Iniciado em	sábado, 17 set 2022, 17:40
Estado	Finalizada
Concluída em	domingo, 18 set 2022, 19:20
Tempo empregado	1 dia 1 hora
Avaliar	**9,67** de um máximo de 10,00(**96,67**%)

Questão **1**

Parcialmente correto

Atingiu 0,67 de 1,00

Algoritmo de ordenação em computação é um algoritmo, de manipulação de dados, que coloca os elementos de uma dada sequência em uma certa ordem -- em outras palavras, efetua sua ordenação completa ou parcial. As ordens mais usadas são a numérica e a lexicográfica. Existem várias razões para se ordenar uma sequência. Uma delas é a possibilidade se acessar seus dados de modo mais eficiente. Com base no que foi aprendido sobre métodos de ordenação, selecione as alternativas que são verdadeiras.

- [ ] a. Os algoritmos Merge Sort, Quick Sort e Insertion Sort requerem a implementação de funções auxiliares para controle das chamadas recursivas.
- [ ] b. O Selection Sort possui um desempenho quase sempre superior quando comparado com o Quick Sort.
- [x] c. O Bubble Sort compara pares de elementos adjacentes e os troca de lugar se estiverem na ordem errada ✔
- [x] d. No algoritmo do Quick Sort, a metodologia na escolha do pivô interfere no desempenho do algoritmo. ✔
- [ ] e. O Heap Sort combina características de métodos simples - como o Insertion Sort, ao ordenar um elemento por vez; e de métodos mais robustos - ao explorar a estratégia de dividir para conquistar, reduzindo o problema original em subproblemas menores.
- [ ] f. O Heap Sort é um algoritmo de ordenação que usa uma estrutura de dados auxiliar chamada Heap, que é um tipo de árvore ternária.
- [x] g. O Bubble Sort é um método eficiente para ordenação de conjuntos pequenos. ✖
- [ ] h. Na prática, o Merge Sort realiza menos comparações que o Quick Sort.
- [x] i. O Selection Sort é um algoritmo que posiciona iterativamente o melhor elemento para ocupar uma posição do vetor. ✔
- [x] j. O Insertion Sort desloca um conjunto de opções para posicionar corretamente um valor dentro do vetor. ✔

Sua resposta está parcialmente correta.

Você selecionou corretamente 4.
As respostas corretas são:
O Bubble Sort compara pares de elementos adjacentes e os troca de lugar se estiverem na ordem errada, O Selection Sort é um algoritmo que posiciona iterativamente o melhor elemento para ocupar uma posição do vetor.,

O Insertion Sort desloca um conjunto de opções para posicionar corretamente um valor dentro do vetor.,

Na prática, o Merge Sort realiza menos comparações que o Quick Sort.,

No algoritmo do Quick Sort, a metodologia na escolha do pivô interfere no desempenho do algoritmo. ,

O Heap Sort combina características de métodos simples - como o Insertion Sort, ao ordenar um elemento por vez; e de métodos mais robustos - ao explorar a estratégia de dividir para conquistar, reduzindo o problema original em subproblemas menores.

Questão **2**

Completo

Atingiu 1,00 de 1,00

Um vetor contendo a sequência {23, 17, 14, 6, 13, 10, 1, 5, 7, 12} é um heap máximo? Justifique.

Para ser um Heap máximo a raiz precisa ser o primeiro elemento do vetor,
no nosso caso o 23 é o maior elemento e está no índice zero inferindo que ele é a
raiz, mas olhando para nossa árvore ao todo V[4] viola a propriedade de heap máximo,
pois um de seus filhos é o 7, um numero maior que ele mesmo, então podemos concluir
que essa sequência não é um heap máximo, caso a sequência fosse essa descrita abaixo:
{23, 17, 14, 7, 13, 10, 1, 5, 6, 12}, com essa alteração ai sim seria um heap máximo.

