1 Trabalho

O trabalho consiste em duas etapas: a) entrega da resolução dos exercícios e b) defesa do trabalho no laboratório.

O trabalho é individual e deverá ser entregue até o dia 28 de julho de 2008, através do e-mail tiagodemelo@gmail.com. A defesa do trabalho ocorrerá no dia 30 de julho de 2008, no horário da aula. O trabalho entregue consiste num arquivo texto com as respostas e demais arquivos com as implementações.

Existem dois tipos de trabalho: A e B. O sorteio do trabalho acontecerá em sala de aula, no dia 14 de julho de 2008. Cada aluno ficará responsável por um único tipo de trabalho. O valor de cada questão é descrito no próprio enunciado.

Para cada dia de atraso no envio do trabalho acarretará no desconto de um ponto da nota final do aluno. As dúvidas, em tempo hábil, poderão ser sanadas com o professor.

2 Tipo A

1. Dado um vetor com n elementos do tipo: [valor 2,0]

   ```
   typedef struct Cliente {
       char *nome;
       float saldo;
   } Cliente;
   ```

   Escreva uma função em C que ordene este vetor em ordem ascendente pelo campo saldo, de tal forma que a lista de clientes com o mesmo saldo fique em ordem alfabética ascendente. Utilize a lógica do algoritmo da Bolha, fazendo as modificações que julgar necessárias.

2. Muitas vezes o vetor já está ordenado antes da execução do número total de passos de ordenação. Se o vetor já estiver ordenado, não há necessidade de continuar executando o algoritmo. Crie um programa em C que implemente o método Seleção Direta com estas condições, ou seja, o programa deve parar de executar assim que o vetor estiver ordenado. Se o vetor já estiver ordenado, no início da execução, não deve ser realizada nenhuma operação de troca de elementos, ou seja, não é necessário executar o método de ordenação Seleção Direta. [valor 2,0]

3. Suponha que você deseja implementar um programa que realiza diversas tarefas dentre as quais a ordenação de vários itens de dados que contêm diversas informações. Cada item ocupa cerca de 1kbyte de memória. Considerando cada uma das situações abaixo, indique (e justifique) qual o método de ordenação mais indicado em cada situação: [valor 1,0]

   - Se os dados na entrada já estão quase ordenados.
   - Se o objetivo é minimizar a quantidade de movimentos de itens.
   - Se o objetivo é otimizar o tempo necessário para se ordenar os itens.
   - Se a posição original dos itens não pode ser alterada.

4. No caso do algoritmo *Shell Sort*, suponha que este algoritmo foi modificado de modo que os valores do incremento h sejam tais que o último valor utilizado seja $h = 2$. Neste caso, o algoritmo produz o resultado correto? Justifique a sua resposta. [valor 1,0]

5. Modifique o programa Quicksort apresentado em sala de aula de modo que, se um subvetor for pequeno, a classificação pelo método Bolha seja empregada. Determine, usando execuções reais de programas de computador, quão pequeno o subvetor deve ser para que essa estratégia cominada seja mais eficiente do que a implementação comum do Quicksort. [valor 2,0]

6. Considere uma seqüência com um milhão de elementos dispostos de forma ordenada, invertida e desordenada. Preencha a tabela a seguir com os tempos gastos pelas implementações dos diferentes algoritmos de ordenação para estabelecer uma ordem ascendente para elementos daquela seqüência. [valor 2,0]

	Tempo gasto quando		
	Invertida	Desordenada	Ordenada
Bolha			
Inserção			
Seleção			
ShellSort			
QuickSort			
HeapSort			

O trecho de código a seguir sugere como o tempo gasto por cada implementação pode ser calculado. Utilize as implementações dos métodos de ordenação apresentados em sala de aula.

```
#include "time.h"
time_t tempo0, tempo1;
time(&tempo0); /*obtem o tempo antes do inicio da função */
ChamadaDaFuncao();
time(&tempo1); /*obtem o tempo antes do final da função */
tempoDecorridoEmSegundo = difftime(tempo1, tempo0);
```

3 Tipo B

1. Dado um vetor com n elementos do tipo: [valor 2,0]

```
typedef struct Cliente {
    char *nome;
    float saldo;
} Cliente;
```

   Elabore um programa que apresente um menu ao usuário oferecendo a possibilidade de ordenar um vetor pelo nome ou pelo saldo do cliente, utilizando a estrutura Cliente. Após a resposta do usuário, o programa deve perguntar o tamanho e os valores desse vetor.

   A resposta do programa deve ser o vetor recebido, porém ordenado. A ordenação do *nome* deve desconsiderar as diferenças entre maiúsculas e minúsculas.

2. Invente um vetor-exemplo de entrada para demonstrar que a ordenação através do método de Seleção é um método instável. Mostre os passos da execução do algoritmo até que a estabilidade seja violada. Note que quanto menor for o vetor que você inventar, mais rápido você vai resolver a questão. [valor 1,0]

3. Existe alguma semelhança do HeapSort com a ordenação por Inserção Direta? se sua resposta for "sim", explique qual. Se sua resposta for "não", justifique-a. [valor 1,0]

4. Faça uma nova implementação do método Shellsort considerando um forma diferente, da apresentada em sala de aula, para gerar os valores de h. Compare o método implementado por você com o programa apresentado em sala de aula. Utilize um tamanho de vetor que seja suficiente para demonstrar a diferença de desempenho. [valor 2,0]

5. Escreva um programa em C para remover o menor elemento do heap. Após a remoção, a estrutura de heap deve ser mantida. [valor 2,0]

6. Considere uma seqüência com um milhão de elementos dispostos de forma ordenada, invertida e desordenada. Preencha a tabela a seguir com os tempos gastos pelas implementações dos diferentes algoritmos de ordenação para estabelecer uma ordem ascendente para elementos daquela seqüência. [valor 2,0]

	Tempo gasto quando		
	Invertida	Desordenada	Ordenada
Bolha			
Inserção			
Seleção			
ShellSort			
QuickSort			
HeapSort			

O trecho de código a seguir sugere como o tempo gasto por cada implementação pode ser calculado.

```
#include "time.h"
time_t tempo0, tempo1;
time(&tempo0); /*obtem o tempo antes do inicio da função */
ChamadaDaFuncao();
time(&tempo1); /*obtem o tempo antes do final da função */
tempoDecorridoEmSegundo = difftime(tempo1, tempo0);
```