**SCC0220 - Laboratório Introdução à Ciência da Computação II**

**Relatório - Aula 3**

**Alunos NUSP**

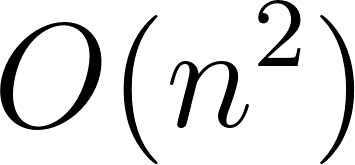
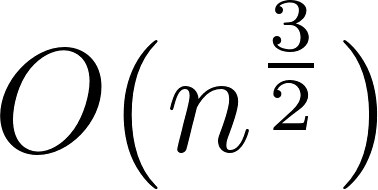
Pedro Henrique de Sousa Prestes 15507819

Pedro Lunkes Villela 15484287

**Trabalho 3 - Seleção**

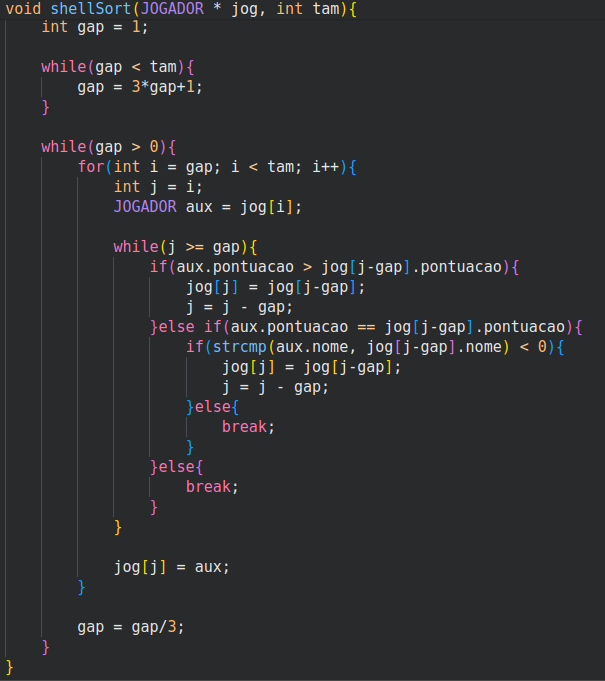
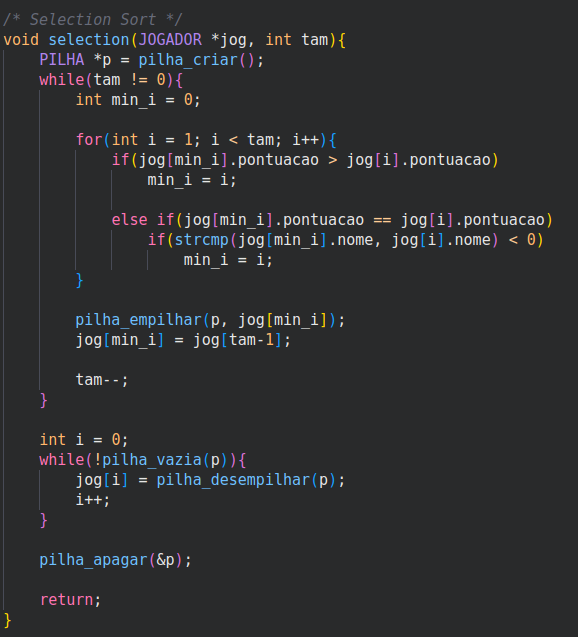
**Selection Sort x Shell Sort**

