CPS- Faculdade de Tecnologia da Zona Leste
Tecnologia em Análise e Desenvolvimento de Sistemas

KAIO VENÂNCIUS GALVÃO SANTOS SILVA - 1110481813005
LUAN FERNANDES – 1110481622017
LUCCAS DI CATERINA - 1110481622018
LUCIANO ANDRÉ DOS SANTOS - F1614479

Projeto de Desenvolvimento de Sistema

Desenvolvimento de sistema para automatização e melhoria de processos do CPS voltados Inscrição de Processo Seletivo.

São Paulo – São Paulo
2020

KAIO VENÂNCIUS GALVÃO SANTOS SILVA - 1110481813005
LUAN FERNANDES – 1110481622017
LUCCAS DI CATERINA - 1110481622018
LUCIANO ANDRÉ DOS SANTOS - F1614479

Projeto de Desenvolvimento de Sistema

Desenvolvimento de sistema para automatização e melhoria de processos do CPS voltados Inscrição de Processo Seletivo.

Monografia voltada a Estrutura de Dados para obter a compreensão dos Conceitos apresentados na disciplina como: estruturas de dados não-lineares, armazenamento de dados em matrizes e listas de adjacências, tabela hash, métodos de ordenação e algoritmos de buscas, apresentado a Faculdade de Tecnologia da Zona Leste, como parte das exigências para cumprimento da disciplina de Estrutura de Dados no curso Tecnólogo em Análise e Desenvolvimento de Sistemas.

São Paulo, 03 de dezembro de 2020.

Professora Doutoranda Eliane Oliveira Santiago.

_____

Professora Doutoranda Eliane Oliveira Santiago.

Dedicatória

Dedico esse trabalho a todos os profissionais de TI que auxiliaram na construção de conhecimento presente nesse trabalho.

Dedico este trabalho aos meus pais falecidos, a quem agradeço as bases que deram para me tornar a pessoa que sou hoje.

A minha Professora de Estrutura de Dados – Eliane O. Santiago – que auxiliou no caminho de nossa formação como profissionais de TI por meio dos desafios feitos ao longo do semestre voltados a Estrutura de Dados.

Todo conhecimento fornecido e obtido deste trabalho é creditado aos alunos que compõem o grupo, aos professores que nos guiaram nesse aprendizado e a todas as referências ao fim descritas.

Sumário

Dedicatória 4

1.Introdução 1

1.1.Contexto 1

1.2.Problema 1

Definição do sistema 1

Histórico das Inscrições 2

Futuro das Inscrições 3

1.3.Objetivo 3

2.Referencial Teórico 3

2.1.Políticas FIFO e LIFO 4

2.2.Algoritmos de Ordenação 5

2.3. Justificativa 8

2.5. Comparação 9

3.Projeto Digitaliza FATEC 11

3.1.Técnicas 11

3.2. Algoritmo implementado 13

3.3.Boas práticas 16

4. Considerações finais 16

5.Referências 18

1. Introdução

1.1. Contexto

Dada a necessidade de informatizar os processos de inscrição para processos seletivos no CPS, se faz a necessário encontrar soluções viáveis e que atendam o negócio. Devido a burocracia e complexidade que convivemos no mundo dos negócios, passamos a utilizar da tecnologia como um facilitador para nossas práticas diárias.

1.2. Problema

Desenvolver um sistema baseado no projeto desenvolvido na disciplina de Engenharia de Software I, utilizando os conceitos da disciplina de Estrutura de Dados e Engenharia de Software II em grupos de no máximo 4 pessoas.

Projeto de Desenvolvimento de Sistema de Inscrição Processo Seletivo.

Definição do sistema

Os atos administrativos são divulgados oficialmente para amplo conhecimento por meio de um edital. O processo seletivo para o curso de Pós-Graduação (GETI) é divulgado semestralmente por um edital, cuja elaboração é responsabilidade da Comissão do Processo Seletivo (CPS) para o GETI. Este edital é composto por definições do curso, do público alvo, período de inscrição, quantitativo de vagas por tipo: ampla concorrência, ações afirmativas e candidatos com deficiência. Este quantitativo não é fixo e pode ser alterado a cada edital. Os documentos exigidos podem ser alterados para cada campus, pois fica a cargo deles construírem o edital.

Cada edital especifica um cronograma de atividades composta com datas definidas previamente. Segue um exemplo do processo seletivo para o primeiro semestre de 2020, conforme o Quadro 1.

Quadro 1 - Cronograma de atividades

Atividade	Data
Inscrições para o processo seletivo	03/10/2019 a 27/10/2019
Divulgação das inscrições deferidas	30/10/2019
Solicitação de recursos sobre as inscrições	31/10/2019 a 01/11/2019
Divulgação das inscrições homologadas	04/11/2019
Processo seletivo Etapa 1 – Análise do Currículo	04/11/2019 a 08/11/2019
Divulgação do resultado da Etapa 1 – Análise do Currículo	11/11/2019
Divulgação do cronograma das entrevistas	13/11/2019
Processo seletivo Etapa 2 – Entrevista	18/11/2019 a 22/11/2019
Divulgação de resultado preliminar	02/12/2019
Solicitação de recursos da nota final	09/12/2019
Divulgação da classificação final	16/12/2019
Matrícula	08/01/2020 a 17/01/2020
Segunda chamada para vagas remanescentes	20/01/2020 a 24/01/2020
Início das Aulas	17/02/2020

O processo seletivo conta com duas etapas: análise do currículo e entrevista. A análise de currículo é realizada após a homologação das inscrições. A análise da inscrição é realizada pela CPS ou mesmo pela Coordenação de Registro Acadêmico (CRA) verificando se os documentos entregues são válidos. Não são exigidos documentos originais ou cópias autenticadas, apenas cópias simples. Após a constatação que os documentos são válidos, é publicada a divulgação prévia do deferimento das inscrições. Caso um candidato tenha sua inscrição indeferida, ele pode solicitar um recurso para reavaliação do seu processo. A etapa de análise de currículo e entrevista também permitem a solicitação de recursos.

