Dredd - Juiz Online

Principal | Perfil | Minhas Provas | Sair

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Exercícios de Ordenação

Prova Aberta Até: 30/11/2019 06:00:00

Número Máximo de Tentativas: 6

Atenuação da Nota por Tentativa: 0%

Instruções para a prova:

Questão 1: Ordenação - Selection Sort

Uma forma de realizar a ordenação é pesquisando o menor valor presente no vetor e colocá-lo em seu devido lugar. Diante dessa ideia elabore um algoritmo que implemente o método **Selection Sort** e que tenha como entrada o tamanho n do vetor e os números para preenchê-lo.

Entrada
5
6
2
7
1
8

Saida
1
2
6
7
8

Peso: 1

Última tentativa realizada em: 21/11/2019 21:42:52

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado | Enviar Resposta

Questão 2: Ordenação - Selection Sort passo a passo

Implemente a seguinte variação do algoritmo de ordenação *selection sort*: Procurar o maior valor de um vetor e trocá-lo com o que estiver na última posição do vetor. Em seguida, procurar o segundo maior valor e trocá-lo com o que estiver na penúltima posição do vetor. Assim sucessivamente, até que o vetor esteja ordenado.

Não existe teste para verificar se um valor está trocado com ele mesmo. Veja no exemplo que nesta situação, o vetor é o mesmo antes e depois da troca.

Entradas:

1. Tamanho do vetor que será ordenado.
2. Vários número inteiros que serão ordenados.

Saídas:

1. Os elementos do vetor a cada troca de valor.

```
Exemplo de entrada:
5
4 1 7 2 3

Exemplo de saída:
4 1 3 2 7
2 1 3 4 7
2 1 3 4 7
1 2 3 4 7
```

Peso: 1

Última tentativa realizada em: 24/11/2019 22:20:54

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 3: Ordenação - Ordenação nas Metades do Vetor

Implemente um programa que receba vários números inteiros, armazene os números num vetor e ordene a primeira metade deste vetor em ordem crescente e a segunda metade em ordem decrescente. Caso a quantidade de números seja ímpar, o elemento do meio deve ser considerado como parte da primeira metade.

As ordenações devem ser feitas usando algum dos algoritmos vistos em aula.

Entradas:

1. A quantidade de números a ler.
2. Vários números inteiros para armazenamento num vetor (numa mesma linha).

Saídas:

- Os valores do vetor, após as ordenações das metades.

Exemplo de entradas:

```
10
5 3 8 6 10 5 6 3 4 7
```

Exemplo de saídas:

```
3 5 6 8 10 7 6 5 4 3
```

Exemplo de entradas:

```
7
65 91 0 45 87 10 32
```

Exemplo de saídas:

```
0 45 65 91 87 32 10
```

Peso: 1

Última tentativa realizada em: 21/11/2019 23:32:39

Tentativas: 2 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 4: Ordenação - Bubble Sort

Faça um programa que receba um vetor de **N** posições e o ordene usando o **método bolha**.

O **bubble sort**, ou **ordenação por flutuação** (literalmente **"por bolha"**), é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vetor diversas vezes, a cada passagem fazendo flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo. (Wikipédia)

Entradas:

- `int n` - Tamanho do vetor que deve ser ordenado.
- `float vet[n]` - Vetor que será preenchido e ordenado.

Saídas:

- Vetor ordenado (`float`).

Exemplos de Entradas e Saídas:

Entradas:

```
10
5   3   4   2   1   6   7   9   8   10
```

Saídas:

```
1   2   3   4   5   6   7   8   9   10
```

Referências:

- Link: Método Bolha

Peso: 1

Última tentativa realizada em: 21/11/2019 23:36:48

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 5: Matriz - Ordenar Linhas Pares Crescente, Ímpares Descrescente

Faça um programa que receba uma matriz quadrada (NxN) e a preencha. Feito isso, ordene as linhas pares em ordem crescente e as linhas ímpares em ordem decrescente. Imprima a nova matriz.

Entradas:

- `int n` - Tamanho da matriz quadrada.
- `int mat[n][n]` - Valores que preencherão a matriz.

Saídas:

- Matriz ordenada de forma crescente nas linhas pares e decrescente nas linhas ímpares (`int`).