**🡺 Comentário**

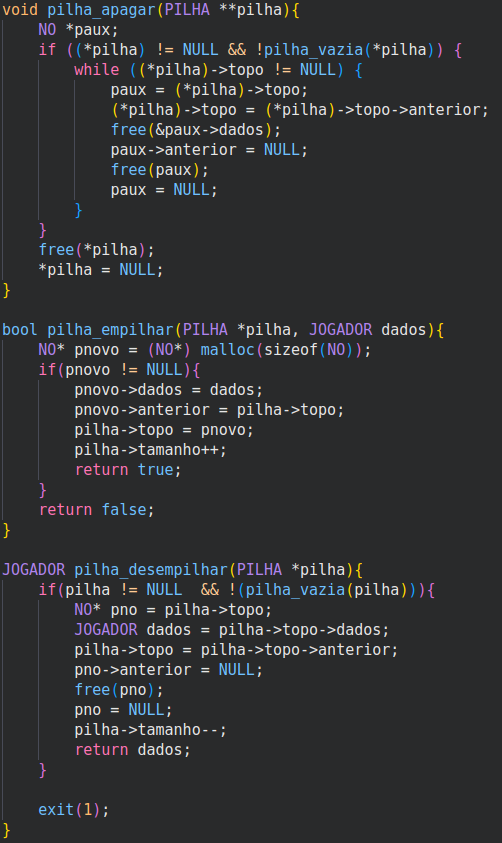
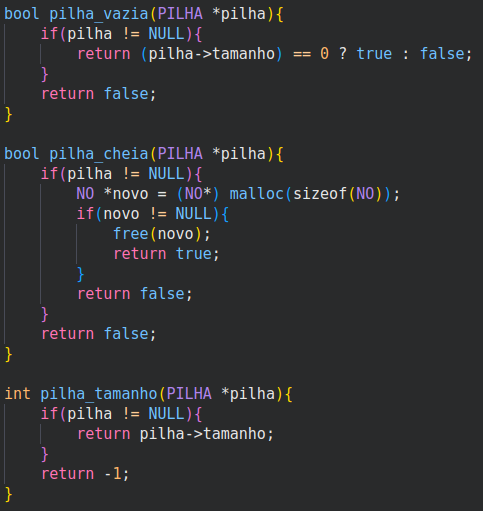
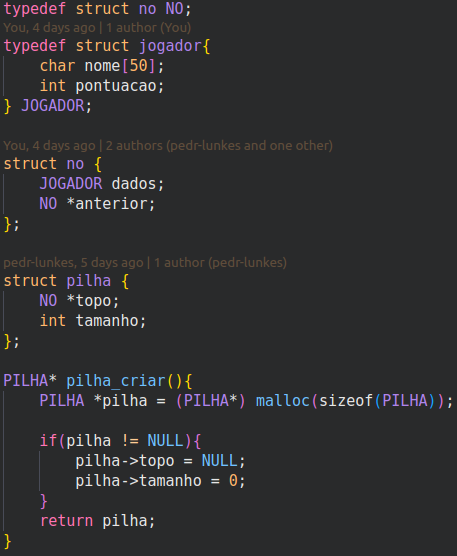
2. ❖ **Selection Sort:** O funcionamento baseia-se na ideia de **selecionar** o menor (ou maior) elemento de uma lista e colocá-lo na posição correta até todos os elementos estarem ordenados. Por ter que selecionar o menor valor no vetor até todos os valores serem selecionados, ele reitera N vezes para todos os N elementos, resultando em uma complexidade [](https://www.codecogs.com/eqnedit.php?latex=O(n%5E%7B2%7D)#0)No exercício em questão, seu funcionamento foi adaptado para empilhar os valores na estrutura de dados Pilha.
3. ❖ **Shell Sort:** Consiste em uma melhoria significativa do método Insertion Sort, dividindo a lista a ser ordenada em vários segmentos e aplicando o método de inserção em cada uma delas, com complexidade variável dependendo de como essa sequência de intervalos é implementada. A sequência de intervalos utilizada no algoritmo usado nesta atividade é a Sequência de Knuth, gerando um algoritmo de complexidade (no pior caso) de aproximadamente [](https://www.codecogs.com/eqnedit.php?latex=O(n%5E%7B%5Cfrac%7B3%7D%7B2%7D%7D)#0).

Para comparar os sorts, os casos são baseados em uma sequência de nomes e valores, representando jogadores de futebol e suas pontuações em um jogo de videogame. A organização deve ser feita em ordem decrescente e realizando comparações lexicográficas para jogadores de mesmas pontuações.

**🡺 Código**



(Implementação do Selection Sort e do Shell Sort)

****(Implementação do TAD Pilha e da struct JOGADOR)

**🡺 Saída**

Seguem alguns casos testados por nós comparando o desempenho de ambos algoritmos.