Comentário:

Questão **3**

Completo

Atingiu 2,00 de 2,00

O algoritmo **Quick Sort** é um método de ordenação muito rápido e eficiente, inventado por C.A.R. Hoare em 1960. Naquela época, Hoare trabalhou em um projeto de tradução de máquina para o National Physical Laboratory. Ele criou o **Quick Sort** ao tentar traduzir um dicionário de inglês para russo, ordenando as palavras, tendo como objetivo reduzir o problema original em subproblemas que possam ser resolvidos mais fácil e rápido. Assim sendo, analise o código do Quick Sort apresentado abaixo, e insira/corrija o(s) comando(s) necessário(s) para que o funcionamento do algoritmo seja correto.

```
1  void quickSort(int *vetor, int inicio) {
2    if(fim < inicio) {
3      pivo = criaParticoes(vetor, inicio, fim);
4      quickSort(vetor, inicio, pivo-1);
5      quickSort(vetor, pivo, fim);
6    }
7  }
8
9  int criaParticoes(int *vetor, int inicio, int fim) {
10
11   int esquerda = fim;
12   int direita  = inicio;
13   int pivo     = vetor[inicio];
14
15   while(esquerda < direita) {
16     while(vetor[esquerda] <= vetor[pivo]) {
17       esquerda+;
18     }
19
20     if(esquerda < direita) {
21       swap(&vetor[esquerda], &vetor[direita]);
22     }
23   }
24   vetor[inicio]  = v[direita];
25   vetor[direita] = pivo;
26   retorna(pivo);
27 }
28
```

Obs: Realize o upload do código corrigido (em C) e insira casos de teste na main mostrando que o programa funciona. Adicione todo seu código em um único arquivo .c para que possa ser executado e conferido pelo professor.

```
#include <stdio.h>



void quickSort(int *vetor, int inicio,int fim)
{
   int pivo;
   if (fim > inicio)
   {
      pivo = criaParticoes(vetor, inicio, fim);
      quickSort(vetor, inicio, pivo - 1);
      quickSort(vetor, pivo+1, fim);
   }
}


void swap(int *v1, int *v2)
{
   int temp = *v1;
```

```
    *v1 = *v2;
    *v2 = temp;
}

int criaParticoes(int *vetor, int inicio, int fim)
{

    int esquerda = inicio;
    int direita = fim;
    int pivo = vetor[inicio];

    while (esquerda < direita)
    {
        while (vetor[esquerda] <= pivo && esquerda <= fim)
        {
            esquerda++;
        }
        while (vetor[direita] > pivo && direita >= inicio)
        {
            direita--;
        }
        if (esquerda < direita)
        {
            swap(&vetor[esquerda], &vetor[direita]);
        }
    }
        swap(&vetor[inicio], &vetor[direita]);
    return direita;
}

int main(int argc, char const *argv[])
{
  //teste

  int vetor1[] = {9,8,4,6,5,7,3,2,1};
  quickSort(vetor1, 0, sizeof(&vetor1));

    for(int i = 0; i <= sizeof(&vetor1); i++) {
    printf("%d ", vetor1[i]);
  }printf("\n");



    return 0;
}
```

codigo.c

Comentário:

Questão **4**

Completo

Atingiu 3,00 de 3,00

Acesse a planilha abaixo e execute passo a passo o algoritmo de Heap Sort para ordenação a sua sequência.

Link: https://docs.google.com/spreadsheets/d/1LjPEQXK5nIluE4u0ho1gdvi61ZpSsuJCi-ONeiwfF-g/edit?usp=sharing

Adicione na resposta um documento externo (.pdf, .doc, .png) com **TODOS** os detalhes de **TODAS** as iterações necessárias para ordenar o vetor fornecido como input. Lembrem-se de: criar o Heap antes de começar o processo iterativo, e recriar o heap após cada iteração.

