

**`**

**Algoritmos e Estruturas de Dados – 2022/2023 1º Semestre**

Relatório de Projeto

**“ArtAuctions - Obras de Arte e Leilões”**

**Cadeira de Algoritmos e Estruturas de Dados, 2023/2024**

**Realizado por:**

**Joana Neves, 65441**

**Rui Xavier, 65815**

Índice

[1. Introdução 2](#_Toc152092589)

[2. Estrutura, hierarquia, e funcionamento do projeto 3](#_Toc152092590)

[2.1. Tipos Abstratos de Dados 4](#_Toc152092591)

[2.1.1. Interface AuctionManagement 4](#_Toc152092592)

[2.1.1.1. Estruturas de dados da classe 4](#_Toc152092593)

[2.1.2. Interface User 6](#_Toc152092594)

[2.1.2.1. Interface PrivateUser 6](#_Toc152092595)

[2.1.2.2. Interface Collector 6](#_Toc152092596)

[2.1.2.3. Interface Artist 6](#_Toc152092597)

[2.1.3. Interface Auction 7](#_Toc152092598)

[2.1.3.1. Estruturas de dados da classe 7](#_Toc152092599)

[2.1.4. Interface Bid 7](#_Toc152092600)

[2.1.5. Interface Artwork 8](#_Toc152092601)

[2.1.6. Interface ArtworkInAuction 8](#_Toc152092602)

[2.1.3.1. Estruturas de dados da classe 9](#_Toc152092603)

[3. Estudo da Complexidade Temporal 10](#_Toc152092604)

[3.1. Inserir utilizador 10](#_Toc152092605)

[3.2. Inserir artista 10](#_Toc152092606)

[3.3. Remover utilizador 11](#_Toc152092607)

[3.4. Inserir obra de arte 11](#_Toc152092608)

[3.5. Consultar dados do utilizador 12](#_Toc152092609)

[3.6. Consultar dados do artista 13](#_Toc152092610)

[3.7. Consultar dados da obra de arte 13](#_Toc152092611)

[3.8. Criar um leilão 14](#_Toc152092612)

[3.9. Inserir uma obra de arte num leilão 15](#_Toc152092613)

[3.10. Inserir uma bid (proposta para uma obra) 16](#_Toc152092614)

[3.11. Fechar um leilão 17](#_Toc152092615)

[3.12. Listar todas as obras de um leilão 18](#_Toc152092616)

[3.13. Listar todas as obras de um artista 19](#_Toc152092617)

[3.14. Listar todas as bids de uma obra de arte 20](#_Toc152092618)

[3.15. Listar todas as obras vendidas por preço 21](#_Toc152092619)

[3.16. Terminar a execução 21](#_Toc152092620)

[4. Estudo da Complexidade Espacial 22](#_Toc152092621)

[5. Conclusão 22](#_Toc152092622)

# Introdução

No âmbito da disciplina de Algoritmos e Estruturas de Dados do 2º ano de Licenciatura em Engenharia Informática, foi-nos proposta a realização do projeto **“ArtAuctions - Obras de Arte e Leilões”** de forma a aplicar os conhecimentos adquiridos ao longo do decorrer das aulas práticas e teóricas.

Este trabalho tinha como objetivo principal o desenvolvimento de um sistema de gerenciamento de obras de arte em leilões. O programa destina-se a ser usado por dois tipos de utilizadores: os **colecionadores** **(*Collectors*)** e os **artistas (A*rtists)***. Os colecionadores podem efetuar licitações de compra num determinado leilão para uma certa obra de arte (***Artwork***). Os artistas usam o sistema para introduzir obras de arte, introduzir o nome da obra, o ano da criação da mesma e o autor desta obra, neste caso ele mesmo.

O presente relatório destina-se à descrição detalhada de todo o processo de desenvolvimento do projeto, bem como à justificação de algumas das decisões tomadas relativamente à maximização da eficiência temporal e espacial do programa. Inclui ainda uma breve conclusão sobre a utilidade deste trabalho na melhor compreensão dos temas relevantes à realização do mesmo.