Exemplos de Entradas e Saídas:

Entradas:

```
4
12  42   3   1
 6  14  53  32
44  31  26  52
 7   8   9   2
```

Saídas:

```
 1   3  12  42
53  32  14   6
26  31  44  52
 9   8   7   2
```

Peso: 1

Última tentativa realizada em: 24/11/2019 23:07:16

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 6: Registros - Busca e Ordenação em Registros

Faça um programa que receba N entradas de cadastro de alunos. Cada cadastro de aluno é definido por dois campos:

```
struct aluno {
    int id;
    int matricula;
};
```

Os valores recebidos estarão desordenados, você deve ordenar o vetor pelo campo `id` do registro de aluno. Após entrar as N entradas, o programa deve receber 4 (quatro) ids de alunos e imprimir a matricula correspondente à cada id. Caso um id não seja encontrado, deve-se imprimir a matrícula 0 (zero). A busca pelos ids deve ser feita pelo método da Busca Binária.

Entradas:

- `int n;` - Número de registros de alunos que serão recebidos.
- `struct aluno alunos[n];` - Vetor (desordenado pelo campo `id`) de registros de alunos. Será fornecido primeiro o id e depois a matricula, um par para cada aluno, exemplo: id1 matricula1 id2 matricula2 ...
- `int idsParaBusca[4];` - Quatro ids para serem buscados nos registros recebidos.

Saídas:

- IDs ordenados dos alunos.
- A matrícula referente a cada um dos quatro ids recebidos para serem buscados no vetor.

Exemplos de Entradas e Saídas

Entradas:

```
5
10 200910040
1 200810440
5 200820507
17 200810110
```

```
2 200920240
2 17 5 4
```

Saídas:

```
1 2 5 10 17
200920240
200810110
200820507
0
```

Peso: 5

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Questão 7: Ordenação - vetores - Ordena até posição

Nem sempre deseja-se ordenar um vetor inteiro. Faça um programa que receba um vetor de tamanho n e a posição até a qual deseja-se ordenar. O programa deverá imprimir o vetor resultante.

OBS: deve ser utilizado um dos método de ordenação visto na disciplina

Entradas:

- `int n` - tamanho do vetor
- `int vetor[n]` - vetor de tamanho n.
- `int pos` - posição até onde o vetor deve ser ordenado.

Saídas:

- `int vet[n]`

Exemplos de Entrada e Saída:

Entradas:
5
5 4 3 2 1
2

Saídas:
3 4 5 2 1

Peso: 1

Última tentativa realizada em: 14/11/2019 13:46:06

Tentativas: 2 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 8: Ordenação - Torneio de futebol.

Mario é um grande fã de futebol e por isso decidiu realizar um torneio com times amadores de sua cidade, ficou decidido que todos os time jogariam entre si onde uma vitória daria 3 pontos empate 1 ponto e derrota 0 pontos, ao final do torneio o time vencedor seria aquele com maior número de pontos, caso houvesse empate seria escolhido o time com maior número de vitórias, persistindo o empate a escolha seria feita pelo time com maior saldo de gols (considere que não existirão times com saldo de gols iguais), Mario no entanto não ordenou a tabela do campeonato durante a competição e agora pede a você que crie um programa que a ordene e informe sua ordem correta.

Entradas:

1. Quantidade de times que participaram da competição.
2. vetor de registros representando os times contendo os campos de nome do time (cadeia de caracteres), pontos que o time fez no decorrer da competição (número inteiro), número de vitórias (número inteiro) e saldo de gols (número inteiro).

Saídas:

1. O programa deve ordenar o vetor dos times modo que o time com mais pontos apareça primeiro seguido do segundo com mais pontos e assim por diante, caso ocorra empate no número de pontos seu programa deve ordenar os times empatados pelo número de vitórias, se o empate ainda existir o programa deve ordenar pelo saldo de gols.

Exemplo de Entrada:

```
5
DesertorFC 9 3 10
CarrascoFC 13 4 7
Campinais 13 4 9
```

```
Machado 11 3 10
RealSport 1 0 -4
```

Exemplo de Saída:

```
Campinais 13 4 9
CarrascoFC 13 4 7
Machado 11 3 10
DesertorFC 9 3 10
RealSport 1 0 -4
```

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Peso: 1

Última tentativa realizada em: 25/11/2019 14:43:05

Tentativas: 2 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 9: Ordenação - Insertion Sort Passo a Passo

Implemente o algoritmo de ordenação *insertion sort* que percorre o vetor do inicio ao fim e coloca cada valor em sua correta posição a esquerda desse valor. Assim o vetor sera percorrido da posição 1 (segunda posição) ate a última posição. Cada passo do algoritmo devera ser imprimido, ou seja, a cada iteração do loop principal o estado atual do vetor devera ser impresso.