P1 Exercicio Madu.pdf

Comentário:

Questão **5**

Completo

Atingiu 1,00 de 1,00

Você tem que desenvolver um algoritmo para ordenar números que estão vindo por uma rede de computadores. Como bom programador, você não quer esperar todos os dados chegarem para começar o trabalho de ordenação. Dos seis algoritmos vistos em sala qual seria mais adequado para esse caso? Justifique explicando com suas palavras como esse algoritmo consegue realizar esse trabalho nesse cenário.

```
Pensando na quantidade de dados, como teremos um bom conjunto de dados já podemos
descartar os métodos Bubble, Selection. O heap acredito que não seria
um método fácil visto que teríamos várias ordenações o metodo precisaria esperar todos
os dados chegarem pra só depois rodar, o que não seria algo util, ir esperando os dados
chegarem e ordenando deixando sempre no Heap-maximo para quando todos os dados chegarem
você fazer a ordenação.
O método Insertion seria uma ótima escolha por ser estável e capaz de ordenar em
tempo real, é muito eficiente para conjuntos pequenos o que pode atrapalhar
seu rendimento nesse caso se tivermos um conjunto grande de dados e um fluxo grande
de entradas.
O método Quick eu não diria ser uma boa escolha visto que a cada divisão ele já tem
um valor do vetor ordenado, pensando nisso caso chegue mais dados a ordenação poderia
mudar podendo assim ter que refatorar novamente toda a ordenação e ter que "rodar"
o código a cada entrada de dados.
Para mim o método que melhor realizaria esse trabalho fora o convencional
```

Comentário:
Só o Insertion já seria suficiente aqui :)

Questão **6**

Correto

Atingiu 1,00 de 1,00

Ainda sobre os algoritmos de ordenação, relacione cada método com seu princípio de funcionamento:

Selection Sort	Ordenação por seleção. A complexidade computacional do caso médio é O(N^2) ✔
Quick Sort	Ordenação por particionamento. A complexidade computacional do caso médio é O(N log N) ✔
Bubble Sort	Ordenação por borbulhamento. A complexidade computacional do caso médio é O(N^2) ✔
Merge Sort	Ordenação por intercalação. A complexidade computacional do caso médio é O(N log N) ✔
Insertion Sort	Ordenação por inserção. A complexidade computacional do caso médio é O(N^2) ✔
Heap Sort	Ordenação por heap. A complexidade computacional do caso médio é O(N log N) ✔

Sua resposta está correta.

A resposta correta é:
Selection Sort → Ordenação por seleção. A complexidade computacional do caso médio é O(N^2),

Quick Sort → Ordenação por particionamento. A complexidade computacional do caso médio é O(N log N),

Bubble Sort → Ordenação por borbulhamento. A complexidade computacional do caso médio é O(N^2),

Merge Sort → Ordenação por intercalação. A complexidade computacional do caso médio é O(N log N),

Insertion Sort → Ordenação por inserção. A complexidade computacional do caso médio é O(N^2),

Heap Sort → Ordenação por heap. A complexidade computacional do caso médio é O(N log N).

Questão **7**

Correto

Atingiu 1,00 de 1,00

O ***merge sort***, ou ordenação por mistura ou intercalação, é um exemplo de algoritmo de ordenação por comparação do tipo dividir para conquistar. Sua ideia básica consiste em **dividir** (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e **conquistar** (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas). Como o algoritmo usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas.

Dessa forma, supondo que apresentamos como entrada para o algoritmo o seguinte vetor:

$$\mathbf{v} = \{45, 36, 2, 8, 0, 1, 23, 7, 5, 10\}$$

ordene as chamadas recursivas realizadas pelo algoritmo, exatamente como ele resolve o problema de ordenação.

- ✔ mergeSort({45, 36, 2, 8, 0, 1, 23, 7, 5, 10})
- ✔ mergeSort({45, 36, 2, 8, 0})
- ✔ mergeSort({45, 36, 2})
- ✔ mergeSort({45, 36})
- ✔ mergeSort({45})
- ✔ mergeSort({36})
- ✔ mergeSort({2})
- ✔ mergeSort({8, 0})
- ✔ mergeSort({8})
- ✔ mergeSort({0})
- ✔ mergeSort({1, 23, 7, 5, 10})
- ✔ mergeSort({1, 23, 7})
- ✔ mergeSort({1, 23})
- ✔ mergeSort({1})
- ✔ mergeSort({23})
- ✔ mergeSort({7})
- ✔ mergeSort({5, 10})
- ✔ mergeSort({5})
- ✔ mergeSort({10})

Sua resposta está correta.

◄ Submissão da Atividade 03

Seguir para...

P2 - 2/2022 ►