**1° Caso:**

Tempo de execução: 0.000003 segundos (Selection Sort)

Tempo de execução: 0.000006 (Shell Sort)

Tempo de execução: 0.000004 segundos (Selection Sort)

Tempo de execução: 0.000006 (Shell Sort)

Tempo de execução: 0.000003 segundos (Selection Sort)

Tempo de execução: 0.000002 (Shell Sort)

**3° Caso:**

Tempo de execução: 0.000023 segundos (Selection Sort)

Tempo de execução: 0.000012 (Shell Sort)

Tempo de execução: 0.000049 segundos (Selection Sort)

Tempo de execução: 0.000027 (Shell Sort)

Tempo de execução: 0.000049 segundos (Selection Sort)

Tempo de execução: 0.000037 (Shell Sort)

**6° Caso:**

Tempo de execução: 0.463462 segundos (Selection Sort)

Tempo de execução: 0.019632 (Shell Sort)

Tempo de execução: 0.432514 segundos (Selection Sort)

Tempo de execução: 0.006716 (Shell Sort)

Tempo de execução: 0.401974 segundos (Selection Sort)

Tempo de execução: 0.020326 (Shell Sort)

**8° Caso:**

Tempo de execução: 26.455711 segundos (Selection Sort)

Tempo de execução: 0.063506 (Shell Sort)

Tempo de execução: 27.166242 segundos (Selection Sort)

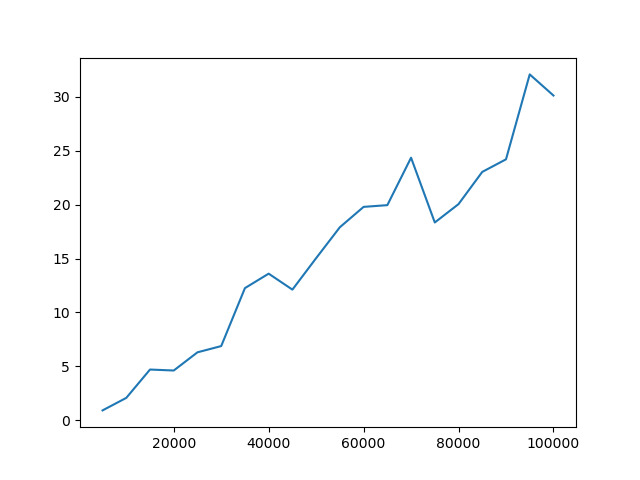
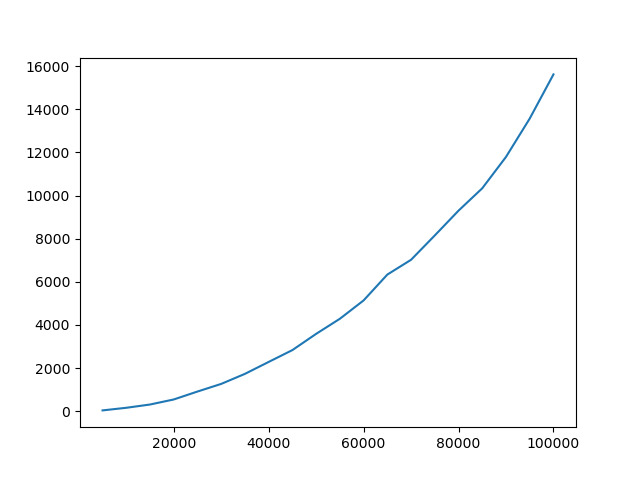
Tempo de execução: 0.066794 (Shell Sort)

Tempo de execução: 27.062608 segundos (Selection Sort)

Tempo de execução: 0.055689 (Shell Sort)

**🡺 Conclusão**

O Shell Sort mostrou-se mais eficiente comparado ao Selection Sort, tendo uma curva de crescimento significativamente menor para casos com muitas entradas.

(Gráfico Selection e Shell, Eixo X representa entradas e o Eixo Y milissegundos, feito a partir de casos teste a parte do Runcodes)