Histórico das Inscrições

Inicialmente, os candidatos se inscreviam na secretaria, entregando a documentação física exigida pelo edital. Em processo continuo de melhoria, as inscrições do segundo semestre de 2019 tornaram-se digitais, com entrega de

documentos via e-mail para a CRA. Porém, o processo interno não teve a mesma evolução que desejamos.

Futuro das Inscrições

A CPS necessita de um sistema que possa automatizar todo o processo, com a inscrição, publicação, análise dos documentos e currículo digital, melhorando o processo totalmente. Foram elaborados os requisitos funcionais iniciais para a construção do sistema.

Um desejo futuro é a publicação por meio de envio de e-mail, em que as pessoas possam se cadastrar para terem notícias sobre do processo seletivo.

1.3.Objetivo

Temos como objetivo compreender os conceitos presentes em estrutura de dados como: as políticas de acesso aos dados, bem como a conclusão sobre o melhor algoritmo de classificação de dados. Ainda sim, poder aplicar tais conceitos e desenvolver uma solução que vise automatizar todo o processo de inscrição, publicação, análise dos documentos e currículo digital a fim de digitalizar os processos que o CPS tem durante os processos seletivos.

1.4.Motivação

Motivação: o desafio se tornou uma experiência de aprendizado e aperfeiçoamento de nossos conhecimentos voltados tanto as áreas de lógica, desenvolvimento e estrutura de dados presentes no projeto.

Também temos como motivação - esse desafio que exprime a ideia de expor uma solução para um problema real enfrentado por diversas universidades públicas.

2.Referencial Teórico

Diante das necessidades de estruturar, ordenar e classificar os dados estruturados e não-estruturados no sistema, devemos deixar claro os conceitos que são usados no projeto.

2.1.Políticas FIFO e LIFO

Política FIFO – do inglês que significa “First In, First Out”, refere-se ao primeiro que entra na fila é o primeiro a sair dela.

No Brasil esse termo é também conhecido como PEPS, que significa o “Primeiro que Entra é o Primeiro que Sai”.

Segundo o site www.escolaedti.com.br “...o sistema traz grandes vantagens para o controle de estoque e a rotação de produtos, então, é indicado para diversas ocasiões, pois seu bom uso é capaz de trazer benefícios para a empresa”, tendo isso em mente podemos notar que o FIFO pode ser aplicado em diversos segmentos que necessitam de um controle de entrada e saída, priorizando o primeiro que entrou.

Para ilustrar melhor essa política realizamos a ilustração abaixo:

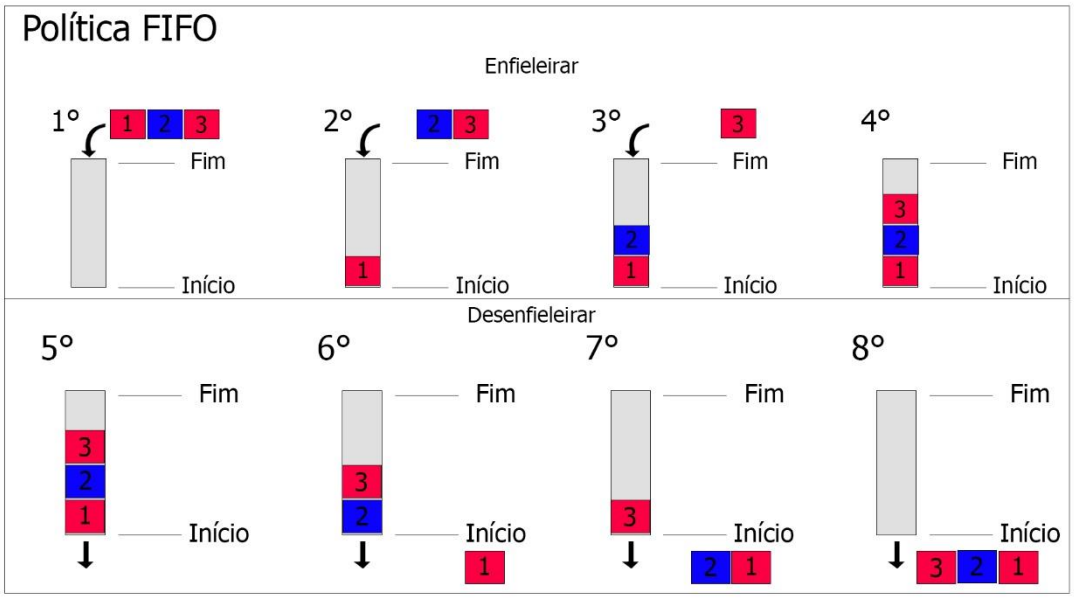

Figura 1 - Política FIFO. Autor: Kaio V. G. S. Silva

Política LIFO

Política LIFO – do inglês que significa “Last In, First Out”, refere-se ao último que entra na fila é o primeiro a sair dela.

No Brasil esse termo é também conhecido como UEPS, que significa o “Ultimo que Entra é o Primeiro que Sai”.