# Estrutura, hierarquia, e funcionamento do projeto

Abaixo encontra-se um diagrama de classes que visa dar um melhor entendimento da hierarquia das classes usadas no projeto, bem como da forma como estas se interligam. Como é possível interpretar pelo mesmo, todas as interfaces, à exceção da Interface ***AuctionManagement***, são estendidas por uma subinterface com o prefixo “***Private***”. Este procedimento tem como objetivo a melhor divisão dos métodos declarados pelas interfaces, evitando a partilha desnecessária de informação com o exterior das classes que as implementam. Os métodos “***set***”, ou seja, que alteram de alguma forma os valores da classe, estão declarados na interface privada e só podem ser acedidos pela classe de topo, ficando assim inacessíveis diretamente pela classe **Main**. Todos os restantes métodos estão declarados na classe pública e podem ser acedidos livremente.

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

## Tipos Abstratos de Dados

### Interface AuctionManagement

É implementada pela classe ***AuctionManagementClass***, que funciona como o sistema central do programa. Faz o tratamento dos dados e a comunicação entre a **Main** e as restantes classes. Define 3 estruturas de dados e 2 constantes *int*:

* + - 1. Dictionary<String, PrivateUser> users
      2. Dictionary<String, PrivateArtwork> artworks
      3. Dictionary<String, PrivateAuction> auctions

Definem dicionários implementados por uma classe ***SepChainHashTable***. São utilizados para guardar, respetivamente, todos os utilizadores, obras de arte e leilões registados no sistema.

* + - 1. OrderedDictionary<Integer, OrderedDictionary<String, PrivateArtwork>> artworksRanking

Define um dicionário ordenado implementado por uma classe ***AVLTree***(***AVL – Adelson-Velskii and Landis)***. É utilizado para guardar todas obras de arte no sistema pelo ranking de venda. Em que o ***value*** de cada **key** neste dicionário é uma também uma ***AVLTree*** mas ordenada por um comparador de forma a estar ordenada por ordem decrescente do valor de venda, e em caso de empate, por ordem lexicográfica.

* + - 1. final int MINIMUM\_AGE = 18

Define a idade mínima que um ***User*** deve ter para ser inserido no sistema.

#### Estruturas de dados da classe

Muitas das funcionalidades suportadas pelo programa requerem a inserção, remoção, pesquisa e modificação do estado de leilões, obras de arte e utilizadores várias vezes, não sendo necessário considerar a sua organização no sistema. Assim, achámos conveniente utilizar tabelas de dispersão para guardar os dados descritos acima nas alíneas **1.**, **2.**, **3.**, uma vez que a complexidade temporal da inserção, remoção, e pesquisa nesta estrutura é baixa e sempre constante (O(1)). Nestas usamos como ***key*** o login do utilizador ou identificador da obra de arte ou do leilão e como ***value*** uma instância da classe ***CollectorClass***, ***ArtistClass***, ***ArtworkClass***, ***AuctionClass***.

Sabemos ainda que outra funcionalidade do programa permite listar as artworks por ordem decrescente de acordo com o seu preço de venda mais alto, listando, assim o ranking das obras – ***listArtworksByValue()***. Para tal achámos conveniente utilizar uma Árvore de pesquisa Binária AVL como descrito acima na alínea **4.**.

Note-se, ainda, que é possível que exista mais de uma obra de arte com o mesmo preço de venda, e pretende-se que estas sejam então listadas por ordem decrescente dos respetivos preços de venda e em caso de empate, as obras são listadas por ordem lexicográfica do nome das respetivas obras. Para dar resposta a essa necessidade, esta estrutura tem uma particularidade: Cada nó da árvore (alínea **4.**) tem como ***key***,um Integer, e como ***value*** uma outra árvore de pesquisa binária cujos nós, por sua vez, contêm as obras de arte, com o mesmo preço de venda, nestas outras árvores as ***keys*** são os nomes das obras e os ***values*** são as respetivas obras. Estas estão ordenadas por ordem lexicográfica (caso de empate entre preços de venda).

Na última alínea está definida uma constante que correspondente à idade mínima de um ***User***. Serve apenas para facilitar a interpretação do código e, caso seja necessário, uma eventual mudança dos valores definidos sem ter de alterar todos os métodos que os utilizam.

