UNISINOS

UNIVERSIDADE DO VALE DO RIO DOS SINOS
Escola Politécnica
Ciência da Computação

Algoritmos e Programação: Estruturas de Dados

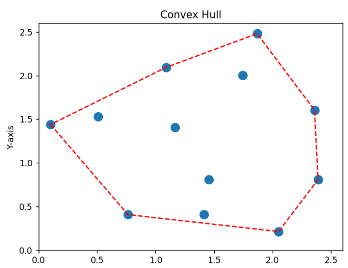
Prova do Grau B

1. Explique, com suas palavras, os seguintes conceitos relacionados à Complexidade de Algoritmos:
    a) notacão Big-O
    b) melhor, pior e caso médio.

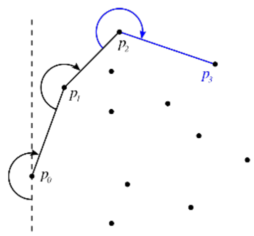
2. Considerando um algoritmo e sua representação de complexidade de tempo f(n) expressa pela notação Big-O, relacione as colunas:

a) $f(n) = O(1)$ b) $f(n) = O(log\, n)$ c) $f(n) = O(n\, log\, n)$ d) $f(n) = O(n^2)$ e) $f(n) = O(2^n)$	( ) ocorre tipicamente em algoritmo que resolvem um problema transformando-o em problemas menores ( ) ocorre tipicamente quando os itens de dados são processados aos pares, em um, loop dentro de outro. ( ) ocorre tipicamente em algoritmos que resolvem um problema quebrando-o em problemas menores, resolvendo cada um deles independentemente e depois juntando as soluções ( ) o algoritmo independe do tamanho de n, as instruções são executadas um número fixo de vezes ( ) são os algoritmos do tipo "força-bruta", nada práticos para processar ordenação

3. O algoritmo "Embrulho de presente" (Jarvis March) é utilizado para encontrar o fecho convexo de um conjunto de pontos (**Figura 1.a)**). Considerando que n é o número total de pontos e h é o número de pontos no fecho convexo, a complexidade do algoritmo é $O(nh)$.



a) Exemplo de fecho convexo: os n pontos azuis representam o conjunto S, enquanto os pontos interligados pela linha vermelha são os h pontos que pertencem ao fecho convexo de S



b) Representação da busca pelo ponto mais à esquerda do ponto final durante a execução do algoritmo (você pode ver uma animação bastante didática aqui)

Figura 1. Representação do problema do fecho convexo a) e da execução do algoritmo "Embrulho de presente"

O algoritmo funciona da seguinte forma, como mostra o **Algoritmo 1**. Iniciando a partir do ponto mais à esquerda no conjunto inicial, o algoritmo itera através de uma série de passos que progressivamente adicionam os pontos que formam o fecho convexo. Em cada iteração, ele seleciona o próximo ponto que faz o ângulo mais à esquerda em relação ao último ponto adicionado (**Figura 1.b**), garantindo que o fecho convexo seja formado de maneira incremental. Esse processo continua até que o ponto final coincida com o ponto inicial no fecho convexo, completando assim a construção do fecho convexo.

Algorithm 1 Jarvis March (Embrulho de presente)

```
Entrada: Conjunto de pontos S
Saída: Conjunto de pontos P que formam o fecho convexo
pontoNoFecho ← ponto mais à esquerda em S
i ← 0
repeat
    P[i] ← pontoNoFecho
    pontoFinal ← S[0]
    for j ← 0 to |S| − 1 do
        if pontoFinal == pontoNoFecho or S[j] está à esquerda da linha de
        P[i] para pontoFinal then
            pontoFinal ← S[j]
        end if
    end for
    i ← i + 1
    pontoNoFecho ← pontoFinal
until pontoFinal == P[0]
```

Algorimo 1. Pseudocódigo do algoritmo "Embrulho de Presente" (Jarvis March) para computar o fecho convexo de um conjunto de pontos S.

