Pesquisa e Classificação de Dados - Prova 1 - C

2020/1 - ADNP – Prof. Ricardo Oliveira

Instruções: Envie um pacote (**zip** ou **tar.gz**) contendo:

- Para as questões 1, 2, 3, 4, 5 (itens (d) e (e)), 7 e 8, arquivos legíveis (como arquivos de texto, PDFs, imagens/fotos, etc) com suas respostas;
- Para as questões 5 (itens (a) a (c)) e 6, dois programas em C (`.c`), um para cada questão.

Certifique-se de estar resolvendo a versão correta da prova, de acordo com seu RA. Envie um pacote **zip** ou **tar.gz**; pacotes em outros formatos que não puderem ser abertos pelo professor não serão corrigidos. Respostas detectadas como plágios ou cópias receberão nota zero.

---

1. (5 pontos) Prove que $3N^2 - 8 = \Theta(N^2)$.
2. (5 pontos) Prove que $\frac{N}{8} \notin O(1)$.
3. (20 pontos) Determine a complexidade de tempo de pior caso do algoritmo recursivo abaixo:

```
int funcao(int N) {
    int i, x;
    if (N <= 1)
        return 0;
    i=0;
    x=1;
    while (x <= N) {
        i = i + x;
        x = 2*x;
    }
    return x + funcao(N/2);
}
```

4. (15 pontos) O algoritmo abaixo computa a soma dos elementos inteiros positivos de um vetor v de tamanho N. Determine a complexidade (de tempo) de *caso médio* deste algoritmo. Considere que cada posição do vetor contém um inteiro entre 1 e 100 (inclusive) com a mesma probabilidade.

```
soma = 0
Para i = 0 a N-1 faça:
    Para j = 0 a v[i]-1 faça:
       soma = soma + 1
 retorne soma
```

5. (20 pontos) Seja M a posição do maior elemento de um vetor v com N elementos distintos. Um vetor *topzeira* é um vetor no qual o subvetor $v[0..M-1]$ (isto é, à esquerda da posição M) é crescente, e o subvetor $v[M+1..N-1]$ (isto é, à direita da posição M) é decrescente. Como exemplo, os vetores $[12, 21, 30, 42, 29, 18]$ e $[3, 2, 1]$ são topzeira, mas $[1, 3, 2, 5, 4]$ não é.

   Escreva um programa em C que:

   (a) lê do usuário o tamanho N e o vetor v. Considere que o usuário sempre irá entrar com um vetor topzeira de no máximo 100 elementos;

   (b) utilizando o algoritmo que preferir, determina a posição M do maior elemento do vetor;

   (c) lê do usuário um inteiro x e determina se x ocorre no vetor, *de forma que o pior caso deste passo tenha tempo* $O(\lg N)$.

Considere o seguinte exemplo de execução (o sinal `>` indica entrada do usuário):

```
Tamanho do vetor> 6
Vetor> 12 21 30 42 29 18
Posicao M = 3
Valor a buscar> 25
25 nao esta no vetor dado
```

(d) Analize a complexidade de tempo de *melhor* e de *pior* caso do algoritmo usado no passo (b);

(e) Analize a complexidade de tempo de *melhor* e de *pior* caso do algoritmo usado no passo (c).

6. (15 pontos) A última página desta prova contém um programa em C que faz a seguinte operação, primeiramente para $N = 50000$ (cinquenta mil) e depois para $N = 100000$ (cem mil):

   - gera aleatoriamente dois vetores *iguais* `v` e `w` de tamanho N;
   - ordena `v` com o *selectionSort*, e `w` com o *mergeSort*;
   - imprime para o usuário o tempo (em segundos) levado em cada ordenação.

   Ao final, o programa imprime para cada método sua *taxa de crescimento*: razão entre o tempo levado para ordenar o vetor com cem mil elementos e o tempo levado para ordenar o vetor com cinquenta mil elementos.

   (a) Implemente o *selectionSort* na função `void selectionSort(int *v, int N)`;

   (b) Implemente o *mergeSort* na função `void mergeSort(int *v, int ini, int fim)` (onde `ini` e `fim` são as posições de início e fim do vetor, respectivamente);

   (c) Analisando a taxa obtida em cada método, justifique por que um método foi pior/equivalente/melhor que outro, e descreva como você poderia deduzir (aproximadamente) a taxa obtida de cada método de maneira não empírica (sem precisar implementar e executar o método na sua máquina). Imprima sua resposta na própria saída do programa (na linha `printf("[RESPOSTA ITEM C]\n");`).

   OBS: Você pode, se quiser, criar outras funções além das já existentes no programa. Entregue na tarefa o programa completo (e não apenas as funções que você implementou).

7. (15 pontos)

   (a) Escreva em pseudo-código um algoritmo que, dado um vetor de N inteiros, faz um particionamento *estável* para o *QuickSort*, isto é, seu algoritmo deve escolher um pivô e particionar o vetor de forma a manter a ordem relativa dos elementos considerados iguais. Escreva apenas o algoritmo de particionamento (e não o *QuickSort* inteiro). Você pode criar o algoritmo como preferir, independente da sua complexidade ou uso de memória. Entretanto, **não** simplesmente chame um método estável de ordenação. Seja criativo(a)! Também não assuma a existência de nenhuma função já pronta.

   (b) Analise a complexidade de tempo de pior caso para o algoritmo que você criou no item anterior.

8. (5 pontos) Assista o video https://youtu.be/9zb5QTOZ1v4 . Qual é este método de ordenação? Explique informalmente, com suas palavras, o que lhe levou a reconhecer o método.

```c
/* PROGRAMA PARA A QUESTAO 6 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void troca(int *a, int *b) {
    int c = *a;
    *a = *b;
    *b = c;
}

void gera(int *v, int *w, int N) {
    int i;
    for (i=0;i<N;i++)
        v[i] = w[i] = rand()%1000000 + 1;
}

void selectionSort(int *v, int N) {
    /* ... */
}

void mergeSort(int *v, int ini, int fim) {
    /* ... */
}

int main() {

    int *v, *w;
    int tam[2] = {50000, 100000}, i, N;
    double comeco, fim, tempo[2][2];

    srand(time(NULL));

    for (i=0;i<2;i++) {
        N = tam[i];

        v = (int *)malloc(N*sizeof(int));
        w = (int *)malloc(N*sizeof(int));
        gera(v, w, N);

        comeco = ((double)clock())/CLOCKS_PER_SEC;
        selectionSort(v, N);
        fim = ((double)clock())/CLOCKS_PER_SEC;
        tempo[i][0] = fim - comeco;
        printf("N = %d, selectionSort: %.4lf segundos\n", N, tempo[i][0]);

        comeco = ((double)clock())/CLOCKS_PER_SEC;
        mergeSort(w, 0, N-1);
        fim = ((double)clock())/CLOCKS_PER_SEC;
        tempo[i][1] = fim - comeco;
        printf("N = %d, mergeSort: %.4lf segundos\n\n", N, tempo[i][1]);

        free(v);
        free(w);

    }

    printf("Taxa selectionSort: %.2lf vezes\n", tempo[1][0]/tempo[0][0]);
    printf("Taxa mergeSort: %.2lf vezes\n\n", tempo[1][1]/tempo[0][1]);

    printf("[RESPOSTA ITEM C]\n");

    return 0;
}
```