46 | 32 | 23 | 37 | 41 | 17 | 10 |

Cada um dos subvetores é novamente dividido.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 53 | 25 | 46 | 32 | 23 | 37 | 41 | 17 | 10 |

E assim por diante.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 53 | 25 | 46 | 32 | 23 | 37 | 41 | 17 | 10 |
| 53 | 25 | 46 | 32 | 23 | 37 | 41 | 17 | 10 |

Após todo o processo de divisão, ocorre o processo da fusão ordenada dos subvetores. O subvetor (53) com o subvetor (25). Ordenando os dois.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 25 | 53 | 46 | 32 | 23 | 37 | 41 | 17 | 10 |

O subvetor (25, 53) com o subvetor (46). Ordenando os dois.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 25 | 46 | 53 | 32 | 23 | 37 | 41 | 17 | 10 |

O subvetor (32) com o subvetor (23).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 25 | 46 | 53 | 23 | 32 | 37 | 41 | 17 | 10 |

O subvetor (25,46,53) com o subvetor (23,32). Ordenando os dois.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 23 | 25 | 32 | 46 | 53 | 37 | 41 | 17 | 10 |

O mesmo processo se repete no subvetor (37, 41, 17, 10).

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 23 | 25 | 32 | 46 | 53 | 37 | 41 | 17 | 10 |
| 23 | 25 | 32 | 46 | 53 | 37 | 41 | 10 | 17 |
| 23 | 25 | 32 | 46 | 53 | 10 | 17 | 37 | 41 |

Os subvetores resultantes (23,25, 32, 36,53) e (10, 17, 37, 41) são fundidos ordenandos durante a fusão.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 10 | 17 | 23 | 25 | 32 | 37 | 41 | 46 | 53 |

O processo de ordenação termina!

Veja o algoritmo a seguir:

#include

void lerVet( int \*p, int t ){

int i;

for ( i=0; i<="" t;="" printf("\tposicao="" %d:="" %d\n",i,\*p);="" merge(="" )="" {="" meio;="" i,="" j,="" k;="" int\*="" tmp;="" tmp="(int\*)" malloc(t="" \*="" sizeof(int));="" if="" null="" exit(1);="" meio="t" 2;="" j="meio;" k="0;" while="" &&="" (p[i]="" p[j]="" tmp[k]="p[i];" ++i;="" else="" ++j;="" ++k;="" (i="=" meio)="" ++i="" p[i]="tmp[i];" free(tmp);="" mergesort(=""> 1 ) {

meio = t / 2;

mergeSort(p, meio);

mergeSort(p + meio, t - meio);

merge(p, t);

}

}

void main(){

int \*p, tam;

printf("Quantidade de elementos do vetor?");

scanf("%d",&tam);

p = (int\*) malloc(tam \* sizeof(int));

printf("\nDigite o conteudo do vetor:\n");

lerVet(p, tam);

printf("\nConteudo digitado para o vetor:\n");

mostrarVet(p, tam);

printf("\nOrdenando o vetor...\n");

mergeSort(p, tam);

printf("\nConteudo do vetor ja ordenado:\n");

mostrarVet(p, tam);

free(p);

}

## Cocktail Sort

Cocktail Sort é uma variação do tipo Bubble. O algoritmo de classificação de bolha sempre atravessa os elementos da esquerda e move o maior elemento para sua posição correta na primeira iteração e o segundo maior na segunda iteração e assim por diante. O Cocktail Sort atravessa uma determinada matriz em ambas as direções alternadamente.

**Algoritmo:**

Cada iteração do algoritmo é dividida em 2 estágios:

O primeiro estágio percorre a matriz da esquerda para a direita, assim como o Bubble Sort. Durante o loop, os itens adjacentes são comparados e se o valor à esquerda for maior que o valor à direita, então os valores são trocados. No final da primeira iteração, o maior número residirá no final da matriz.

O segundo estágio percorre o array na direção oposta - começando do item imediatamente antes do item classificado mais recentemente e voltando para o início do array. Aqui também, os itens adjacentes são comparados e trocados, se necessário.