Entradas:

1. Tamanho do vetor que será ordenado.
2. Os diversos valores para ordenação.

Saídas:

1. Cada passo da ordenação. Ou seja, os valores do vetor a cada inserção feita.

Exemplo de entradas:

```
6
4 7 2 3 8 6
```

Exemplo de saídas:

```
4 7 2 3 8 6
2 4 7 3 8 6
2 3 4 7 8 6
2 3 4 7 8 6
2 3 4 6 7 8
```

Peso: 1

Última tentativa realizada em: 25/11/2019 11:42:00

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 10: Ordenação - Mediana

Em estatística, a **mediana** é a medida de tendência central, ou seja, metade dos valores são menores que a mediana e metade são maiores que ela. Caso a quantidade de informações seja par, a mediana é a média simples dos dois valores medianos.

Faça um programa que calcula a mediana de um conjunto de valores.

A entrada de dados é composta de vários números positivos. Um número negativo na entrada indica o final dos valores.

Obs: Para programas em C++ considere o tamanho máximo do vetor como 20.

```
Exemplo de entrada:
4.4 5.1 1.2 9.3 -1
```

```
Exemplo de saída:
4.75
```

Peso: 1

Última tentativa realizada em: 25/11/2019 21:57:58

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 11: Ordenação - Completar o Merge Sort

Você deverá fazer um programa que recebe um vetor de inteiros V de N posições e o ordena. Para ordenar este vetor, você deverá usar o método **Merge Sort**.

Dica: Abaixo temos uma implementação **incompleta** do Merge Sort. Você deverá completá-la se quiser utilizá-la.

```
void Troca(int &A, int &B)
{
        int aux = A;
        A = B;
        B = aux;
}

void Merge(int V[], int p, int q, int r, int U[])
{
        int a = p;
        int b = q+1;

        for(int i = p; i <= r; i++)
                if( b > r || (a <= q && V[a] < V[b]) )
                        U[i] = V[a++];
                else
                        U[i] = V[b++];

        for(int i = p; i <= r; i++)
                V[i] = U[i];
```

```
}

void MergeSort(int V[], int primeira, int ultima, int U[])
{
        if(primeira >= ultima) return;

        int p = primeira;
        int r = ultima;
        int q = (p+r)/2;

}
```

Entradas:

- Tamanho N do vetor.
- Os N elementos do vetor V.

Saídas:

- Vetor V ordenado.

Exemplo de Entrada:

```
5
7 8 5 3 6
```

Exemplo de saída:

```
3 5 6 7 8
```

Peso: 1

Última tentativa realizada em: 14/11/2019 13:35:41

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 12: Ordenação - Completar o Quick Sort 1

Você deverá fazer um programa que recebe um vetor de inteiros V de N posições e o ordena. Para ordenar este vetor, você deverá usar o método **Quick Sort**.

Dica: Abaixo temos uma implementação **incompleta** do Quick Sort. Você deverá completá-la (e não refazer o método).

```
int Rearranja(int V[], int e, int d, int pivo, int U[])
{
        int j = e, k = d;

        swap(V[pivo], V[d]);
        pivo = d;

        for(int i = e; i <= d; i++)
                if(V[i] <= V[pivo])
                        U[j++] = V[i];
                else
                        U[k--] = V[i];

        for(int i = e; i <= d; i++)
                V[i] = U[i];

        return j-1;
}

void QuickSort(int V[], int e, int d, int U[])
{
        if(e >= d) return;

        int pivo = e;
        pivo = Rearranja(V, e, d, pivo, U);
        //Complete aqui
}
```

Entradas:

- Tamanho N do vetor.
- Os N elementos do vetor V.

Saídas:

- Vetor V ordenado.