### Interface User

É estendida pelas sub-interfaces ***PrivateUser***, ***Collector*** e ***Artist***.

#### Interface PrivateUser

É implementada pela classe abstrata ***AbstractUserClass***, que é a classe mãe das subclasses ***CollectorClass*** e ***ArtistClass***. Cada ***User***, seja ele ***Collector*** ou ***Artist***, é um utilizador individual registado no sistema. Este possui um identificador único, um nome, um email e a sua idade. Os 3 primeiros elementos são definidos por constantes *String* e o último por uma constante *int*:

1. final String login

Inicializada no construtor como o identificador único do utilizador registado;

1. final String name

Inicializada no construtor como o nome do utilizador registado;

1. final String email

Inicializada no construtor como o email do utilizador registado;

1. final int age

Inicializada no construtor como a idade do utilizador registado.

1. Protected int countBidsActive

Inicializada no construtor como o counter das bids ativas do utilizador registado;

* + - 1. final int ZERO = 0

Define o número zero e é utilizada para inicializar o countBidsActive e verificar se este é igual a zero ou não.

#### Interface Collector

É estendida pela interface ***PrivateCollector***, que por sua vez é implementada pela classe ***CollectorClass***. Cada ***Collector*** é um tipo específico de ***User***, porém não precisa de mais dados para além daqueles que foram definidos na sua superclasse.

#### Interface Artist

É estendida pela interface ***PrivateArtist***, que por sua vez é implementada pela classe ***ArtistClass***. Cada ***Artist*** é um tipo específico de ***User*** com características particulares. Para além dos dados definidos na sua superclasse, possui ainda um nome artístico. Este é definido por uma constante *String*:

1. final String artistName

Inicializada no construtor como o nome artístico do artista registado.

1. protected int hasArtworkInAuction

Inicializada no construtor como o counter do número de obras de arte que o artista registado tem em leilões (Auctions).

### Interface Auction

É estendida pela interface ***PrivateAuction***, que por sua vez é implementada pela classe ***AuctionClass***. Cada ***Auction*** representa um leilão registado no sistema e possui um identificador único, uma quantidade indefinida de obras de arte, definidos através de uma constante *String*, duas estruturas de dados.

* + - 1. final String id

Definida no construtor como o identificador único do leilão.

* + - 1. Dictionary<String, ArtworkInAuciont> artworkInAuctionDictionary

Define uma tabela de dispersão implementada por uma classe ***SepChainHashTable***. É utilizada para guardar todas as obras atualmente colocadas no leilão correspondente.

* + - 1. DoubleList<ArtworkInAuction> artworkInAuctionList

Define uma lista implementada por uma classe ***DoubleList***. É utilizada para guardar todas as obras atualmente colocadas no leilão correspondente, estando ordenada por ordem de inserção.

#### 2.1.3.1. Estruturas de dados da classe

Pretende-se que o programa nos permita listar as obras de arte que estão atualmente registadas num determinado leilão, por ordem cronológica de inserção. Como tal, optámos por usar uma ***DoubleList***, uma vez que esta estrutura nos permite organizar os dados pela ordem pretendida, isto através da artworkInAuctionList pois é mais rápido para uma iteração. Por outro lado, também guardamos todas as obras de arte no leilão, neste caso numa tabela de dispersão para facilitar a pesquisa e acesso a uma determinada obra de arte, sabendo o hash desta.

### Interface Bid

É estendida pela interface ***PrivateBid***, que por sua vez é implementada pela classe ***BidClass***. Cada ***Bid*** representa uma proposta registada numa certa obra de arte (artwork) e possui o valor da proposta feita, um user, o que realizou a proposta em causa e um leilão (auction), que corresponde ao leilão em que esta bid foi feita, definidos através de uma constante *int*, de um ***User*** e de uma ***Auction***, respetivamente.

1. final int value

Inicializada no construtor como o valor da proposta de venda da obra de arte registado.

1. final User bidder

Inicializada no construtor como o ***User*** que realizou esta proposta para a obra no sistema.

1. final Auction auction

Inicializada no construtor como a ***Auction*** em que foi realizada esta proposta para a obra no sistema.