Exemplo: vamos considerar uma matriz de exemplo (5 1 4 2 8 0 2)

Primeiro passe para a frente:

(5 1 4 2 8 0 2)? (1 5 4 2 8 0 2), Trocar desde 5> 1

(1 5 4 2 8 0 2)? (1 4 5 2 8 0 2), Trocar desde 5> 4

(1 4 5 2 8 0 2)? (1 4 2 5 8 0 2), Trocar desde 5> 2

(1 4 2 5 8 0 2)? (1 4 2 5 8 0 2)

(1 4 2 5 8 0 2)? (1 4 2 5 0 8 2), Trocar desde 8> 0

(1 4 2 5 0 8 2)? (1 4 2 5 0 2 8), Trocar desde 8> 2

Após a primeira passagem para frente, o maior elemento da matriz estará presente no último índice da matriz.

Segundo passe para a frente:

(0 1 4 2 5 2 8)? (0 1 4 2 5 2 8)

(0 1 4 2 5 2 8)? (0 1 2 4 5 2 8), Trocar desde 4> 2

(0 1 2 4 5 2 8)? (0 1 2 4 5 2 8)

(0 1 2 4 5 2 8)? (0 1 2 4 2 5 8), Trocar desde 5> 2

Segundo passe para trás:

(0 1 2 4 2 5 8)? (0 1 2 2 4 5 8), Trocar desde 4> 2

Agora, a matriz já está classificada, mas nosso algoritmo não sabe se ela foi concluída. O algoritmo precisa completar toda essa passagem sem nenhuma troca para saber se está classificado.

(0 1 2 2 4 5 8)? (0 1 2 2 4 5 8)

(0 1 2 2 4 5 8)? (0 1 2 2 4 5 8

Abaixo está a implementação do algoritmo acima:

/ C# program for implementation of Cocktail Sort

using System;

class GFG {

    static void cocktailSort(int[] a)

    {

        bool swapped = true;

        int start = 0;

        int end = a.Length;

        while (swapped == true) {

            // reset the swapped flag on entering the

            // loop, because it might be true from a

            // previous iteration.

            swapped = false;

            // loop from bottom to top same as

            // the bubble sort

            for (int i = start; i < end - 1; ++i) {

                if (a[i] > a[i + 1]) {

                    int temp = a[i];

                    a[i] = a[i + 1];

                    a[i + 1] = temp;

                    swapped = true;

                }

            }

            // if nothing moved, then array is sorted.

            if (swapped == false)

                break;

            // otherwise, reset the swapped flag so that it

            // can be used in the next stage

            swapped = false;

            // move the end point back by one, because

            // item at the end is in its rightful spot

            end = end - 1;

            // from top to bottom, doing the

            // same comparison as in the previous stage

            for (int i = end - 1; i >= start; i--) {

                if (a[i] > a[i + 1]) {

                    int temp = a[i];

                    a[i] = a[i + 1];

                    a[i + 1] = temp;

                    swapped = true;

                }

            }

            // increase the starting point, because

            // the last stage would have moved the next

            // smallest number to its rightful spot.

            start = start + 1;

        }

    }

    /\* Prints the array \*/

    static void printArray(int[] a)

    {

        int n = a.Length;

        for (int i = 0; i < n; i++)

            Console.Write(a[i] + " ");

        Console.WriteLine();

    }

    // Driver method

    public static void Main()

    {

        int[] a = { 5, 1, 4, 2, 8, 0, 2 };

        cocktailSort(a);

        Console.WriteLine("Sorted array ");

        printArray(a);

    }

}

## 4.7 Sobre Educação Ambiental

No nosso projeto queremos mostrar como é importante preservamos o meio ambiente, e educar a sociedade e conscientizar sobre a poluição, pois não somente os seres humanos mas como animais são prejudicados, existem diversas formas de poluir um ambiente jogando lixo fora do local apropriado, derramar substancias químicas e esgoto em agua potável, queimada de florestas, desmatamento, apesar de serem coisas diferentes são diversas formas que acabam poluindo um ambiente puro e limpo, pois interfere no eco sistema do local permanentemente, um exemplo que temos na própria cidade de são Paulo são os rios pinheiros e tiête que eram completamente limpos a anos atrás, e o rio foi completamente tomado por lixo e esgoto, tornando a vida animal completamente inexistente e hoje em dia o rio é considerado morto por não haver oxigênio na água. Devemos descartar todo e qualquer tipo de material no seu local adequado, pois assim a harmonia entre a natureza se mante e tanto humanos e animas conseguem viver em harmonia, tornando o planeta sustentável para todos.