De acordo com o site www.saudebusiness.com “geralmente é utilizado para insumos sem prazo validade e com baixo volume de giro em estoque. Ainda é

usado para garantir uma margem de segurança de produtos em estoque para períodos de pico de giro.” Pode ser usado ao definir os valores de mercadoria, permitindo supervalorização do estoque de seus produtos.
Para ilustrar melhor essa política realizamos a ilustração abaixo:

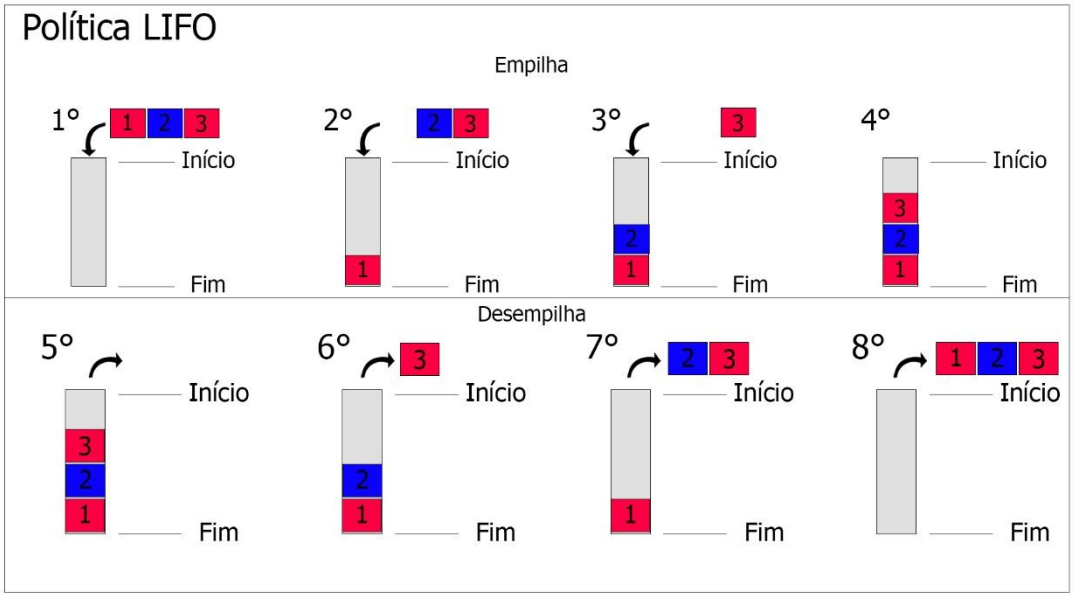

Figura 2 - Política LIFO. Autor: Kaio V. G. S. Silva

2.2.Algoritmos de Ordenação

Existem diversos algoritmos de ordenação, para realizar tal tarefa, porém alguns se destacam dos demais, seja no tempo de execução ou performance.
Dentre os diversos algoritmos, utilizamos QuickSort e InsertionSort.

QuickSort – O algoritmo de ordenação QuickSort é um algoritmo que tem como finalidade ordenar do menor para o maior, definindo um ou mais pivô para a comparação e ordenação entre os elementos. Após definir um pivô, o algoritmo ordena os elementos menores a esquerda do pivô e os maiores a sua direita. A fim de entregar performance, essa solução incrementada de recursividade permite a ordenação em diversas partes do vetor.

“O algoritmo é considerado instável, pois há a possibilidade de elementos com mesma chave mudar de posição no processo de ordenação.” Prof. Jairo

Francisco de Souza da UFJF (Universidade Federal de Juiz de Fora) Ordenação-QuickSort - 2009.

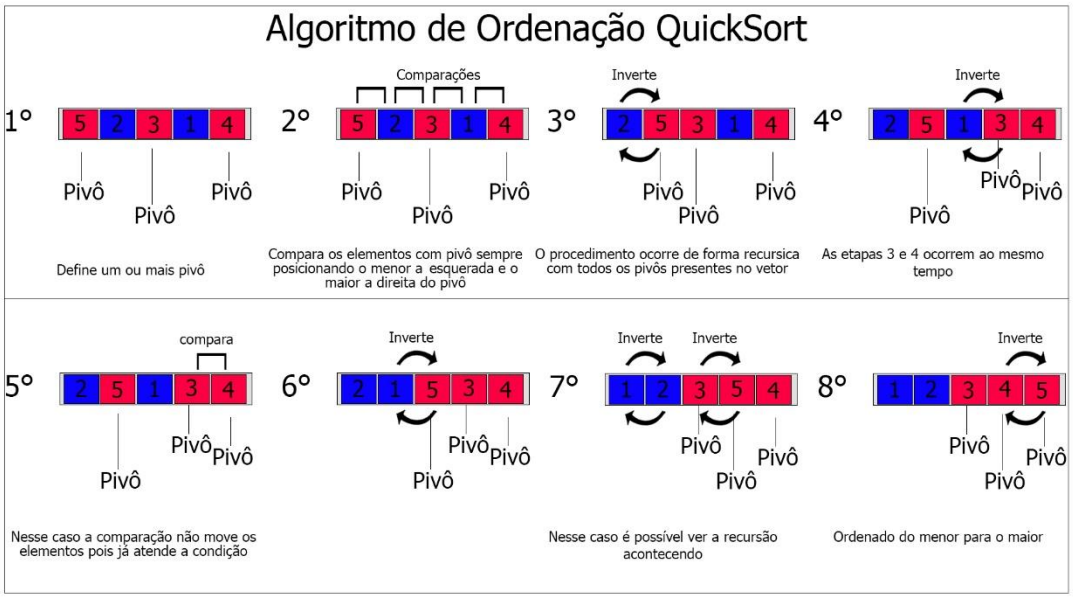

Figura 3 - QuickSort. Autor: Kaio V. G. S. Silva

Esse algoritmo é considerado o mais eficiente (para diversas situações) dentre os algoritmos de ordenação, onde seu pior caso a complexidade é n^2 (quadrática), seu caso médio e melhor caso a complexidade é nlogn (logarítmica).

InsertionSort – O algoritmo InsertionSort também permite a ordenação de um vetor comparando um elemento de cada vez, semelhante ao BubbleSorte, porém com maior eficiência. Segundo Ítalo Cunha – Professor de Estrutura de Dados da USP caso o vetor já esteja ordenado, o InsertionSort é o método mais rápido para conferência dessa ordenação.

