CCM0128 – Computação II

Curso de Ciências Moleculares

Primeiro Semestre de 2025

Prova 1 – 24/4/2025

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, separados por espaços, seguido de `2 7 1 8`. Por exemplo, se seu NUSP fosse 31415926, você teria de simular o programa com entrada

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

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

Importante: seu número USP não é 31415926. `:-)`

```
public class Q1
{
    // Sort array a making some number of passes.
    // In each pass, pairs of consecutive non-decreasing
    // runs are merged.
    public static void sort(int[] a) {
        int[] aux = new int[a.length];
        while (true) {
            int runs = 0;  // will count number of runs as we go along
            int s = 0;     // start of first run
            while (s < a.length) {
                int t = nextStart(a, s);
                runs++;
                if (t == a.length)
                    break;
                int tt = nextStart(a, t);
                merge(a, aux, s, t, tt);
                s = tt;
            }
            StdOut.print(runs + " run(s): "); // print and not println
            show(a);
            if (runs == 1) // now have only one run: array is sorted
                break;
        }
    }

    // finds the start of the next non-decreasing run
    private static int nextStart(int[] a, int s) {
        while (s + 1 < a.length && a[s] <= a[s + 1])
            s++;
        return s + 1;
    }
```

```
    private static void show(int[] a) {
        for (int i = 0; i < a.length; i++)
            StdOut.print(a[i] + " ");
        StdOut.println();
    }

    // stably merge a[lo .. mid) with a[mid .. hi) using aux[lo .. hi)
    private static void merge(int[] a, int[] aux, int lo, int mid, int hi) {

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

        int i = lo, j = mid;
        for (int k = lo; k < hi; k++) {
            if      (i >= mid)              a[k] = aux[j++];
            else if (j >= hi)               a[k] = aux[i++];
            else if (aux[j] < aux[i])       a[k] = aux[j++];
            else                            a[k] = aux[i++];
        }
    }

    public static void main(String[] args)
    {
        int[] a = StdIn.readAllInts();
        show(a);
        sort(a);
        show(a);
    }
}
```

Rascunho

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[] 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 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 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.)

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

(ii) Se a está ordenado, 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, quando a está ordenado, 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) É fácil remover a recursão de `countS()` (e de `countL()`). Implemente `countS()` de forma não-recursiva (aqui queremos substituir as *duas* funções chamadas `countS()` por *uma única* função equivalente com mesmo nome, com parâmetros a e x).

Q3 (4.0 pontos) Considere as seguintes funções `mystery()`:

```
// We assume a[0] <= a[1] <= a[2] <= ...
public static int mystery(int x, int[] a) {
    int N = a.length;
    if (N == 0 || x > a[N - 1])
        return N;
    return mystery(x, a, 0, N - 1);
}

// We assume x <= a[hi] and lo <= hi
public static int mystery(int x, int[] a, int lo, int hi) {
    if (lo == hi)
        return hi;
    int mid = lo + (hi - lo) / 2;
    if (x <= a[mid])
        return mystery(x, a, lo, mid);
    else
        return mystery(x, a, mid + 1, hi);
}
```

Considere o vetor de inteiros

`a = {0, 3, 4, 4, 5, 6, 7, 7, 7, 9, 9, 10, 11, 12, 12, 14, 17, 17, 18, 20}`

com 20 inteiros em ordem não-decrescente.

(a) Qual é o valor devolvido pela chamada `mystery(50, a)`?

(b) Qual é o valor devolvido pela chamada `mystery(7, a)`?

(c) Qual é o valor devolvido pela chamada `mystery(13, a)`?

(d) Qual é o valor devolvido pela chamada `mystery(-50, a)`?

(e) O que é o valor devolvido por `mystery(x, a)`, supondo que a chamada é executada com um vetor a em ordem não-decrescente? Seja preciso em sua resposta. [**Sugestão.** Considere o conjunto I dos índices i tais que $x \leq a[i]$. Analise separadamente os casos em que $I = \emptyset$ e $I \neq \emptyset$.]

(f) Justifique sua resposta do item anterior: prove que, de fato, `mystery(x, a)` devolve o valor especificado em sua resposta para o item (e). [**Sugestão.** Separe os casos em que $I = \emptyset$ e $I \neq \emptyset$. Para analisar o caso em que $I \neq \emptyset$, diga explicitamente o que é o valor devolvido pela chamada `mystery(x, a, lo, hi)` no caso em que a "resposta correta" `r` satisfaz `lo <= r <= hi`. Prove sua afirmação por indução em `hi - lo`.]

(g) Suponha que executamos `mystery(x, a)` com um vetor a com `N = a.length` igual a 2^k para algum inteiro $k \geq 0$ e algum valor x. Quantas vezes o teste `x <= a[mid]` será executado? Justifique sua resposta brevemente. [**Observação.** A função recursiva `mystery()` é sucessivamente executada em trechos do vetor a; para justificar sua resposta, diga explicitamente quantos elementos esses trechos têm. **Bônus.** O que acontece se `N` não é necessariamente uma potência de 2?]

* * *

(Rascunho)

(Rascunho)