1. [**2** pontos] Considere a seguinte função de hashing:

```
int hash(char *v, int M)
  { int h = 0, a = 128;
    for (; *v != '\0'; v++) h = (a*h + *v) % M;
    return h;
  }
```

Suponha que o tamanho `M` da tabela que está sendo usada é `1024`. O usuário percebe que muitas colisões ocorrem quando esta função de hashing é utilizada para processar as palavras que ocorrem em um dado texto (isto é, o espalhamento esperado não ocorre). Você teria uma explicação para este fenômeno? Em particular, quando ocorre desta função devolver o mesmo valor para chaves distintas?

2. [**2** pontos] Escreva uma função que rearranja os elementos de uma lista dada, pondo todos os elementos que ocorrem nas posições pares antes de todos os elementos de ocorrem nas posições ímpares. Você deve preservar a ordem relativa dos elementos pares entre si, e a ordem relativa dos elementos ímpares entre si.

3. [O problema das panquecas; **3** pontos] Suponha que temos, por exemplo, a seqüencia de números

```
8 2 3 5 1 4 9 7 6 0
```

Imagine que esta seqüência indica uma pilha de panquecas de 10 tamanhos distintos, empilhadas de forma que a menor está no fundo da pilha, a maior é a quarta panqueca a partir do fundo da pilha, a segunda maior está no topo, etc. As únicas operações permitidas são inverter um segmento inicial desta seqüência. Portanto, existem 9 operações não-triviais para a pilha acima; uma delas produz, por exemplo,

```
5 3 2 8 1 4 9 7 6 0
```

Podemos chamar esta operação deste exemplo de 'operação número 4', já que invertemos a ordem das 4 panquecas do topo. O problema é o seguinte: dados um inteiro positivo `n` e uma permutação dos inteiros não-negativos menores do que `n`, determinar uma seqüência de operações (permitidas) que resultam na seqüência

```
0 1 2 ... n - 1
```

Isto é, com as panquecas em ordem crescente, do topo para o fundo. Escreva uma função de protótipo

```
void imprima_solucao(int n, int panq[]);
```

para resolver este problema.

3. [**3** pontos] Descreva cuidadosamente o projeto de um programa que recebe uma cadeia de caracterers T como entrada e devolve o trecho repetido mais comprido neste texto. Por exemplo, se o texto de entrada for

```
aacabcabcbabcbabac
```

então a saída de seu programa deve ser `abcbab`.

O seu projeto deve supor que a entrada T pode ser grande; por exemplo, T poderia um livro todo. Um programa implementado de acordo com o seu projeto deve ser tal que,

(a) sob hipóteses razoáveis, ele leva tempo basicamente proporcional a $n \log n$, onde n é o número de caracteres em T,

(b) ele gasta uma quantidade de memória proporcional ao número de caracteres em T.

Você deve argumentar por que o programa teria estas propriedades (em particular, você deve explicitar as hipóteses que você usa em sua análise). [*Sugestão*. Um *sufixo* de uma cadeia de caracteres T é um 'segmento final' dela. Os sufixos de `abcde` são

```
abcde
bcde
cde
de
e
```

e a cadeia vazia, com 0 caracteres (6 sufixos no total). Considere os sufixos de T.]