Exemplo de Entrada:

```
5
7 8 5 3 6
```

Exemplo de saída:

```
3 5 6 7 8
```

Peso: 1

Última tentativa realizada em: 27/11/2019 00:01:48

Tentativas: 5 de 6

Nota (0 a 100): 80.6

Status ou Justificativa de Nota: O programa pode realizar uma repetição sem fim. A quantidade de dados escritos pelo programa é diferente da quantidade de dados esperados.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 13: Ordenação - QuickSort Soma dos não trocados

O **QuickSort** é um método de ordenação muito rápido e eficiente, que adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar os elementos de modo que os elementos "menores" precedam os elementos "maiores". Em seguida, o QuickSort ordena os dois subvetores de elementos menores e maiores recursivamente até que o vetor completo se encontre ordenado.

Abaixo está uma implementação do método QuickSort **com erros**. Sua missão é corrigi-lo para que ele funcione corretamente e também fazer uma modificação de forma que o algoritmo imprima todo o vetor resultante a cada iteração.

A impressão do vetor resultante deverá ser impresso antes das chamadas recursivas.

```
void quickSort(int v[], int esq, int dir) {
        int i, j, aux, pivo;
        i = esq;
        j = dir;
        pivo = v[(esq+dir)/2];
        while (i <= j) {
                while (v[i] < pivo) {
                        i++;
                }
                while (v[j] > pivo) {
                        j--;
                }
                if (i <= j) {
                        v[i] = v[j];
                        v[j] = v[i];
                        i++;
                        j--;
                }
        }

        if (esq < j) {
```

```
                quickSort(v, esq, j);
        }
        if (i < dir) {
                quickSort(v, i, dir);
        }
}
```

Entradas:

1. A quantidade de números a ser ordenada.
2. Os números inteiros positivos entre 1 e 1000 a serem ordenados.

Saídas:

1. A situação do vetor a cada iteração.

Exemplo de Entrada:

```
10
10 1 9 2 8 3 7 4 6 5
```

Exemplo de Saída:

```
5 1 6 2 4 3 7 8 9 10
2 1 6 5 4 3 7 8 9 10
1 2 6 5 4 3 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Exemplo de Entrada:

```
10
1 10 2 9 3 8 4 7 5 6
```

Exemplo de Saída:

```
1 3 2 9 10 8 4 7 5 6
1 2 3 9 10 8 4 7 5 6
1 2 3 9 10 8 4 7 5 6
1 2 3 4 10 8 9 7 5 6
1 2 3 4 6 8 5 7 9 10
1 2 3 4 6 7 5 8 9 10
1 2 3 4 6 5 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

Peso: 1

Última tentativa realizada em: 27/11/2019 00:54:52

Tentativas: 2 de 6

Nota (0 a 100): 35.5

Minutos Restantes:
?

Usuário:
Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Status ou Justificativa de Nota: A quantidade de dados escritos pelo programa é diferente da quantidade de dados esperados.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 14: Ordenação - Pontos de Vitória

Certo jogo de estratégia, com N jogadores, é jogado em volta de uma mesa. O primeiro a jogar é o jogador 1, o segundo a jogar é o jogador 2 e assim por diante. Uma vez completada uma rodada, novamente o jogador 1 faz sua jogada e a ordem dos jogadores se repete novamente. A cada jogada, um jogador garante uma certa quantidade de Pontos de Vitória. A pontuação de cada jogador consiste na soma dos Pontos de Vitória de cada uma das suas jogadas. Dado o número de jogadores, o número de rodadas e uma lista representando os Pontos de Vitória na ordem em que foram obtidos, você deve determinar e imprimir, em ordem decrescente de pontuação cada jogador. Caso mais de um jogador obtenha a pontuação máxima, o jogador com pontuação máxima que tiver jogado por último é o vencedor.

Entradas:

1. Número de jogadores
2. Número de rodadas
3. Pontos de vitórias de todos os jogadores em cada rodada

Saídas:

1. Jogadores em ordem decrescente de pontuação.

Exemplo de Entrada:

```
3
2
1 2 3
3 2 2
```

Exemplo de Saída:

```
3
2
1
```

baseado em: http://maratona.ime.usp.br/hist/2015/primeira-fase/maratona.pdf (Problema J)

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 15: Ordenação - Shell Sort

Shell Sort é uma generalização, e melhoria, do algoritmo de ordenação Insertion Sort, utilizando a ideia de um salto (gap) entre os elementos. Assim, começando com lacunas maiores, vai-se diminuindo o tamanho do gap, até permitir a comparação dos elementos um a um. O algoritmo pode ser expresso por 3 estruturas de repetição:

- Estrutura de repetição para definir o tamanho da lacuna (gap)
- Estrutura de repetição para pegar um dos valores do conjunto não ordenado como sendo o pivô para achar sua posição correta.
- Estrutura de repetição para comparar o pivô com os elementos localizados à sua esquerda no arranjo (respeitando-se a lacuna). Enquanto o subconjunto dos elementos à sua esquerda não acabar e eles forem maiores do que o pivô, move-se os maiores valores para a direita.

