MAC0122 – Princípios de Desenvolvimento de Algoritmos

Engenharia de Computação

Segundo Semestre de 2023

Prova Substitutiva – 20/12/2023

Nome completo: _____

NUSP: _____

Assinatura: _____

Instruções:

1. Não destaque as folhas deste caderno.
2. Preencha o cabeçalho acima.
3. A prova pode ser feita a lápis. Cuidado com a legibilidade.
4. A prova consta de **3 questões**. Verifique antes de começar a prova se o seu caderno de questões está completo. **A prova vale 11 pontos.**
5. Não é permitido o uso de folhas avulsas para rascunho.
6. Não é permitido o uso de artefatos eletrônicos.
7. Não é permitida a consulta a livros, apontamentos ou colegas.
8. Não é necessário apagar rascunhos no caderno de questões.

DURAÇÃO DA PROVA: 2 horas

Questão	Nota
1	
2	
3	
Total	

Q1 (4.0 pontos) Diga qual será a saída do programa abaixo quando executado com os dígitos de seu número USP, dados um a um, separado por espaços, seguido de `1 2 2`. Por exemplo, se seu NUSP *fosse* 31415926, você teria de simular o programa com entrada

```
3 1 4 1 5 9 2 6 1 2 2
```

Lembre que seu NUSP *não é* 31415926 `:-)`.

Importante: seu rascunho deve dar alguma indicação de como você chegou a sua resposta.

```
public class Q1
{
    private Stack<Integer>[] piles;
    private Queue<Integer>[] indices;

    Q1(int[] s) {
        int N = s.length;
        int p = 0;
        Stack<Integer>[] dec = (Stack<Integer>[]) new Stack[N];
        Queue<Integer>[] ind = (Queue<Integer>[]) new Queue[N];

        for (int i = 0; i < N; i++) {
            int j = pile(s[i], dec, p);
            if (j == p) {
                dec[p] = new Stack<>();
                ind[p++] = new Queue<>();
            }
            dec[j].push(s[i]);
            ind[j].enqueue(i);
        }
        piles = (Stack<Integer>[]) new Stack[p];
        for (int i = 0; i < p; i++)
            piles[i] = dec[i];
        indices = (Queue<Integer>[]) new Queue[p];
        for (int i = 0; i < p; i++)
            indices[i] = ind[i];
    }

    private static int pile(int x, Stack<Integer>[] dec, int p) {
        int i = 0;
        while (i < p && dec[i].peek() < x)
            i++;
        return i;
    }

    public void show() {
        int p = piles.length;
        for (int i = 0; i < p; i++) {
            Stack<Integer> t = new Stack<Integer>();
            for (int x : piles[i])
                t.push(x);
            StdOut.printf("%2d: ", i);
            for (int j = 0; j < piles[i].size(); j++)
                StdOut.print(t.pop() + " (" + indices[i].dequeue()+ ") ");
            StdOut.println();
        }
    }
```

```
    public static void main(String[] args)
    {
        int[] s = StdIn.readAllInts();
        Q1 q = new Q1(s);
        q.show();
    }
}
```

DICA. O seguinte trecho de código

```
    Stack<Integer> s = new Stack<>();
    s.push(0);
    s.push(1);
    s.push(2);
    for (int x : s)
        StdOut.print(x + " ");
```

imprime

```
2 1 0
```

Rascunho

Rascunho

Saída do programa

Q2 (3.0 pontos) Considere as seguintes funções:

```
private static void merge(int[] a, int[] aux, int lo, int mid, int hi) {
    // prDots(hi - lo);
    int i = lo, j = mid;
    for (int k = lo; k < hi; k++) {
        if      (i == mid)    aux[k] = a[j++];
        else if (j == hi)     aux[k] = a[i++];
        else if (a[j] < a[i]) aux[k] = a[j++];
        else                  aux[k] = a[i++];
    }

    for (int k = lo; k < hi; k++)
        a[k] = aux[k];
}

public static void sort(int[] a, int[] aux, int lo, int hi) {

    if (hi - lo <= 1) return;

    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid, hi);

    merge(a, aux, lo, mid, hi);
}

public static void sort(int[] a) {
    int n = a.length;
    int[] aux = new int[n];
    sort(a, aux, 0, n);
}

private static void prDots(int t) {
    for (int i = 0; i < t; i++)
        StdOut.print(".");
    StdOut.println();
}
```

Se `a` é um vetor de `int`, a chamada `sort(a)` ordena `a`. Vale que o tempo de execução dessa chamada é basicamente proporcional ao número número total de caracteres `'.'` impressos por essa chamada quando descomentamos a linha

```
// prDots(hi - lo);
```

em `merge()`.

(a) Suponha que executamos `sort(a)` com `a` com N inteiros. Suponha que descomentamos a linha acima de forma que caracteres `'.'` são impressos. Quantos caracteres `'.'` são impressos no total quando $N = 1$, 2, 4 e 8? Neste item, basta dizer cada um desses 4 valores (não é necessário justificar).

(b) Repita o item (a) para $N = 16$. Desta vez, justifique sua resposta por completo.

(c) Seja $P(N)$ o número de caracteres `'.'` impressos pela chamada `sort(a)` quando `a` tem N inteiros. Dê uma fórmula exata para $P(N)$ no caso em que N é uma potência de 2. Justifique sua resposta, possivelmente usando sua resposta para o item (b) acima como base/exemplo.

(d) Seja $T(N)$ o tempo de execução de `sort(a)` para `a` com N inteiros. Com base no item anterior, qual é a ordem de grandeza/crescimento de $T(N)$ (não é necessário justificar sua resposta).

Q3 (4.0 pontos) Escreva um programa chamado `Q3.java` como especificado a seguir. Seu programa `Q3.java` deve receber como argumentos de linha de comando um inteiro positivo `N` e um strings `padrao` de 0s e 1s não-vazio. Com tais dados, `Q3.java` deve ter como saída todos os strings de 0s e 1s de comprimento `N` **que não contêm** `padrao` **como segmento** (isto é, não são da forma `s + padrao + t`, onde `s` e `t` são strings quaisquer, e `+` indica concatenação). Seguem alguns exemplos de execução:

```
$ java-introcs Q3 3 11
000
001
010
100
101
$ java-introcs Q3 4 10
0000
0001
0011
0111
1111
$ java-introcs Q3 5 001
00000
01000
01010
01011
01100
01101
01110
01111
10000
```

```
10100
10101
10110
10111
11000
11010
11011
11100
11101
11110
11111
$ java-introcs Q3 2 0
11
$ java-introcs Q3 3 0
111
$
```

No primeiro exemplo, `N` vale 3 e `padrao` é `11`. Assim, a saída é a lista de todos os 5 strings de 0s e 1s sem dois 1s consecutivos.

Observação. O tempo de execução de seu programa `Q3.java` deve ser basicamente proporcional ao "tamanho" de sua saída. Em particular, a solução que gera todos os 2^N strings de 0s e 1s de comprimento N e que imprime apenas aqueles strings válidos não satisfaz esta exigência desta questão.

(a) Escreva inicialmente uma função de assinatura

```
public static boolean isSuffix(String f, String p)
```

que recebe dois strings `f` e `p` e que devolve `true` se e só se `f` é um sufixo de `p`, isto é, o string `p` é da forma `s + f` para algum string `s` (e `+` indica concatenação).

(b) Escreva aqui seu programa `Q3.java`. Seu programa `Q3.java` deve **obrigatoriamente** usar a função `isSuffix()` do item anterior (mesmo que você não tenha feito aquele item). Não é necessário reescrever `isSuffix()` aqui.

* * *

(Rascunho)