### Interface Artwork

É estendida pela interface ***PrivateArtwork***, que por sua vez é implementada pela classe ***ArtworkClass***. Cada ***Artwork*** representa uma obra de arte e possui um identificador único, o ano em que foi feita, um nome, um ***User*** que a criou e um valor de venda mais alto. Estes dados estão definidos através de uma constante *String*, de 2 Strings, de um ***User*** e de um *int*, respetivamente.

1. final String id

Definida no construtor como o identificador único da obra de arte.

1. protected String year

Inicializada no construtor como o ano em que foi criada a obra de arte.

1. protected String name

Inicializada no construtor como o nome da obra de arte.

1. Protected User author

Inicializada no construtor como o ***User*** que criou a obra de arte.

1. protected int highestPrieValueSold

Inicializada a zero no construtor como o valor mais alto pelo qual a obra de arte foi vendida entre todos os leilões existentes.

### Interface ArtworkInAuction

É estendida pela interface ***PrivateArtworkInAuction***, que por sua vez é implementada pela classe ***ArtworkInAuctionClass***. Cada ***ArtworkInAuction*** representa uma obra de arte que se encontra num leilão e possui uma obra de arte, um preço mínimo para as licitações, uma licitação que é a mais alta já feita à obra de arte nesse leilão, uma variável de estado para saber se a mesma já foi vendida e uma quantidade indefinida de licitações. Estes dados estão definidos através de um ***Artwork***, um int, umabid, um boolean e uma estrutura de dados, respetivamente.

1. final PrivateArtwork artwork

Definida no construtor como a obra de arte que se encontra num leilão.

1. final int minimumPrice

Inicializada no construtor como o preço mínimo pelo qual a obra de arte pode ser vendida neste leilão.

1. protected Bid highestBid

Inicializada no construtor como a bid mais alta relativa à obra de arte no respetivo leilão.

1. Protected boolean sold

Inicializada a falso no construtor como o *boolean* que informa se a obra de arte já foi vendia neste leilão.

1. DoubleList<Bid> bids

Define uma lista implementada por uma classe ***DoubleList***. É utilizada para guardar todas as ***Bids*** atualmente realizadas à obra de arte no leilão correspondente, estando ordenada por ordem de inserção.

#### 2.1.3.1. Estruturas de dados da classe

Pretende-se que o programa nos permita listar as ***Bids*** relativas a esta obra de arte que estão atualmente registadas num determinado leilão, por ordem cronológica de inserção. Como tal, optámos por usar uma ***DoubleList***, uma vez que esta estrutura nos permite organizar os dados pela ordem pretendida e pela sua iteração ser mais rápida.

# Estudo da Complexidade Temporal

## Inserir utilizador

Este comando insere um novo colecionador no sistema desde que não exista outro usuário com o mesmo login. Para fazer esta verificação fazemos um find no dicionário ***users*** para averiguar a existência de um ***User*** com o mesmo login, caso não se verifique criamos um novo objeto com os dados fornecidos pela ***Main*** e inserimo-lo no dicionário.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
| Criação | O(1) | O(1) | O(1) |
| Inserção | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda :

* + - n - Número total de utilizadores no dicionário ***users***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Inserir artista

Este comando insere um novo artista no sistema desde que não exista outro usuário com o mesmo login. Para fazer esta verificação fazemos um find no dicionário ***users*** para averiguar a existência de um ***User*** com o mesmo login, caso não se verifique criamos um novo objeto com os dados fornecidos pela ***Main*** e inserimo-lo no dicionário.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
| Criação | O(1) | O(1) | O(1) |
| Inserção | O(1) | O(1 + ℷ) | O(n) |
|  |  |  |  |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda :

* + - n - Número total de utilizadores no dicionário ***users***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Remover utilizador