Faça uma implementação do Shell Sort.

Entradas:

1. Tamanho do vetor a ser ordenado
2. Sequência (vetor) de valores desordenados
3. Tamanho do vetor com as lacunas (gaps)
4. Sequência (vetor) de números representando o tamanho das lacunas (gaps) em ordem crescente.

Saídas:

1. Elementos do vetor ordenados.

Exemplo de Entrada:

```
9
3 4 9 2 5 1 8 0 11
9
1  4  10 23  57 132 301 701 1750
```

Exemplo de Saída:

```
0 1 2 3 4 5 8 9 11
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes:
?

Usuário:
Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 16: Ordenação - Corrigir o Merge Sort

Você deverá fazer um programa que recebe um vetor de inteiros V de N posições e o ordena. Para ordenar este vetor, você deverá usar o método **Merge Sort**.

Dica: Abaixo temos uma implementação **incorreta** do Merge Sort. Você deverá corrigí-la (e não refazer o método).

```
void Troca(int &A, int &B)
{
        int aux = A;
        A = B;
        B = aux;
}

//A função Merge abaixo está incorreta
void Merge(int V[], int p, int q, int r, int U[])
{
        int a = p;
        int b = q+1;

        for(int i = p; i <= q; i++)
                if( b > r || (a <= q && V[a] < V[b]) )
                        U[i] = V[a++];
                else
                        U[i] = V[b++];

        for(int i = p; i <= q; i++)
                V[i] = U[i];
}

void MergeSort(int V[], int primeira, int ultima, int U[])
{
        if(primeira >= ultima) return;

        int p = primeira;
        int r = ultima;
        int q = (p+r)/2;

        MergeSort(V, p, q, U);
        MergeSort(V, q+1, r, U);
        Merge(V, p, q, r, U);
}
```

Entradas:

- Tamanho N do vetor.
- Os N elementos do vetor V.

Saídas:

- Vetor V ordenado.

Exemplo de Entrada:

```
5
7 8 5 3 6
```

Exemplo de saída:

```
3 5 6 7 8
```

Peso: 1

Última tentativa realizada em: 27/11/2019 16:04:10

Tentativas: 1 de 6

Nota (0 a 100): 100

Status ou Justificativa de Nota: Nenhum erro encontrado.

Ver Código da Última Tentativa

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes: ?

Usuário: Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Questão 17: Ordenação - QuickSort Maiores e Menores

O **QuickSort** é um método de ordenação muito rápido e eficiente, que adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar os elementos de modo que os elementos "menores" precedam os elementos "maiores". Em seguida, o QuickSort ordena os dois subvetores de elementos menores e maiores recursivamente até que o vetor completo se encontre ordenado.

Abaixo está uma implementação correta do método QuickSort. Sua missão é alterá-lo para que ele exiba, a cada rearranjo, o maior e o menor elemento trocados (não necessariamente entre si). O algoritmo será usado apenas para números inteiros positivos entre 1 e 1000, e, caso não ocorra nenhuma troca em um rearranjo deve ser impresso -1 no lugar de cada número.

Observações:

1. É altamente recomendável que você utilize a implementação abaixo, pois outra estratégia poderia gerar os números em ordem diferente (ou até valores diferentes).
2. Um elemento ser trocado por ele mesmo (mesma posição) não deve ser considerado como uma troca.
3. O vetor final ordenado não deve ser exibido.