Sua complexidade no pior caso e no caso médio é quadrática, ou seja, n^2 para situações como um vetor totalmente desordenado ou parcialmente desordenado. Já para um vetor ordenado o algoritmo InsertionSort possui complexidade linear N cujo a verificação dos elementos já ordenados oferece melhor performance.

Para realizar a ordenação de um vetor desordenado, começamos da segunda posição do vetor, é preciso definir uma variável para a posição atual do vetor, e duas outras variáveis, uma com o intuito de percorrer o vetor ao comparar (n -1 onde n é a posição atual) as posições, e outro que é o auxiliar para armazenar o conteúdo do vetor que será comparado com as posições anteriores. Por exemplo

um vetor de 3 posições (V[0,1,2] [4,7,5], lembrando que a variável ‘atual’ começa com 1, o conteúdo da segunda posição do vetor será atribuído para a variável ‘auxiliar’ (auxiliar = v[atual]) que é comparada com o conteúdo do vetor anterior (auxiliar > v[atual -1]), nesse caso o conteúdo da segunda posição ‘7’ é maior que o da primeira ‘4’, logo permanecerá na mesma posição. Ao final desse primeiro loop o valor do auxiliar é zerado. No segundo loop (a variável dentro de um for é incrementada), a variável atual recebe + 1, ou seja, nesse caso está apontando para a posição do vetor v[2], que repetirá o mesmo procedimento explicado anteriormente, nesse caso segundo loop a comparação entre os conteúdos retorna que ‘5’ não é maior do que ‘7’, então deve ser feita a troca da seguinte forma: a atribuição do conteúdo do v[atual -1] para o v[atual] (v[2]= v[1]) e a atribuição da variável auxiliar ao vetor[1], como no início do segundo loop o auxiliar recebeu o v[atual], ou seja. V[2] então tem o ‘5’ armazenado temporariamente para tal troca. Antes que seja trocado, é verificado com o conteúdo anterior ao v[atual-1], caso seja falso, é verificado com as demais posições, como foi positivo é realizado a troca. Ao final, o último elemento é comparado com os demais. Dessa forma podemos ordenar um vetor de forma eficiente. Abaixo apresentamos de forma ilustrativa o algoritmo InsertionSort.

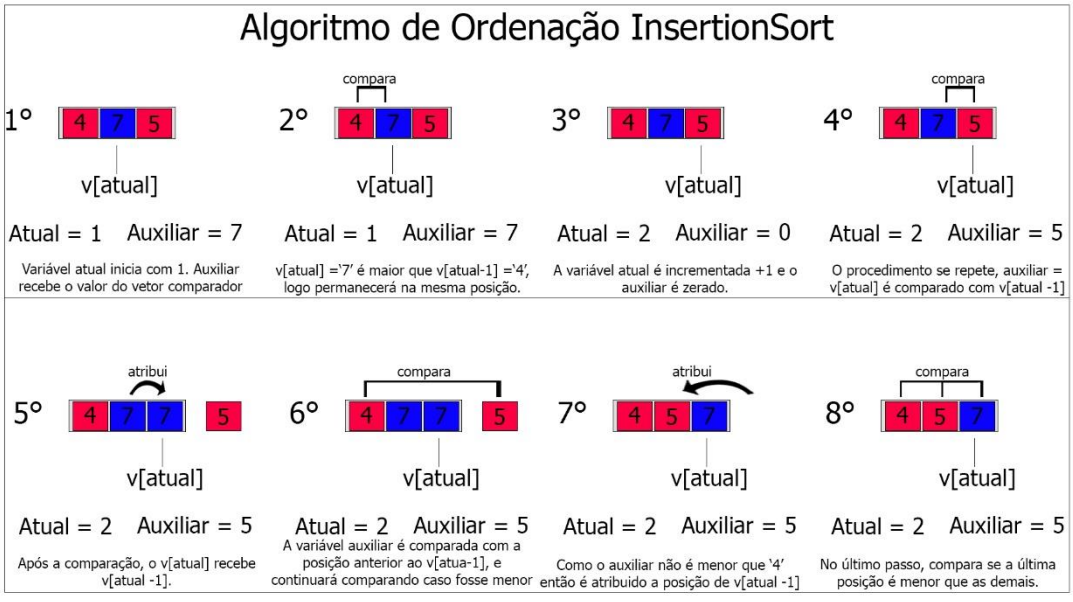

Figura 4 - InsertionSort. Autor: Kaio V. G. S. Silva

2.3. Justificativa

Utilizamos do algoritmo de ordenação QuickSort pois é o melhor algoritmo de ordenação em diversas situações: caso o vetor esteja totalmente aleatório, caso esteja semi-ordenado, caso o vetor seja muito grande, realizamos estudos em cima das nossas referencias e replicamos os testes de desempenho, em média o QuickSort entregou o resultado no menor tempo possível, exceto quando o vetor já estava ordenado, foi então que demos o mérito para o InsertionSort que teve melhor performance nesse quesito, independentemente do tamanho do vetor foi capaz de conferir se estava ordenado no menor tempo possível.