E dentro do projeto temos um animal (cobra) que é geneticamente modificado e se alimenta de lixo e mantém a cidade ao fundo limpa, e a mensagem a ser passada e que todos devemos preservar não só para o nosso bem, mas sim para o bem de todos os seres vivos existem, pois de certa forma todos precisam um dos outros para viver.

# 5. DESENVOLVIMENTO

No desenvolvimento do jogo foi utilizado uma cpu com o processador Intel®Core™ i7, RAM de 16.00 GB, sistema operacional Windows 10 Pro x64 e HD de 1TB e um ssd de 120 gb. O principal software utilizado no processo foi o Microsoft Visual Studio Community

O programa foi escrito em c# utilizando das bibliotecas nativas do dot.net e algumas bibliotecas auxiliares como por exemplo json.net para possibilitar uma conversão de json para objetos.

A fonte das informações externas escolhida foi json, que se assemelha a um xml ou arquivo txt com uma sintaxe simples visando organizar e rotular as informações em série. O processo para conversão dos dados json para objeto c# é chamado de desserialização, assim como o processo inverso é chamado de serialização. Desta forma as informações podem ser facilmente reconhecidas no layout modelo de uma classe compatível.

Essas informações resgatadas são agrupadas em coleções genéricas em lista para percorrer cada objeto com um índice, assemelhando-se a um tipo de vetor de objetos.

Dentre o conjunto de atributos disponíveis, o usuario irá escolher um atributo desejado para ordenar na interface construída com windows form, na sequência, um tipo de data irá armazenar a data com tempo atual, executar o algoritmo de ordenação, e novamente será obtido uma nova data com tempo atual que será subtraida pela inicial sendo passada para o tipo timespan com o método milisegundos para obter como resultado final apenas a diferença de milissegundos do inicio até o final.

Os métodos de ordenação escolhidos foram QuickSort, cockTailSort, MergeSort.

## 5.1 Ferramentas

O programa foi escrito em c# utilizando das bibliotecas nativas do dot.net e algumas bibliotecas auxiliares como por exemplo json.net para possibilitar uma conversão de json para objetos.

A fonte das informações externas escolhida foi json, que se assemelha a um xml ou arquivo txt com uma sintaxe simples visando organizar e rotular as informações em série. O processo para conversão dos dados json para objeto c# é chamado de desserialização, assim como o processo inverso é chamado de serialização. Desta forma as informações podem ser facilmente reconhecidas no layout modelo de uma classe compatível.

Essas informações resgatadas são agrupadas em coleções genéricas em lista para percorrer cada objeto com um índice, assemelhando-se a um tipo de vetor de objetos.

Dentre o conjunto de atributos disponíveis, o usuario irá escolher um atributo desejado para ordenar na interface construída com windows form, na sequência, um tipo de data irá armazenar a data com tempo atual, executar o algoritmo de ordenação, e novamente será obtido uma nova data com tempo atual que será subtraida pela inicial sendo passada para o tipo timespan com o método milisegundos para obter como resultado final apenas a diferença de milissegundos do inicio até o final.