Este comando remove um utilizador do sistema caso este exista no sistema. Para verificar isto fazemos um find no dicionário ***users*** para averiguar a sua existência. Caso este não exista é lançada uma exceção. Caso este exista vamos verificar se o mesmo tem propostas ativas, chamando um boolean, e se o mesmo for uma artista vamos verificar se tem obras em leilão, chamando um outro boolean. Em qualquer um destes casos, se o utilizador tiver propostas ativas ou obras em leilão, é lançada uma exceção. Caso nenhuma exceção seja lançada, caso o utilizador seja um artista, é ainda necessário remover as suas obras do sistema, iterando pelas obras do utilizador e removendo-as do dicionário ***artworks***.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa do utilizador | O(1) | O(1 + ℷ) | O(n1) |
| Criação do iterador das obras do artista | O(1) | O(1) | O(1) |
| Iteração das obras do artista | O(1) | O(log(n2)\*n2) = a | O((1.44)\*log(n2)\*n2) = b |
| Remoção do utilizador | O(1) | O(1 + ℷ) | O(n1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(a)** | **O(b)** |

Legenda:

* + - n1 - Número total de utilizadores no dicionário ***users***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n2 – número de obras de arte que o artista tem.

## Inserir obra de arte

Este comando insere uma nova obra de arte no sistema desde que não exista outra obra de arte com o mesmo identificador. Para fazer esta verificação fazemos um find no dicionário ***artworks*** para averiguar a existência de uma ***Artwork*** com o mesmo identificador, caso não se verifique criamos um novo objeto com os dados fornecidos pela ***Main*** e inserimo-lo no dicionário.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
| Criação | O(1) | O(1) | O(1) |
| Inserção | O(1) | O(1 + ℷ) | O(n) |
|  |  |  |  |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de utilizadores no dicionário ***users***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Consultar dados do utilizador

Este comando lista as informações de um utilizador genérico. Começamos por verificar se o utilizador existe no sistema, fazendo um find no dicionário ***users***, se o utilizador existir devolvemo-lo, caso contrário é lançada uma exceção.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
|  |  |  |  |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de utilizadores no dicionário ***users***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Consultar dados do artista

Este comando lista as informações de um utilizador do tipo ***Artist***. Começamos por verificar se o utilizador existe no sistema, fazendo um find no dicionário ***users***, se o utilizador existir e for um artista devolvemo-lo, caso contrário é lançada uma exceção.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de utilizadores no dicionário ***users***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Consultar dados da obra de arte

Este comando lista as informações de uma obra de arte. Começamos por verificar se a obra de arte existe no sistema, fazendo um find no dicionário ***artworks***, se a obra existir devolvemo-la, caso contrário é lançada uma exceção.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de obras de arte no dicionário ***artworks***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Criar um leilão

Este comando adiciona um novo leilão ao sistema desde que não exista outro leilão com o mesmo identificador. Para fazer esta verificação fazemos um find no dicionário ***auctions*** para averiguar a existência de um ***Auction*** com o mesmo identificador, caso não se verifique criamos um novo objeto com os dados fornecidos pela ***Main*** e inserimo-lo no dicionário.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa | O(1) | O(1 + ℷ) | O(n) |
| Criação | O(1) | O(1) | O(1) |
| Inserção | O(1) | O(1 + ℷ) | O(n) |
|  |  |  |  |
| **Total** | **O(1)** | **O(1 +** ℷ**)** | **O(n)** |

Legenda:

* + - n - Número total de leilões no dicionário ***auctions***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.

## Inserir uma obra de arte num leilão

Este comando permite inserir uma obra de arte num leilão. Para que este comando tenha sucesso temos de verificar:

- se o leilão existe  
 - se a obra de arte existe

Para tal, fazemos uma pesquisa no dicionário ***auctions*** e uma pesquisa no dicionário ***artworks***. Se as condições anteriores estiverem reunidas, o comando prossegue com a sua execução e adiciona a obra ao leilão. Para adicionar a obra ao leilão, dentro da ***AuctionClass*** é feita uma inserção no dicionário ***artworksInAuctionDictionary*** e no final da lista ***artworksInAuctionList***.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pelo leilão | O(1) | O(1 + ℷ) | O(n) |
| Pesquisa pela obra | O(1) | O(1 + ℷ) | O(n1) |
| Inserção da obra no dicionário do leilão | O(1) | O(1 + ℷ) | O(n2) |
| Inserção da obra no final da lista do leilão | O(1) | O(1) | O(1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1 +** ℷ) | **O(max(n, n1, n2))** |

Legenda:

* + - n - Número total de leilões no dicionário ***auctions***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n1 - Número total de obras no dicionário ***artworks***. (número de entradas da tabela)
    - n2 - Número de obras no dicionário ***artworksInAuctionDictionary***.