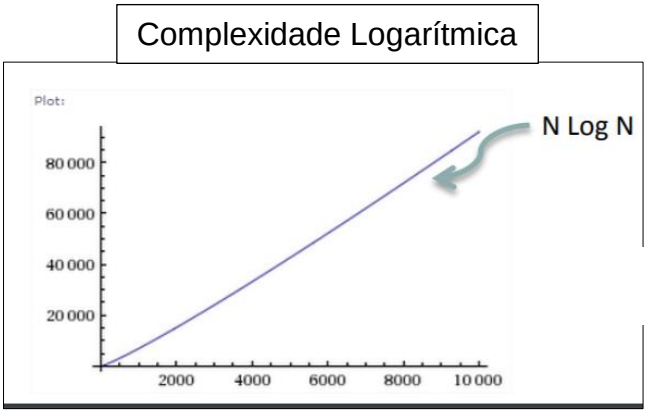

Figura 5 - material de aulas do Prof. Demétrios Coutinho – IFRN

A figura 5 apresenta a complexidade de um algoritmo QuickSort no caso médio e no melhor caso, se comparado com os demais algoritmos apresenta o resultado no menor tempo e com menos comparações possíveis. Já o pior caso é representado pelo gráfico abaixo, onde a complexidade é dada por n^2.

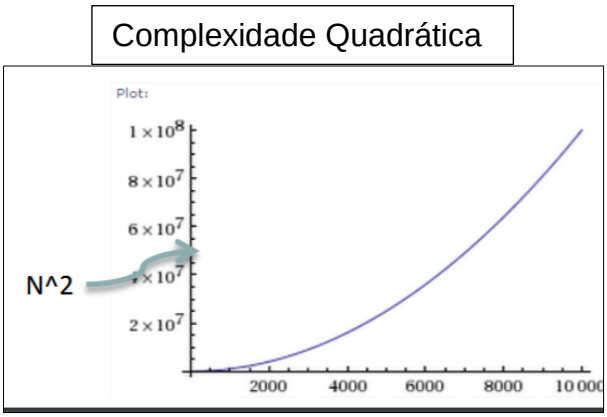

Figura 6 - material de aulas do Prof. Demétrios Coutinho - IFRN

O gráfico a direita também representa a complexidade no pior caso e no caso médio do InsertionSort que tem maior utilidade ao comparar as posições já ordenadas pois em seu melhor caso a complexidade é linear.

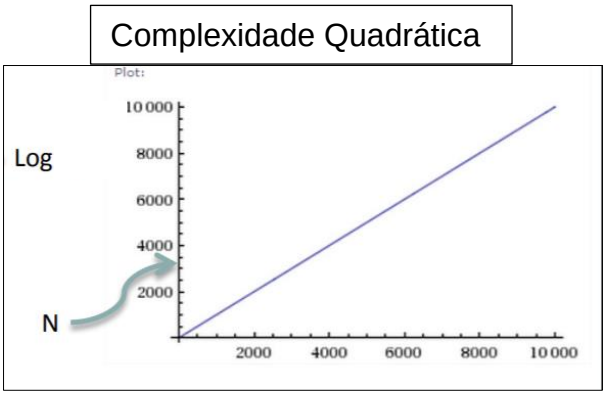

Figura 7 - material de aulas do Prof. Demétrios Coutinho - IFRN

Podemos afirmar que os demais algoritmos, possuem não apenas a complexidade, mas o tempo de resposta acima da média a qual situamos o QuickSort. Deixo claro que cada algoritmo pode ser utilizado de melhor forma para cada situação, mas em um contexto geral o algoritmo QuickSort acaba sendo a melhor escolha.

2.5. Comparação

Para ilustrar melhor a eficiência entre o QuickSort e os demais algoritmos, anexamos os testes feitos em uma máquina (para evitar qualquer vício no resultado – optamos por usar o método caixa de areia / SandBox para isolar o programa em uma virtualização que não sofra influência dos programas e drivers presentes na máquina física) nas mesmas situações. Utilizamos do Eclipse IDE, com código que realiza o cálculo de variação de tempo em **milissegundos** (ms) enquanto chama o algoritmo de ordenação contido em outra classe.
Como em um sistema normalmente as informações são inseridas de forma aleatória, realizamos os testes para essa situação, onde o QuickSort teve melhor resultado.

	A	B	C	D	E	F	G
1	Teste de Algoritmos de Ordenação Aleatória						
2	Tamanho do Vetor	BubbleSort	HeapSort	InsertionSort	MergeSort	QuickSort	SelectionSort
3	10	0	0	0	0	0	0
4	100	0	0	0	0	0	0
5	1.000	16	16	16	15	0	15
6	10.000	219	16	63	16	15	62
7	100.000	32406	46	5739	47	46	4968
8	Unidade de medida utlizada ms.						

Figura 8 - Testes de Algoritmos de ordenação tabelado no Excel

Obs.: os testes foram realizados 3 vezes, obtivemos 3 resultados que respectivamente buscamos a média ((a + b + c)/3) para ter um equilíbrio entre precisão e tempo de resposta do computador utilizado.

Como podemos ver na figura 7 o algoritmo HeapSort também tem um potencial refinado se comparado com os demais algoritmos. Geralmente utilizado em sistemas que precisam de uma precisam maior em seu tempo de resposta como por exemplo: gráficos, conversão de uma moeda ou até mesmo venda e compra de ações.

“Apresentamos até agora dois algoritmos que podem ordenar n números no tempo O(n log n). HeapSort alcança esse limite superior no pior caso, enquanto QuickSort o alcança na média.” Frase retirada do Mestrado feito na UNICAMP por André Luis Trevisan com suporte da FAPESP.

Como realizamos a média dos tempos obtidos pudemos avalizar que o QuickSort obteve melhor resultado, contudo se seguir a linha de pensamento do André Luis Trevisan podemos notar certa vantagem no HeapSort, sendo que não teve oscilação nos 3 valores obtidos sendo respectivamente 46,46,46 para os mesmos elementos no vetor. Já o QuickSort teve variação nos 3 elementos obtidos 45,47,46 chegando à média de 46.

Como tivemos 2 campeões nessa avaliação, decidimos avaliar outro critério, sua performance no quesito quantidade de elementos verificados, logo, assim como a tabela na figura 7 apresenta, o QuickSort teve excelente performance em poucos elementos e em muitos elementos comparados além de que atende as necessidades de ordenação no pior caso, no caso médio e no melhor caso com maestria, por esses motivos utilizamos dele para verificações e comparações.

