MAC0121 – Algoritmos e Estruturas de Dados I

Bacharelado em Ciência da Computação

Segundo Semestre de 2023

Prova Substitutiva – 21/12/2023

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 12 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. Por exemplo, se seu NUSP *fosse* 31415926, você teria de simular o programa com entrada

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

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

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

```
public class Q1
{
    public static String expr(int[] seq) {
        return expr(seq, 0, seq.length, true);
    }

    public static String expr(int[] seq, int s, int t, boolean prod) {
        if (t == s + 1)
            return seq[s] + " ";
        int mid = s + (t - s) / 2;
        if (prod) {
            String e = "( " + expr(seq, s, mid, !prod)
                       + "* " + expr(seq, mid, t, !prod) + ") ";
            StdOut.println(e);
            return e;
        } else {
            String e = "( " + expr(seq, s, mid, !prod)
                       + "+ " + expr(seq, mid, t, !prod) + ") ";
            StdOut.println(e);
            return e;
        }
    }

    public static int val(int[] seq) {
        return val(seq, 0, seq.length, true);
    }

    public static int val(int[] seq, int s, int t, boolean prod) {
        if (t == s + 1)
            return seq[s];
        int mid = s + (t - s) / 2;
        if (prod)
            return val(seq, s, mid, !prod) * val(seq, mid, t, !prod);
        else
            return val(seq, s, mid, !prod) + val(seq, mid, t, !prod);
    }

    public static void main(String[] args)
    {
        int[] NUSP = StdIn.readAllInts();
        String expr = expr(NUSP);
        int val = val(NUSP);
        StdOut.println(val);
    }
}
```

Rascunho

Saída do programa

Q2 (4.0 pontos) Considere o seguinte programa:

```
import java.util.Arrays;

public class Q2
{
    public static long count(int[] a, int x) {
        long N = a.length;
        long M = N * (N - 1) / 2;
        return M - countL(a, x) - countS(a, x);
    }

    public static long countS(int[] a, int x) {
        return countS(a, 0, a.length - 1, x);
    }

    public static long countS(int[] a, int i, int j, int x) {
        if (i >= j) return 0;
        if (a[i] + a[j] >= x)
            return countS(a, i, j - 1, x);
        return j - i + countS(a, i + 1, j, x);
    }

    public static long countL(int[] a, int x) {
        return countL(a, 0, a.length - 1, x);
    }

    public static long countL(int[] a, int i, int j, int x) {
        if (i >= j) return 0;
        if (a[i] + a[j] <= x)
            return countL(a, i + 1, j, x);
        return j - i + countL(a, i, j - 1, x);
    }

    public static int[] randomSeq(int N, int M) {
        int[] a = new int[N];
        for (int i = 0; i < N; i++)
            a[i] = StdRandom.uniformInt(-M, M + 1);
        return a;
    }

    public static int[] sum(int[] a) {
        int N = a.length;
        int M = N * (N - 1) / 2;
        int[] sum = new int[M];
        int k = 0;
        for (int i = 0; i < N; i++)
            for (int j = i + 1; j < N; j++)
                sum[k++] = a[i] + a[j];
        Arrays.sort(sum);
        return sum;
    }

    public static void show(int[] a) {
        int N = a.length;
        for (int i = 0; i < N; i++)
            StdOut.printf("%3d ", a[i]);
        StdOut.println();
    }
```

```
    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        int M = Integer.parseInt(args[1]);
        long seed = Long.parseLong(args[2]);
        StdRandom.setSeed(seed);

        int[] a = randomSeq(N, M);
        Arrays.sort(a);
        int x = StdRandom.uniformInt(-2*M, 2*M + 1);
        StdOut.println(x + " occurs " + count(a, x) + " times in a + a");
        int[] sum = sum(a);
        StdOut.print("a:     ");
        show(a);
        StdOut.print("a + a: ");
        show(sum);
    }

}
```

Seguem alguns exemplos de execução:

```
$ java-introcs Q2 5 2 20231221
-2 occurs 3 times in a + a
a:      -2   0   0   0   1
a + a:  -2  -2  -2  -1   0   0   0   1   1   1
$ java-introcs Q2 5 1 121
2 occurs 3 times in a + a
a:      -1  -1   1   1   1
a + a:  -2   0   0   0   0   0   0   2   2   2
$ java-introcs Q2 1000 100 121 | head -n1
-48 occurs 1955 times in a + a
$ java-introcs Q2 2000 100 121 | head -n1
-16 occurs 9040 times in a + a
$ java-introcs Q2 5000 100 121 | head -n1
-113 occurs 26776 times in a + a
$
```

(`head -n1` faz com que todas as linhas após a primeira sejam ignoradas.)

