Universidade Federal Fluminense / Instituto de Computação
Disciplina: Análise e Projeto de Algoritmos (Graduação) **Prof.** Carlos Martinhon

Divisão e Conquista e Método Guloso– Lista 2

01 – Seja S uma lista ordenada com n elementos arbitrários. Faça uma busca do tipo "ternária" que procure um elemento CHAVE nesta lista. Calcule a complexidade desse algoritmo.

02 - Modifique o algoritmo da *Bolha* para ordenação de forma que tenhamos $\Omega(n)$ passos no melhor caso.

03- Mostre através de esquemas como ordenar as listas A_1 e A_2 pelo Algoritmo *Quicksort*: $A_1 = \{4,7,2,1,6,6,8,5\}$ e $A_2 = \{2,2,2,3,3,3,3,3,1,1\}$.

04 – Considere uma lista não ordenada L com n valores inteiros. Suponha que um *Quicksort Modificado* seja obtido escolhendo-se o elemento da metade da lista como *pivô* (lembre-se que escolhíamos sempre o primeiro elemento – *Quicksort padrão*). Suponha agora que o *Quicksort padrão* seja executado normalmente após a troca deste elemento com o elemento na primeira posição de *L.* Construa uma lista de 9 elementos para este *Quicksort modificado* (pior entrada), de forma que sejam necessários $O(n^2)$ passos para sua ordenação completa.

05 - A ordenação por inserção (*Insertion sort*) pode ser facilmente exemplificada através das cartas de um baralho de mesmo naipe. Considere inicialmente que as cartas a serem ordenadas estejam todas em uma das mãos. Em seguida, as cartas são colocadas uma a uma sobre a mesa de maneira que a seqüência de cartas obtidas esteja ordenada. Para achar a posição correta de uma carta, comparamos esta carta com cada uma das cartas sobre a mesa da direita para a esquerda (inserção linear). Repete-se o processo até que todas as cartas estejam sobre a mesa.

a) Faça um algoritmo não-recursivo para este problema e calcule sua complexidade.
b) Seja S={14, 3, 8, 5, 11, 7, 2}. Ilustrando cada passo, ordene S pelo algoritmo da ordenação por inserção
c) O algoritmo de ordenação por inserção pode ser expresso por um algoritmo recursivo. Para ordenação de $A=\{x_1,x_2,...,x_n\}$, ordenamos recursivamente os elementos de $A'=\{x_1,x_2,...,x_{n-1}\}$ e inserimos o elemento x_n na sub-lista A' (já ordenada). Escreva um algoritmo recursivo para este problema e calcule sua complexidade.
d) Utilizando-se uma busca binária na inserção de uma nova carta (ao invés da inserção linear), poderemos reduzir a complexidade assintótica da algoritmo de ordenação para $O(n\log n)$? Justifique sua resposta.

08 – Suponha que, a cada escolha do elemento *chave* no algoritmo *Quicksort* tenhamos sempre 2 sub-listas com $9n/10$ e $n/10$ elementos respectivamente. Mostre que a complexidade do *Quicksort* neste caso será de $O(n\log n)$ iterações.

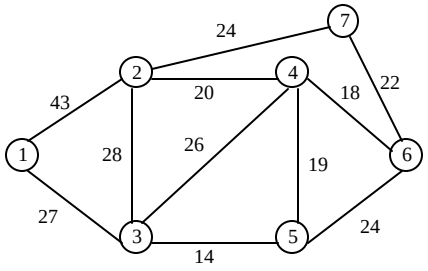
09 - Mostre que para determinar o menor elemento em um lista de *n* elementos são necessários pelo menos *n*-1 comparações (limite inferior do problema).

10 – Seja *G*=(*V,E*) um grafo não orientado com pesos nas arestas. Mostre que a menor aresta de G deve pertencer a Árvore Geradora de Custo Mínimo. (**Sugestão:** prova por absurdo).

11 - Reescreva o Algoritmo de *Dijkstra* e diga os vértices presentes no caminho mínimo.

12 – Mostre que o Algoritmo de *Dijkstra* não pode ser aplicado a grafos com arcos de custo negativo. Construa um contra-exemplo.

13 - Observe o grafo seguinte:



Calcule pelos métodos de *Prim* e *Kruskal* a árvore geradora do grafo (justifique sua resposta). Os métodos acimas são gulosos? Por quê?

14 - Proponha uma heurística guloso para os seguintes problemas: (a) Coloração de Vértices, (b) Conjunto Independente, (c) Cobertura de Vértices. Calcule também sua complexidade. (**Nota:** o enunciado destes problemas pode ser encontrado no último capítulo da Apostila de APA). Seu algoritmo sempre encontra uma solução ótima? Em caso negativo, formule um contra-exemplo.

15 – Considere *n* programas de comprimentos $c_1, c_2, \ldots, c_n$. Esses programas devem ser armazenados em uma fita. O programa **i** é executado com frequência f_i. Se os programas são armazenados na ordem $i_1, i_2, \ldots, i_n$ o tempo esperado de resgate (tempo médio) é:

$$TRE = \frac{\sum_j (f_{i_j} \sum_{k=1}^{j} c_{i_k})}{\sum f_i}$$

a) Mostre que armazenando os programas em ordem decrescente de frequência não chegaremos necessariamente ao mínimo do tempo de resgate esperado (TRE).

b) Faça o mesmo supondo que os programas são armazenados em ordem crescente de comprimento. Teremos neste caso uma estratégia gulosa eficiente?

c) Qual estratégia gulosa poderia ser adotada para o problema acima? Ilustre com um exemplo.

16 – Dado um grafo G=(V,E) não-orientado, dizemos que $W \subseteq V$ define uma *cobertura* de vértices see, $\forall$ (i,j)$\in$ E, tivermos i$\in$ W ou j$\in$ W. No problema da *Cobertura Mínima* desejamos determinar o subconjunto W de menor cardinalidade. Mostre (exibindo um contra-exemplo) que algoritmo guloso abaixo não garante solução ótima:

Procedimento: Cobertura Mínima (Guloso)
Dado: G=(V,E)
W $\leftarrow$ Selecione vértice *v* de grau máximo (em G);
Repita
E $\leftarrow$ E – {(v,w) | (v,w) $\in$ E}
W $\leftarrow$ W $\cup$ {*v* $\in$ V | v tem grau máximo em G=(V,E)};
Até que E = $\varnothing$;
Retorna W;
fim