Abaixo apresentamos as especificações da máquina utilizada para os testes:

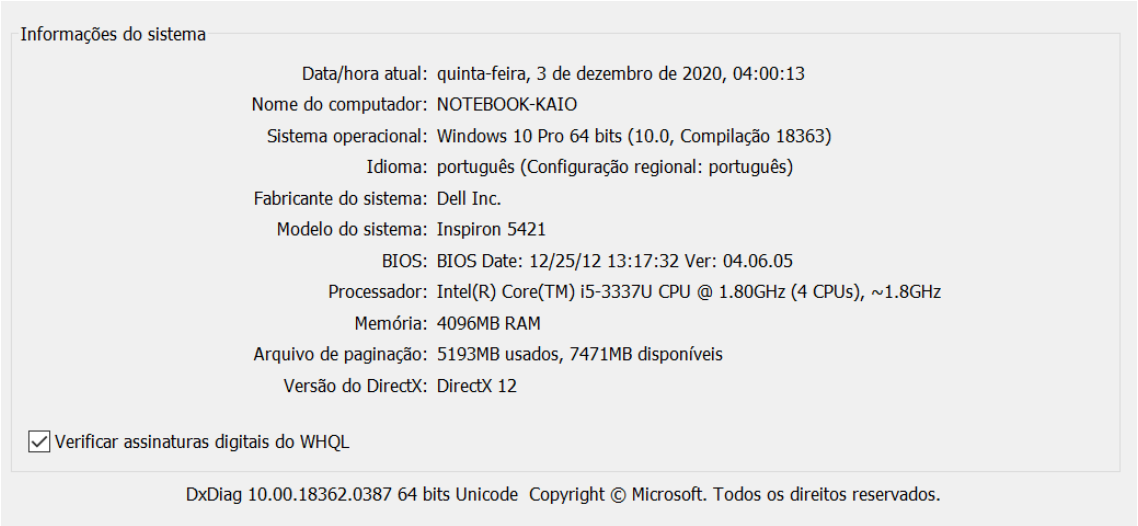

Figura 9 - Informações do sistema obtido pelo DxDiag Software da Microsoft

3.Projeto Digitaliza FATEC

Este projeto visa desenvolver uma solução que vise automatizar todo o processo de inscrição, publicação, análise dos documentos e currículo digital a fim de digitalizar os processos que o CPS tem durante os processos seletivos.

3.1.Técnicas

No projeto utilizamos uma lista encadeada para armazenar os nomes das pessoas, a qual é referida a uma Hash para que possamos armazenar devidamente os nomes por letra, o algoritmo de ordenação foi utilizado QuickSort para realizar a comparação entre os objetos inclusive os textos contidos no objeto a fim de garantir que estará ordenado em ordem alfabética.

Em um primeiro momento criamos o QuickSort para ordenar apenas conteúdos de números, e então adaptamos para comparar textos que estão dentro do objeto guardado no vetor, em seguida ordena da maneira correta.

Além disso, futuramente ao inserir os dados no sistema, em vez de reordenar todo o vetor, por meio do InsertionSort o vetor posicionará o nome na posição

correta do vetor, para manter a ordenação e economizar tempo de processamento futuro com ordenações desnecessárias.

Primeira etapa do projeto foi definido o escopo, para realizar as tarefas necessárias no tempo permitido. Em seguida, buscamos referencias de projetos prontos para compreender melhor a necessidade do solicitante. Como terceira etapa, definimos o que priorizar no sistema, para garantir a entrega por prioridade, semelhante a metodologia SCRUM.

Em um segundo momento, depois de parte da estrutura de dados do projeto estar pronto, passamos a conceituar e aprimorar a documentação. Apesar de não finalizar todas as etapas de codificação, pudemos obter a compreensão sobre a estrutura de dados necessária para implementação em um projeto, além de conseguirmos inseri-las devidamente no sistema mesmo que o sistema não tenha uma interface gráfica, ainda sim é possível utilizar da estrutura de dados para estruturar o vetor, ordenar e oferecer de forma dinâmica a eficiência que o projeto pede.

Como exemplificado na etapa anterior, foram realizados uma serie de testes para definir o melhor algoritmo a ser utilizado. Definimos a massa de dados como estática para não sofrer variação no resultado por parte do conteúdo presente no vetor. O processo de ordenação, permite que em um vetor pequeno, médio ou grande mantenha uma performance no mínimo aceitável para o sistema.

O uso do QuickSort para ordenação da massa de dados já presente no sistema legado oferece eficiência na hora de ordenar. Já o uso do InsertionSort para inserções futuras no sistema permitirá que de forma rápida seja inserido o conteúdo no vetor e salvo em um documento de texto, para que futuramente esse sistema possa armazenar os dados em um banco de dados.

Para realizar a listagem dos valores utilizamos o bloco de notas para gravar o antes e o depois dos elementos, assim como seu tempo de resposta. Por exemplo:

Imagem de um vetor de 10 posições.

```
Desordenado
5 7 9 8 9 7 1 7 1 0

Executado em = 2 ms
Ordenado
0 1 1 5 7 7 7 8 9 9
```

Figura 10 - Exemplo de Saída do programa

O exemplo acima ilustra o sistema em seu primeiro momento que teve o QuickSort inserido sem a comparação dos textos do vetor, apenas comparando inteiros. Posteriormente, a listagem será feita com os nomes presentes na lista de nomes que o sistema legado possui.

