1) (não consegui a pergunta em si) Notações assintóticas, tipo:
A) prove q 5n^3 é O(cn^2) para todo c >= 5n

2) Explique quais as características de um algoritmo guloso apontando sua vantagem e desvantagem em comparação com algoritmos que se baseiam em uma estratégia de tentativa e erro

@ gabriel
O algorítmo guloso soluciona os problemas baseado em tomada de decisões que são mais "agradáveis" ao mesmo, sendo que este jamais se arrepende da decisão tomada. Comparado aos algoritmos de tentativa e erro, tem como vantagem a rápida execução - e, portanto, performance - e possuí como desvantagem o fato de não conseguir atingir com 100% de certeza uma solução ótima para o problema - na maioria das vezes, o algorítmo alcança uma solução boa, dependendo da condição de tomada de decisão definida pelo mesmo.

3) (não consegui a pergunta em si)São apresentadas duas buscas, sequencial e binária, profundidade da árvore de recursão

4) Em relação ao algoritmo de ordenação MERGE (versão recursiva)
A) Explique detalhadamente o princípio de fundamento do algoritmo

@ marco
Merge sort é baseado em indução forte, ou seja, caso base pra um elemento, passo é dividir o problema maior até chegar a um, ele estara ordenado. Isto é: divisão e conquista. O trabalho do algoritmo está concentrado na conquista: a intercalação dos dois subvetores ordenados. Para simplificar a implementação da operação de intercalação e garantir sua complexidade linear, usamos um vetor auxiliar.

B) Desenhe a árvore de recursão referente a execução do algoritmo para o vetor {2, 5, 7, 3, 1, 4, 6, 0}. Indique de forma clara no desenho a ordem em que as chamadas recursivas são feitas e o estado de cada subvetor.

@ marco
ex: merge(int[]A, ini, fim)
A = {2, 5, 7, 3, 1, 4, 6, 0}
merge(A, 0, 7)
meio = 3 //indice, não o valor do vetor
{2, 5, 7, 3, 1, 4, 6, 0}
merge(A, 0, 3)
{2, 5, 7, 3}
meio = 1
merge(A, 0, 1)
{2, 5}
merge(A, 0, 0)
{2}
merge(A, 1, 1)

{5}
//agora sim compara o 2 com o 5
2 < 5 //troca se não for
merge(A, 2, 3)
{7, 3}
merge(A, 2, 2)
{7}
merge(A, 3, 3)
{3}
7 < 3
// arruma eles no vetor
{3, 7}
// faz as comparações q eu não sei a ordem!!!!!!
{2, 3, 5, 7}
merge(A, 4, 7)
{1, 4, 6, 0}
meio = 5
merge(A, 4, 5)
{1, 4}
merge(A, 4, 4)
{1}
merge(A, 5, 5)
{4}
1 < 4
merge(A, 6, 7)
{6, 0}
merge(A, 6, 6)
{6}
merge(A, 7, 7)
{0}
6 < 0 //não, então troca
{0, 6}
// faz as comparações q eu não sei a ordem!!!!!!
{0, 1, 4, 6}
// faz as comparações q eu não sei a ordem!!!!!! agora do array completo
{0, 1, 2, 3, 4, 5, 6, 7}

C) Demonstre que o merge sort possui complexidade de tempo O(nlogn) no pior caso

@gabriel

Visto que o merge sort SEMPRE irá dividir o problema em dois subproblemas a cada chamada recursiva (até atingir o caso base - onde o subvetor terá tamanho 1), podemos definir que a equação de recorrencia do mergeSort será, tanto no pior quanto no melhor caso, a seguinte:

T(n) = 2T(n/2) + [COMPLEXIDADE DO METODO DE INTERCALAÇÃO]

Ou seja, o unico fator que impactará na complexidade final do algorítmo é a complexidade do método de intercalação. Portanto, tomando como base um algoritmo de intercalação ideal (θ(n)), temos:

$T(n) = 2T(n/2) + \theta(n)$

Resolvendo pelo teorema mestre:

Caso 2:

f(n) = θ((nlog22) = Θ(n), logo: T(n) = θ(nlogn), **tanto para o pior caso quanto para o melhor caso.**