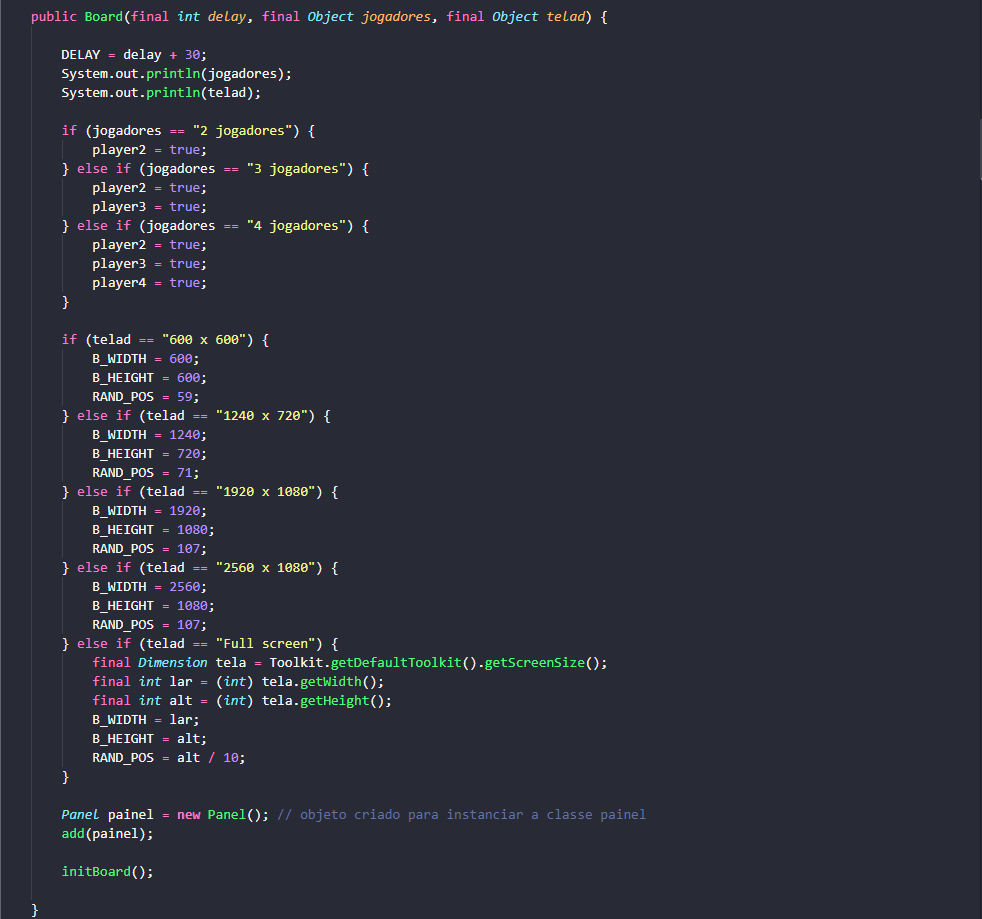
Os métodos de ordenação escolhidos foram QuickSort, cockTailSort, MergeSort.

# 7. CÓDIGO COMENTADO

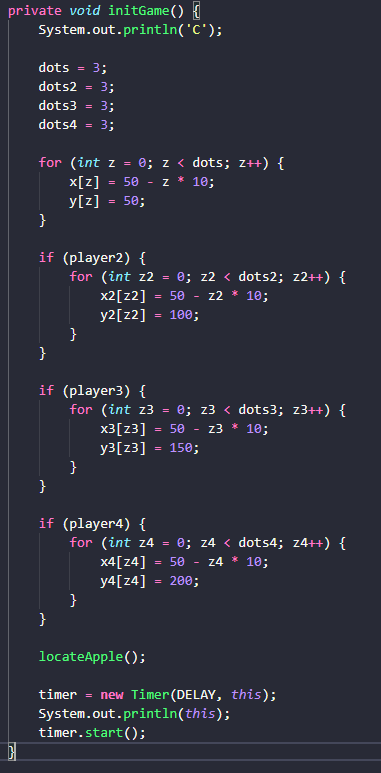
Passagem de parametros da interface gráfica para o jogo