A performance presente no sistema é excelente para o melhor caso em que o vetor já está ordenado. Para os casos médio e pior, o vetor ainda sim apresentará o melhor resultado possível no quesito, tempo, custo e performance. Ordenamos abaixo no gráfico em ordem crescente os algoritmos mais eficientes:

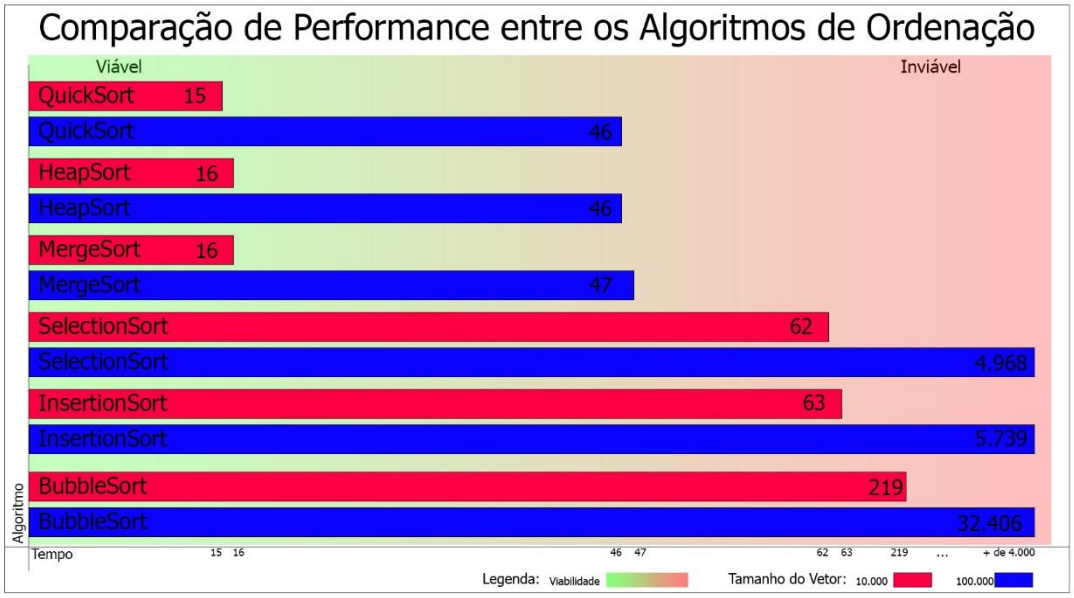

Figura 11 - Comparação. Autor: Kaio V. G. S. Silva

3.2. Algoritmo implementado

PSEUDO CODIGO QUICKSORT

```
QUICKSORT(A, p, r)
    if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)

Chamada inicial: QUICKSORT(A, 1, n)
```

```
PARTITION(A, p, r)
    x = A[r]        // pivot
    i = p − 1
    for j = p to r − 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```

Figura 12 - Livro Introduction to Algorithms, 3rd Edition

Algoritmo 1° versão:

```
// QuickSort Inicio
	public static void QuickSort(int[] vetor) {
		quicksortOrdena(vetor, 0, vetor.length - 1);
	}

	private static void quicksortOrdena(int[] vetor, int esquerda, int direita) {
		if (esquerda < direita) {
			// Define um pivo no vetor e ordena
			int pivo = particao(vetor, esquerda, direita);

			// Quebra em problemas menores
			// recursiva para ordenar do lado esquerdo do vetor
			quicksortOrdena(vetor, esquerda, pivo);
			// recursiva para ordenar do lado direito do vetor
			quicksortOrdena(vetor, pivo + 1, direita);
		}

	}

	private static int particao(int[] vetor, int esquerda, int direita) {
		int meio = (int) (esquerda + direita) / 2;
		// pivo no meio do vetor
		int pivo = vetor[meio];
		// pivo na posicao esquerda
		int pivoEsquerda = esquerda - 1;
		// pivo na posicao direita
		int pivoDireita = direita + 1;

		while (true) {
			do {
				pivoEsquerda++;
				// pivo da esquerda percorrerá o vetor ate que fique ao lado do pivo do meio
			} while (vetor[pivoEsquerda] < pivo);
			do {

				pivoDireita--;
```

```
                        // pivo da direita percorrerá o vetor ate que fique
ao lado do pivo do meio
                    } while (vetor[pivoDireita] > pivo);
                    if (pivoEsquerda >= pivoDireita) {

                        return pivoDireita;
                    }
                    int aux = vetor[pivoEsquerda];
                    vetor[pivoEsquerda] = vetor[pivoDireita];
                    vetor[pivoDireita] = aux;
                }
            }
            // QuickSort Fim
```

PSEUDO CODIGO INSERTIONSORT

```
para j ← 2 até comprimento do vetor, faça
  elemento ← vetor[j]
  i ← j - 1
  enquanto i > 0 e vetor[i] > elemento, faça
    vetor[i + 1] ← vetor[i]
    i ← i - 1
  fim-enquanto
  vetor[i + 1] ← elemento
fim-para
```

Figura 13 - Blog Tiago Madeira

```
// InsertionSort Inicio
        public static void InsertionSort(int[] vetor) {
                int j;
                int chave;
                int i;