Para cada uma das afirmações abaixo, marque (V) para verdadeiro ou (F) para falso:

a) ( ) Para cada ponto no fecho convexo, o algoritmo verifica todos os outros pontos para encontrar o próximo ponto do fecho.

b) ( ) O melhor caso para o algoritmo ocorre quando todos os pontos estão no fecho convexo, resultando em complexidade $O(n \log n)$

c) ( ) O pior caso ocorre quando todos os pontos estão no fecho convexo, resultando em complexidade $O(n^2)$.

d) ( ) Excluindo o caso em que $h = n$, podemos afirmar que a complexidade do algoritmo de Jarvis March é linear, $O(n)$, em relação ao número de pontos.

e) ( ) Quando h é significativamente menor que n, a complexidade do algoritmo se aproxima de $O(n)$.

f) ( ) O algoritmo Jarvis March é mais eficiente que o algoritmo Graham Scan (que é um outro algoritmo para calcular o fecho convexo), com complexidade $O(n\ log\ n)$, em qualquer situação.

4. Você foi contratado(a) para desenvolver um sistema de narrativa não linear para um jogo de aventura. Cada episódio da história terá no máximo 3 caminhos diferentes que o jogador poderá escolher. Nesse sistema, é importante representar as informações dos episódios, suas escolhas (que o levará para o próximo episódio) e os diferentes caminhos de forma eficiente.

Responda:

a) Qual seria estrutura de dados não linear mais adequada para armazenar e gerenciar as informações dos episódios, suas escolhas e os diferentes caminhos da narrativa?

b) Considerando que a história terá 5 episódios, qual o número total de alternativas (caminhos diferentes) que o jogador poderá percorrer ao longo da narrativa? Explique como você chegou a esse resultado, utilizando a estrutura de dados escolhida na pergunta anterior.

5. Relacione os métodos de ordenação abaixo com as informações correspondentes:

Métodos de Ordenação:

a) Bubble Sort
b) Insertion Sort
c) Heap Sort
d) Shell Sort
e) Merge Sort
f) Quick Sort

Informações:

( ) Baseado em dividir o problema em subproblemas menores e conquistar cada subproblema individualmente.

( ) Usa uma estrutura de dados de heap para organizar os elementos.

( ) Compara pares de elementos adjacentes e os troca se estiverem fora de ordem.

( ) Insere cada elemento em sua posição correta em uma sublista ordenada.

( ) Escolhe um pivô e particiona o array em elementos menores e maiores que o pivô.

( ) Melhora a ordenação por inserção comparando elementos distantes e gradualmente diminuindo a distância.

6. Com base no método de classificação denominado *Counting Sort* abaixo, responda:

a) É estável?
b) É baseado em comparação?
c) É recursivo?
d) Qual sua complexidade de tempo?
e) Qual sua complexidade de espaço?

```java
public static void countingSort(int[] arr) {
    int n = arr.length;

    // Encontre o valor máximo no array de
        entrada
    int max = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];

    // Crie o array de contagem com tamanho
        max+1
    int[] count = new int[max + 1];

    // Inicialize o array de contagem com
        zeros
    for (int i = 0; i <= max; i++)
        count[i] = 0;

    // Modifique o array de contagem para
        armazenar as posições dos
        elementos
    for (int i = 1; i <= max; i++)
        count[i] += count[i - 1];

    // Construa o array de saída
    int[] output = new int[n];
    for (int i = n - 1; i >= 0; i--) {
        output[count[arr[i]] - 1] =
         arr[i];
        count[arr[i]]--;
    }
    // Copie os elementos ordenados de
        volta para o array de entrada
    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}
```

```
// Armazene a contagem de cada elemento
    no array de entrada
for (int i = 0; i < n; i++)
    count[arr[i]]++;
```

BOA PROVA! 😊

Dica: lembre-se que um problema complexo pode ser decomposto em problemas menores.