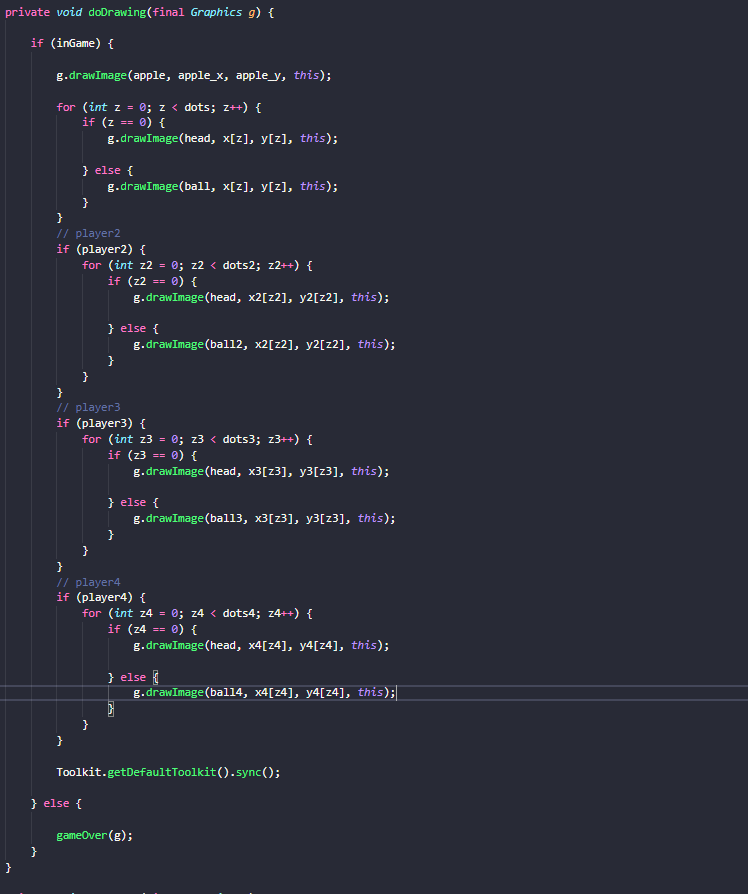
Verificação e ajuste antes de iniciar o jogo de acordo com os parâmetros recebidos