A chamada `count(a, x)` determina o número de pares (i, j) tais que $i < j$ e $a[i] + a[j] = x$. Esta afirmação segue das seguintes duas afirmações:

(i) A chamada `countS(a, x)` determina o número de pares (i, j) tais que $i < j$ e $a[i] + a[j] < x$.

(ii) A chamada `countL(a, x)` determina o número de pares (i, j) tais que $i < j$ e $a[i] + a[j] > x$.

(a) Prove que a chamada `countS(a, i, j, x)` determina o número de pares (i', j') tais que $i \leq i' < j' \leq j$ tais que $a[i'] + a[j'] < x$.

(b) Deduza a afirmação (i) acima do item anterior.

(c) Prove que a chamada `countL(a, i, j, x)` determina o número de pares (i', j') tais que $i \leq i' < j' \leq j$ tais que $a[i'] + a[j'] > x$.

(d) Deduza a afirmação (ii) acima do item anterior.

(e) É fácil remover a recursão de `countS()` e `countL()`. Implemente abaixo `countS()` e `countL()` de forma não-recursiva (note que queremos substituir **quatro** funções do programa acima por **duas** funções).

Q3 (4.0 pontos) Considere o programa incompleto abaixo:

```
import java.util.Arrays;

public class Q3
{
    // Returns the number of occurrences of x in a.
    // Assumes a is non-decreasing.
    public static int occ(int x, int[] a) {
        int m = min(x, a);
        if (m < 0) return 0;
        int M = max(x, a);
        return M - m + 1;
    }

    // Returns the smallest i with a[i] = x if it exists
    // and returns -1 otherwise.  Assumes a is non-decreasing.
    public static int min(int x, int[] a) {
        int N = a.length;
        if (N == 0 || x > a[N - 1]) return -1;
        int i = min(x, a, 0, N - 1);
        if (a[i] == x)
            return i;
        return -1;
    }

    // Returns the smallest i with lo <= i <= hi such that x <= a[i].
    // Assumes a is non-decreasing and x <= a[hi].
    public static int min(int x, int[] a, int lo, int hi) {

        // a completar (item (a))

    }

    // Returns the largest i with a[i] = x if it exists
    // and returns -1 otherwise.  Assumes a is non-decreasing.
    public static int max(int x, int[] a) {
        int N = a.length;
        if (N == 0 || x < a[0]) return -1;
        int i = max(x, a, 0, N);
        if (a[i] == x)
            return i;
        return -1;
    }

    // Returns the largest i with lo <= i < hi such that a[i] <= x.
    // Assumes a is non-decreasing and a[lo] <= x.
    public static int max(int x, int[] a, int lo, int hi) {

        // a completar (item (b))

    }

    public static int occPlain(int x, int[] a) {
        int t = 0;
        for (int i = 0; i < a.length; i++)
            if (a[i] == x) t++;
        return t;
    }
```

```
    public static int[] randomSeq(int N, int M) {
        int[] a = new int[N];
        for (int i = 0; i < N; i++)
            a[i] = StdRandom.uniformInt(M);
        return a;
    }

    public static void main(String[] args)
    {
        int N = Integer.parseInt(args[0]);
        int M = Integer.parseInt(args[1]);
        long seed = Long.parseLong(args[2]);
        StdRandom.setSeed(seed);
        int[] a = randomSeq(N, M);
        int x = StdRandom.uniformInt(M);

        StdOut.println(x + " occurs " + occPlain(x, a) + " times");

        Arrays.sort(a);
        StdOut.println(x + " occurs " + occ(x, a) + " times");
    }
}
```

A chamada `occPlain(x, a)` devolve o número de vezes que `x` ocorre em `a`. A função `occPlain()` tem tempo de execução linear em N, onde N é o número de elementos em `a`. Queremos que `occ()` determine o mesmo valor que `occPlain()`, **mas com tempo de execução logarítmico em** N. Complete o programa acima para cumprir este objetivo. Note que basta você implementar versões eficientes de `min()` e `max()` como especificado no programa.

(a) Implemente aqui a função `min()`. Não deixe de escrever a assinatura (cabeçalho) da função por completo.

(b) Implemente aqui a função `max()`. Não deixe de escrever a assinatura (cabeçalho) da função por completo.

* * *

(Rascunho)