## Inserir uma bid (proposta para uma obra)

Este comando adiciona uma proposta feita por um utilizador a uma obra de arte num leilão. Para que este comando tenha sucesso temos de verificar:  
 - se o utilizador existe  
 - se o leilão existe  
 - se a obra existe no leilão  
 - se o valor da proposta é superior ao valor mínimo definido quando a obra foi adicionada ao leilão

Para tal, fazemos uma pesquisa no dicionário ***users***, uma pesquisa no dicionário ***auctions***, uma pesquisa no dicionário ***artworks*** e uma pesquisa no dicionário ***artworksInAuctionDictionary*** do auction que foi encontrado anteriormente. Se todas as condições anteriores se reunirem, o comando prossegue como é suposto e adiciona a proposta ao utilizador que a fez (aumentando o contador de propostas do mesmo) e à obra de arte no leilão (adicionando a proposta ao final da lista de propostas da obra).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pelo utilizador | O(1) | O(1 + ℷ) | O(n) = a |
| Pesquisa pelo leilão | O(1) | O(1 + ℷ) | O(n1) = b |
| Pesquisa pela obra | O(1) | O(1 + ℷ) | O(n2) = c |
| Pesquisa pela obra no leilão | O(1) | O(1 + ℷ) | O(n3) = d |
| Comparação do valor da proposta | O(1) | O(1) | O(1) |
| Inserção da proposta no utilizador | O(1) | O(1) | O(1) |
| Inserção da proposta na obra de arte | O(1) | O(1) | O(1) |
| **Total** | **O(1)** | **O(1 + ℷ)** | **O(max(a,b,c,d))** |

Legenda:

* + - n - Número total de utilizadores no dicionário ***users***. (número de entradas da tabela)
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n1 - número de leilões no dicionário ***auctions***.
    - n2 – número de obras no dicionário ***artworks***.
    - n3 – número de obras no dicionário ***artworksInAuctionDictionary.***

## Fechar um leilão

Este comando fecha um leilão e lista todas as obras que estavam dentro dele, devolvendo um iterador de obras para a Main. Para o fazer temos que:

* verificar se o leilão existe

Para tal é feita uma pesquisa no dicionário de leilões (auctions). Caso a condição anterior se verifique o comando prossegue como é suposto. Para fechar o leilão iteramos por todas as obras que estão dentro dele, atualizando, caso necessário, o valor de venda mais alto de cada obra existente e atualizando ou adicionando ao ranking em geral das obras vendidas. Para além disto, decrementamos o número de obras em leilões de cada artista, que tenha uma obra no respetivo leilão.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Remoção do leilão | O(1) | O(1 + ℷ) | O(n1) |
| Criação do iterador das obras de arte no leilão | O(1) | O(1) | O(1) |
| Iteração das obras | O(n2\*log(n3)) = a | O(n2\*log(n3)) = b | O(n2\*(1.44)\*log(n3)) = c |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(a)** | **O(b)** | **O(c)** |

Legenda:

* + - n1 - Número total de leilões.
    - ℷ - Fator de ocupação do dicionário, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n2 - número de obras de arte no leilão.
    - n3 – número de obras já vendidas.

## Listar todas as obras de um leilão

Este comando lista todas as obras existentes num leilão por ordem de inserção no sistema. Para o comando ter sucesso tem existir tanto o leilão (auction) como este tem que ter obras de arte inseridas.

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | Melhor Caso | Caso Esperado | Pior Caso. |
| Pesquisa do leilão | O(1) | O(1) | O(n1) |
| Criação do iterador | O(1) | O(1) | O(1) |
| Iteração de obras | O(n2) | O(n2) | O(n2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n2)** | **O(n2)** | **O(max(n1, n2))** |

Legenda:

* + - n1 - Número total de leilões existentes.
    - nremo2 – número total de obras inseridas no respetivo leilão.