```
void quickSort(int v[], int esq, int dir) {
        int i, j, aux, pivo;
        i = esq;
        j = dir;
        pivo = v[(esq+dir)/2];
        while (i <= j) {
                while (v[i] < pivo) {
                        i++;
                }
                while (v[j] > pivo) {
                        j--;
                }
                if (i <= j) {
                        aux = v[i];
                        v[i] = v[j];
                        v[j] = aux;
                        i++;
                        j--;
                }
        }

        if (esq < j) {
                quickSort(v, esq, j);
        }
        if (i < dir) {
                quickSort(v, i, dir);
        }
}
```

Entradas:

1. A quantidade de números a ser ordenada.
2. Os números inteiros positivos entre 1 e 1000 a serem ordenados.

Saídas:

1. O maior e o menor elemento trocados a cada rearranjo realizado pelo método.

Exemplo de Entrada:

```
10
10 1 9 2 8 3 7 4 6 5
```

Exemplo de Saída:

```
10 4
5 2
2 1
```

```
6 3
-1 -1
-1 -1
-1 -1
```

Exemplo de Entrada:

```
10
1 10 2 9 3 8 4 7 5 6
```

Exemplo de Saída:

```
10 3
3 2
-1 -1
9 4
10 5
8 7
7 5
6 5
-1 -1
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Questão 18: Ordenação - Cutoff QuickSort.

Escreva uma versão do algoritmo QuickSort com cutoff para vetores pequenos: quando o vetor a ser ordenado tiver menos que M elementos(definido pelo usuário), a ordenação passa a ser feita por um algoritmo de seleção e para tamanhos maiores ou iguais a M, o programa deverá ordenar pelo método QuickSort.

Entradas:

1. Valor do M (int).
2. Número de elementos do vetor (int).
3. Elementos do vetor (int).

Saídas:

1. Vetor ordenado de maneira crescente.

Exemplo de Entrada:

```
10
5
32 4 1 55 3
```

Exemplo de Saída:

```
1 3 4 32 55
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo | Nenhum arquivo selecionado | Enviar Resposta

Questão 19: Ordenação - Corrigir o Merge Sort (passo a passo)

Faça um programa que lê a capacidade C de um vetor de inteiros V, preenche esse vetor e o ordena. Para ordenar este vetor, você deve **corrigir** o método **Merge** apresentado abaixo. Você deve corrigir o código apresentado e não refazê-lo. Além disso, você deve alterar o código para imprimir a situação do vetor a cada fase de intercalação.

```
//A função Merge abaixo está incorreta
void Merge(int V[], int p, int q, int r, int U[])
{
        int a = p;
        int b = q+1;

        for(int i = p; i <= q; i++)
                if( b > r || (a <= q && V[a] < V[b]) )
                        U[i] = V[a++];
                else
                        U[i] = V[b++];

        for(int i = p; i <= q; i++)
                V[i] = U[i];
}

void MergeSort(int V[], int primeira, int ultima, int U[])
{
        if(primeira >= ultima) return;

        int p = primeira;
        int r = ultima;
        int q = (p+r)/2;

        MergeSort(V, p, q, U);
```

```
        MergeSort(V, q+1, r, U);
        Merge(V, p, q, r, U);
}
```

Entradas:

- Tamanho N do vetor.
- Os N elementos do vetor V.

Saídas:

- Vetor V ordenado.

Exemplo de Entrada:

```
5
7 8 5 3 6
```

Exemplo de saída:

```
7 8
5 7 8
3 6
3 5 6 7 8
3 5 6 7 8
```

Exemplo de Entrada:

```
15
8 4 3 6 -9 7 -2 0 1 -5 10 -11 13 15 -12 14
```

Exemplo de saída:

```
4 8
3 6
3 4 6 8
-9 7
-2 0
-9 -2 0 7
-9 -2 0 3 4 6 7 8
-5 1
-11 10
-11 -5 1 10
13 15
-12 13 15
-12 -11 -5 1 10 13 15
-12 -11 -9 -5 -2 0 1 3 4 6 7 8 10 13 15
-12 -11 -9 -5 -2 0 1 3 4 6 7 8 10 13 15
```

Peso: 1

Nova Resposta:

Selecione o arquivo com o código fonte do programa que resolve o problema para enviá-lo.

Escolher arquivo Nenhum arquivo selecionado Enviar Resposta

Minutos Restantes:
?

Usuário:
Gabriel Nathan Almeida Silva

Notas:
Q1: 100
Q2: 100
Q3: 100
Q4: 100
Q5: 100
Q6: ?
Q7: 100
Q8: 100
Q9: 100
Q10: 100
Q11: 100
Q12: 80.6
Q13: 35.5
Q14: ?
Q15: ?
Q16: 100
Q17: ?
Q18: ?
Q19: ?
Total: 53

Desenvolvido por Bruno Schneider a partir do programa original (Algod) de Renato R. R. de Oliveira.