                for (j = 1; j < vetor.length; j++) {
                        chave = vetor[j];
                        for (i = j - 1; (i >= 0) && (vetor[i] > chave); i--) {
                                vetor[i + 1] = vetor[i];
                        }
                        vetor[i + 1] = chave;
                }
        }       // InsertionSort FIM
```

3.3.Boas práticas

- Indentar o código: facilita no entendimento do código, onde inicia e onde termina cada comando.
- Nomear variável de maneira intuitiva: nomear as variáveis com o sentido dela no código.
- Evitar Funções dentro de loops: para evitar redundância no código ou retrabalho, é mais interessante chamar funções fora de um ‘for’, dessa forma o código fica mais leve.
- Comentar o código: ideal para facilitar a leitura parcial ou total do código em poucos instantes.
- Evite o máximo de loops: evitar loops reduz a complexidade do código, portanto o torna mais rápido.

4. Considerações finais

Ao realizar esse projeto, pudemos compreender a importância de um sistema bem estruturado, fornecendo diversos motivos como: menor complexidade, maior eficiência e eficácia do código seja em busca seja em armazenamento.
O uso da política de acesso FIFO para esse negócio garante a busca pela classificação (do melhor para o pior) das pessoas, uma elasticidade necessária para atender a grande quantidade de inscritos permitindo otimização do tempo e redução do custo de processamento.

O problema apresentado na Introdução foi resolvido?
Não.
O(s) objetivo(s) do trabalho foi(ram) alcançado(s)?
Sim.
Os resultados obtidos indicam que a solução é viável?
Sim.

Também notamos que no quesito ordenação temos diversos algoritmos eficientes como QuickSort, HeapSort, MergeSort, porém cada um tem sua melhor aplicabilidade, então como desenvolvedor recomendamos analisar a necessidade do seu negócio e aplicar o melhor algoritmo de ordenação.

Ademais, nossos testes se assemelharam em resultados se equiparado ao do Ítalo Cunha do Departamento de Ciência da Computação na UFMG. Onde foi compreendido que o QuickSort se trata do melhor algoritmo de ordenação aleatória e o InsertionSort é o melhor para entradas aleatórias. Independendo do tamanho do vetor podemos fazer as comparações, no melhor tempo possível, com menor custo computacional e maior desempenho.

5.Referências

CUNHA, Ítalo. Comparação entre os métodos de ordenação, homepages.dcc.ufmg.br, janeiro de 2012. Disponível em <https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/sortingcmp.pdf >. Acesso em 03 de novembro de 2020.

MADEIRA, Tiago. Ordenação por Inserção, TiagoMadeira.com , 11 de janeiro de 2006. Disponível em < https://tiagomadeira.com/2006/01/ordenacao-por-insercao/>. Acesso em 19 de novembro de 2020.

SOUZA, Jairo Francisco. QuickSort, ufjf.br, 02 de dezembro de 2009. Disponível em <https://www.ufjf.br/jairo_souza/files/2009/12/2-Ordena%c3%a7%c3%a3o-QuickSort.pdf>. Acesso em 19 de novembro de 2020.

RICARTE, Ivan Luiz Marques. Estruturas de dados, UNICAMP , 2008. Disponível em <http://calhau.dca.fee.unicamp.br/wiki/images/0/01/EstruturasDados.pdf>. Acesso em 19 de novembro de 2020.

FARIAS, Ricardo. Estrutura de Dados e Algoritmos, cos.ufrj.br, 02 de setembro de 2014. Disponível em < https://www.cos.ufrj.br/~rfarias/cos121/aula_08.html>. Acesso em 19 de novembro de 2020.

PÍCOLLO, Homero L. Estrutura de Dados, usuarios.upf.br, 16 de setembro de 2005. Disponível em http://usuarios.upf.br/~mcpinto/ed-tsi/ed_parte01.pdf>.Acesso em 19 de novembro de 2020.

COZMAN, Fabio Gagliardi. Estruturas de Dados — Pilhas, Filas, Listas, sites.poli.usp.br, 1934. Disponível em < http://sites.poli.usp.br/p/fabio.cozman/Didatico/Comp/Material/estruturas.pdf>. Acesso em 19 de novembro de 2020.

PETENATE, Marcelo. O que é FIFO — first in first out — e como usar com eficácia!, ESCOLAEDTI, 18 de fevereiro de 2019. Disponível em <https://www.escolaedti.com.br/o-que-e-fifo-first-in-first-out-e-como-usar-com-eficacia>. Acesso em 19 de novembro de 2020.

FONSECA, Domingos Gonçalves de Oliveira. FIFO, FEFO e LIFO: Vamos falar dessas siglas?, SAUDE BUSINESS, 01 de setembro de 2017. Disponível em

<https://saudebusiness.com/gestao/fifo-fefo-e-lifo-por-que-e-importante-falarmos-dessas-siglas/>. Acesso em 19 de novembro de 2020.

HOINASKI, Fabio. MÉTODO UEPS: QUAL A VANTAGEM PARA SUA EMPRESA?, Blog IBID, 17 de abril de 2017. Disponível em <https://ibid.com.br/blog/metodo-ueps-qual-vantagem-para-sua-empresa/>. Acesso em 19 de novembro de 2020.

FEOFILOFF, Paulo. Quicksort, ime.usp.br, 03 de outubro de 2018. Disponível em < https://www.ime.usp.br/~pf/algoritmos/aulas/quick.html>. Acesso em 19 de novembro de 2020.

TREVISAN, André Luis. ALGORITMOS DE ORDENAÇÃO NA OTIMIZAÇÃO DO VALOR ORDENADO, repositorio.unicamp.br, 25 de fevereiro de 2008. Disponível em <http://repositorio.unicamp.br/bitstream/REPOSIP/307470/1/ Trevisan_ AndreLuis_M.pdf>. Acesso em 19 de novembro de 2020.

LOBO, Fernando. Quicksort, fernandolobo.info, 2017. Disponível em < https://www.fernandolobo.info/aed1718/teoricas/a12.pdf>. Acesso em 19 de novembro de 2020.

William. 10 Boas Práticas de Programação, DevWilliam – Programação, Banco de dados e Linux, 06 de junho de 2016. Disponível em <http://www.devwilliam.com.br/extra/profissional/10-boas-praticas-de-programacao>. Acesso em 19 de novembro de 2020.

CASAVELLA, Eduardo. Boas práticas de programação em Linguagem C, linguagemc.com.br, 2019. Disponível em <http://linguagemc.com.br/boas-praticas-de-programacao-em-linguagem-c/>. Acesso em 19 de novembro de 2020.