## Listar todas as obras de um artista

Este comando lista todas as obras de arte de um artista. Para tal acontecer tanto o *user* tem de existir, tal como tem de ser um ***Artist***, como o artista tem que ter obras de arte.Para chegarmos à conclusão que podemos iterar as obras temos que realizar uma busca tanto na estrutura de dados que guarda todos os *users* como verificar se o user caso encontrado é um artista ou não. Caso seja um artista verifica-se se tem obras de arte.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pelo utilizador | O(1) | O(1 + ℷ) | O(n1) |
| Criação do iterador | O(1) | O(log(n2)) | O(n2) |
| Iteração de obras | O(n2) | O(n2) | O(n2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n2)** | **O(n2)** | **O(n2)** |

Legenda:

* + - n1 - Número total de utilizadores.
    - ℷ - Fator de ocupação do dicionário ***users***, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n2 - número de obras que o artista tem.

## Listar todas as bids de uma obra de arte

Este comando lista todas as propostas que uma obra de arte tem num determinado leilão. Para tal acontecer tanto o leilão como a obra de arte têm que existir no sistema, para além que a obra de arte tem que ter propostas, no leilão. Para chegarmos à conclusão que podemos iterar as propostas temos que realizar uma busca tanto na estrutura de dados que guarda todos os leilões (auctions) como na das obras de artes existentes no leilão (artworksInAuction), caso este exista, e caso a obra de arte existe pergunta-se à obra de arte se tem propostas ou não.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Pesquisa pelo leilão | O(1) | O(1 + ℷ1) | O(n1) |
| Pesquisa pela obra | O(1) | O(1 + ℷ2) | O(n2) |
| Criação do iterador | O(1) | O(1) | O(1) |
| Iteração de bids | O(n3) | O(n3) | O(n3) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(n3)** | **O(n3)** | **O(max(n1, n2, n3))** |

Legenda:

* + - n1 - número total de leilões existentes.
    - ℷ1 - Fator de ocupação do dicionário ***auctions***, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n2 - número total de obras de arte existentes.
    - ℷ2- Fator de ocupação do dicionário ***artworks***, dado pela divisão do número de entradas da tabela pela dimensão da tabela.
    - n3 – número total de propostas que a obra de arte tem no leilão.

## Listar todas as obras vendidas por preço

Este comando lista as obras e arte vendidas por ordem descendente de preço mais alto de venda, e em caso de empates o critério de desempate é por ordem lexicográfica dos nomes das obras. Para tal acontecer tem de existir obras de artes vendidas, ou seja, o ***artworksRanking*** não pode estar vazio.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |
| Verificação de existem obras vendidas | O(1) | O(1) | O(1) |
| Criação do iterador | O(h) | O(h) | O(h) |
| Iteração de obras | O(n1) | O(n1) | O(n1) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(max(h, n1))** | **O(max(h, n1))** | **O(max(h, n1))** |

Legenda:

* + - h – altura da árvore em que as chaves são *Integer*.
    - n1 – número de obras que já foram vendidas.

## Terminar a execução

Este comando termina a execução do programa.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operação** | **Melhor Caso** | **Caso Esperado** | **Pior Caso.** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Total** | **O(1)** | **O(1)** | **O(1)** |

# Estudo da Complexidade Espacial

A complexidade espacial é a soma do tamanho ocupado por todas as estruturas de dados:

**O(t1) + O(t2) + O(t3) + O(t4)**

Legenda:

* t1 - tamanho da ***SepChainHashTable*** “users”.
* t2 - tamanho da ***SepChainHashTable*** “artworks”.
* t3 - tamanho da ***SepChainHashTable*** “auctions”.
* t4 - tamanho da ***AVLTreeWithComparator<Integer, AVLTree>*** “artworksRanking”.

# Conclusão

Tendo em conta a ideia base da disciplina - a aprendizagem das formas de utilização e funcionamento das várias estruturas de dados - sabíamos que o objetivo principal deste trabalho seria exatamente isso: fazer-nos pensar e refletir sobre a melhor forma de implementar as diferentes funcionalidades do programa, tendo sempre em mente a eficiência do código e poupança de tempo e recursos. Acreditamos que este objetivo foi alcançado na totalidade, e que somos agora capazes de desenvolver programas com melhor eficiência de tempo e memória.