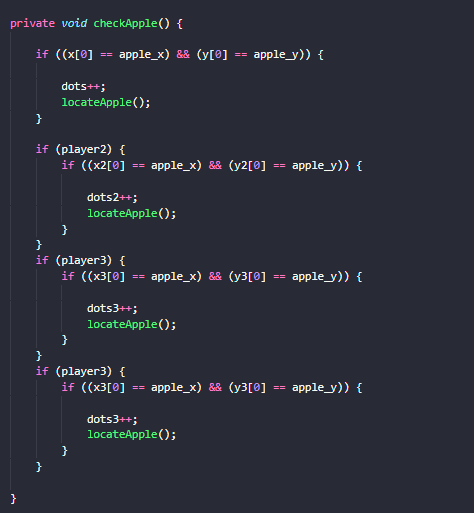
Local de aparição dos jogadores



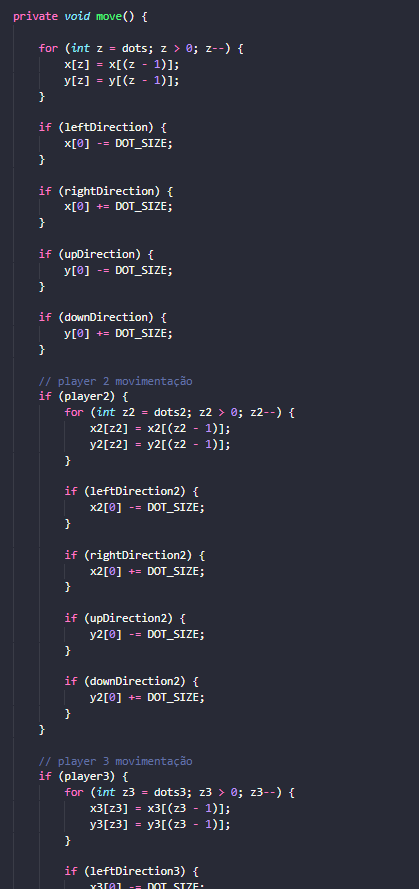
Adição de grafico aos jogadores



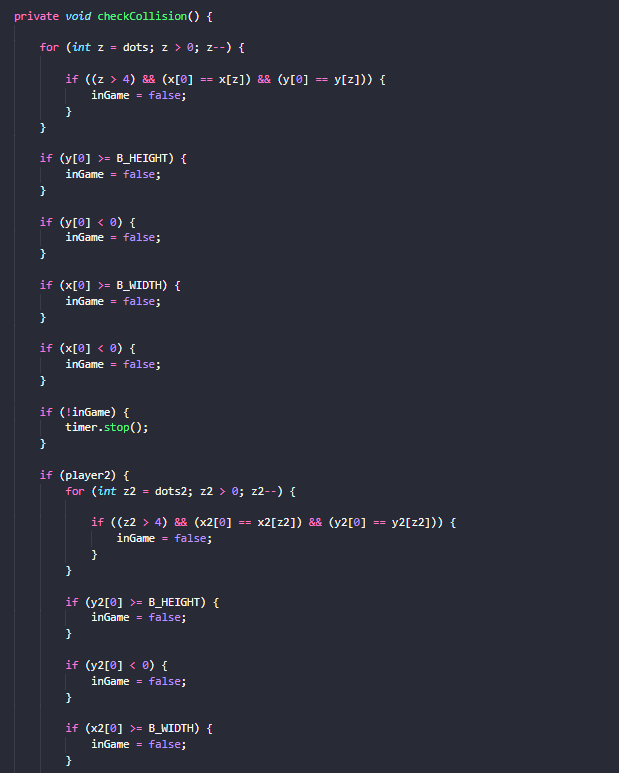
Verificar se a cobra pegou o lixo e qual cobra pegou



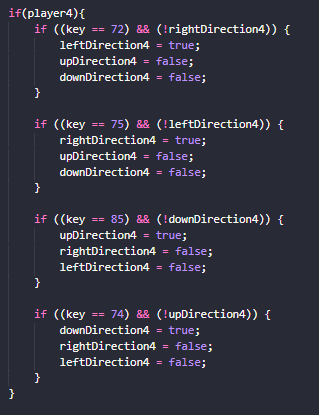
Verificar qual jogador está se movendo



Verificar se a cobra colidiu em seu próprio corpo ou se colidiu em uma parede



Recebimento de eventos do teclado para movimentar a cobra



# 11. BIBLIOGRAFIA

https://docs.microsoft.com/pt-br/dotnet/api/system.collections.generic?view=net-5.0

https://docs.microsoft.com/pt-br/dotnet/api/system.timespan.milliseconds?view=net-5.0

https://www.newtonsoft.com/json

https://www.dotnetperls.com/datetime-now

https://www.c-sharpcorner.com/blogs/date-and-time-format-in-c-sharp-programming1

<http://www.macoratti.net/18/12/c_imodern1.ht>

https://www.devmedia.com.br/csharp-windows-forms-criando-componentes/30149

<https://app.diagrams.net/>

<https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/quick.html>

https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/

https://www.cos.ufrj.br/~rfarias/cos121/aula\_08.html

<https://www.cos.ufrj.br/~rfarias/cos121/aula_07.html>

https://www.geeksforgeeks.org/cocktail-sort/

# 12. FICHAS APS

 FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME:Paulo Cristhian Mariano\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_TURMA: cc4p20\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_RA:F02BHG3\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CURSO:Ciência da computação\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CAMPUS:Norte\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:4°\_\_\_\_\_\_\_TURNO:Noturno\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ CÓDIGO DA ATIVIDADE:77B1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:Quarto\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ANO GRADE:2020\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| DATA DA ATIVIDADE | DESCRIÇÃO DA ATIVIDADE | TOTAL DE HORAS | ASSINATURA DO ALUNO | HORAS ATRIBUÍDAS (1) | ASSINATURA DO PROFESSOR |
| 03/10/2020 | Reunião para alinhamento de equipe | 9 |  |  |  |
| 10/10/2020 | Busca de conteúdo para auxiliar no desenvolvimento | 11 |  |  |  |
| 17/10/2020 | Leitura de livros e aprimoramento de conteúdo específico para melhor desenvolvimento | 11 |  |  |  |
| 24/10/2020 | Inicio do desenvolvimento do algoritmo parte 1 de 2 | 12 |  |  |  |
| 31/10/2020 | Finalização do algoritmo parte 2 de 2 | 12 |  |  |  |
| 07/11/2020 | Inicio do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
| 14/11/2020 | Finalização do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso

 FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: João Victor dos Santos Peres\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_TURMA: cc4p20\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_RA: D8738J7\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CURSO:Ciência da computação\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CAMPUS:Norte\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:4°\_\_\_\_\_\_\_TURNO:Noturno\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ CÓDIGO DA ATIVIDADE:77B1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:Quarto\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ANO GRADE:2020\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| DATA DA ATIVIDADE | DESCRIÇÃO DA ATIVIDADE | TOTAL DE HORAS | ASSINATURA DO ALUNO | HORAS ATRIBUÍDAS (1) | ASSINATURA DO PROFESSOR |
| 03/10/2020 | Reunião para alinhamento de equipe | 9 |  |  |  |
| 10/10/2020 | Busca de conteúdo para auxiliar no desenvolvimento | 11 |  |  |  |
| 17/10/2020 | Leitura de livros e aprimoramento de conteúdo específico para melhor desenvolvimento | 11 |  |  |  |
| 24/10/2020 | Inicio do desenvolvimento do algoritmo parte 1 de 2 | 12 |  |  |  |
| 31/10/2020 | Finalização do algoritmo parte 2 de 2 | 12 |  |  |  |
| 07/11/2020 | Inicio do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
| 14/11/2020 | Finalização do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso

 FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: João Victor Villela\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_TURMA: cc4p20\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_RA: N346868\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CURSO:Ciência da computação\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CAMPUS:Norte\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:3°\_\_\_\_\_\_\_TURNO:Noturno\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ CÓDIGO DA ATIVIDADE:77B1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:Terceiro\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ANO GRADE:2020\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| DATA DA ATIVIDADE | DESCRIÇÃO DA ATIVIDADE | TOTAL DE HORAS | ASSINATURA DO ALUNO | HORAS ATRIBUÍDAS (1) | ASSINATURA DO PROFESSOR |
| 03/10/2020 | Reunião para alinhamento de equipe | 9 |  |  |  |
| 10/10/2020 | Busca de conteúdo para auxiliar no desenvolvimento | 11 |  |  |  |
| 17/10/2020 | Leitura de livros e aprimoramento de conteúdo específico para melhor desenvolvimento | 11 |  |  |  |
| 24/10/2020 | Inicio do desenvolvimento do algoritmo parte 1 de 2 | 12 |  |  |  |
| 31/10/2020 | Finalização do algoritmo parte 2 de 2 | 12 |  |  |  |
| 07/11/2020 | Inicio do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
| 14/11/2020 | Finalização do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso

 FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Alef Moura \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_TURMA: cc4p20\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_RA: F09ADC3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CURSO:Ciência da computação\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_CAMPUS:Norte\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:3°\_\_\_\_\_\_\_TURNO:Noturno\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ CÓDIGO DA ATIVIDADE:77B1\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SEMESTRE:Terceiro\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ANO GRADE:2020\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| DATA DA ATIVIDADE | DESCRIÇÃO DA ATIVIDADE | TOTAL DE HORAS | ASSINATURA DO ALUNO | HORAS ATRIBUÍDAS (1) | ASSINATURA DO PROFESSOR |
| 03/10/2020 | Reunião para alinhamento de equipe | 9 |  |  |  |
| 10/10/2020 | Busca de conteúdo para auxiliar no desenvolvimento | 11 |  |  |  |
| 17/10/2020 | Leitura de livros e aprimoramento de conteúdo específico para melhor desenvolvimento | 11 |  |  |  |
| 24/10/2020 | Inicio do desenvolvimento do algoritmo parte 1 de 2 | 12 |  |  |  |
| 31/10/2020 | Finalização do algoritmo parte 2 de 2 | 12 |  |  |  |
| 07/11/2020 | Inicio do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
| 14/11/2020 | Finalização do desenvolvimento em equipe dos relatórios e documentação exigida | 11 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso