Estruturas de Dados II - 2012-2

L5

Algoritmos de Ordenação e Complexidade de Algoritmos

Nas questões abaixo, você pode usar as seguintes funções:

qsort(v, N): ordena o vetor v de tamanho N em $O(N*\log_N)$

vmax(v, N): retorna a posição do maior elemento de um vetor de tamanho N em O(N)
vmin(v, N): retorna a posição do menor elemento de um vetor de tamanho N em O(N)
bsearch(v, N, num): retorna a posição do elemento **num** em um vetor de N elementos em $O(\log_N)$.

Se não encontrar o elemento **num** no vetor, retorna -1.

1) Crie um algoritmo $O(N^2)$ para ordenar apenas os elementos ímpares de um vetor de inteiros de tamanho N. O algoritmo deve manter os inteiros pares inalterados na mesma posição original. Note que sua solução será considerada incorreta se for $O(N*\log_N)$ ou qualquer complexidade diferente de $O(N^2)$.

2) Você precisa ordenar um vetor com N números inteiros de acordo com o valor não-decrescente absoluto de cada inteiro usando o *mergesort*. Lembrete: |-1| = 1 (o valor absoluto de -1 é 1) e |2| = 2 (o valor absoluto de 2 é 2). Isso significa que o vetor {-1, 2, -3, 4, -5} está ordenado de acordo com os valores absolutos dos inteiros. Note que normalmente em ordem não decrescente o vetor ficaria ordenado como {-5, -3, -1, 2, 4}. Outra observação importante é que -3 e 3 são considerados inteiros iguais para efeito dessa ordenação de tal forma que os vetores {-3, 3, -4} e {3, -3, -4} estão corretamente ordenados. Você já implementou a função *mergesort* e falta apenas agora implementar a primitiva *merge*. Crie o código da função de *merge* para ser usada nesse algoritmo de ordenação.

3) Uma fila de prioridades é uma estrutura de dados em que elementos são inseridos na fila de tal forma que o próximo a ser retirado é sempre aquele que tem a maior prioridade. Você está implementando uma fila de prioridades com números reais e quanto maior o número, maior sua prioridade. Crie as duas primitivas básicas da fila de prioridade abaixo de tal forma que a complexidade de cada uma delas seja O(logN). Importante: soluções O(N) serão consideradas incorretas.

```
insert(Fila f, Real r); // O(logN)
Real remove(Fila f); // O(logN)
```

4) Exemplifique o funcionamento do algoritmo de countsort ordenando **passo-a-passo** o vetor a seguir. Não precisa demonstrar todos os passos, bastando apenas exemplificar os dois ou três primeiros passos de cada etapa do algoritmo e depois o estado final dessa etapa.

1	5	2	3	4	0	6	2	4	2	6	3	4	0	5	2

5) Há alguns bugs no código a seguir, escrito em C++, que deveria ordenar de forma não-decrescente um vetor de inteiros. Corrija-o.

```
#include <algorithm>
void recursiveSort(int *V, int N) {
   int maior = vmax(V, N);
   int menor = vmin(V, N);
   std::swap(V[0], V[maior]); // essa função troca V[0] por V[maior]
   std::swap(V[N-1], V[menor]); // troca V[N-1] por V[menor]
   recursiveSort(V+1, N-2);
}
```