PROVA 1

MAC121 ALGORITMOS E ESTRUTURAS DE DADOS I

2o. ***SEMESTRE DE 2017***

Nome:

Número USP:

Instruções:

(*1*) Esta prova é individual.

(*2*) Não destaque as folhas deste caderno.

(*3*) A prova consiste de **6** questões (contando a Questão 0 nesta página).

(*4*) As respostas devem estar nos locais indicados.

(*5*) Não é permitido o uso de aparelhos eletrônicos de qualquer natureza.

(*6*) Não é permitido o uso de folhas avulsas para rascunho.

(*7*) Não é necessário apagar seus rascunhos.

(*8*) Não é permitido consultar nenhum material ou consultar colegas.

Assinatura:

Sua assinatura acima atesta a autenticidade e originalidade de seu trabalho e que você se compromete a seguir o código de ética da USP em suas atividades acadêmicas, incluindo esta prova.

Boa sorte!

Q	0	1	2	3	4	5	**Total**
Nota							

Q0. [5 pontos] Leia o conteúdo desta página e preencha os itens requisitados. Assine acima, e atente ao significado de sua assinatura.

Data: 2017/10/5, 11:38am

Rascunho

Q1. [15 pontos] Para cada uma das funções $f(N)$ abaixo, dê funções $g(N)$ tais que $f(N) \sim g(N)$ (como nos exemplos). As funções $g(N)$ devem ser *as mais simples possíveis.*

(*i*) $f(N) = N + 1$ *Resp.*: $g(N) = N$

(*ii*) $f(N) = 1 + 1/N$ *Resp.*: $g(N) = 1$

(*iii*) $f(N) = (1 + 1/N)(1 + 2/N)$ *Resp.*: $g(N) = 1$

(*iv*) $f(N) = 2N^3 - 15N + 7$ *Resp.*: $g(N) = 2N^3$

(*v*) $f(N) = \log(2N)/\log N$ *Resp.*: $g(N) = 1$

(*vi*) $f(N) = \log(N^2 + 1)/\log N$ *Resp.*: $g(N) = 2$

(*vii*) $f(N) = 2^N + N^2$ *Resp.*: $g(N) = 2^N$

(*viii*) $f(N) = N^2 2^N$ *Resp.*: $g(N) = N^2 2^N$

Q2. [20 pontos] Diga quantas vezes o comando `sum++` é executado nos códigos abaixo. Dê a resposta usando a notação Θ (como no exemplo).

(*i*)
```
int sum = 0;
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        sum++;
```
Resp.: $\Theta(N^2)$

(*ii*)
```
int sum = 0;
for (int n = N; n > 0; n /= 2)
    for (int i = 0; i < n; i++)
        sum++;
```
Resp.: $\Theta(N)$

(*iii*)
```
int sum = 0;
for (int i = 1; i <= N; i *= 2)
    for (int j = 0; j < i; j++)
        sum++;
```
Resp.: $\Theta(N)$

(*iv*)
```
int sum = 0;
for (int i = 1; i < N; i *= 2)
    for (int j = 0; j < N; j++)
        sum++;
```
Resp.: $\Theta(N \log N)$

(*v*)
```
int sum = 0;
for (int n = N; n > 0; n = 2*n/3)
    for (int i = 0; i < n; i++)
        sum++;
```
Resp.: $\Theta(N)$

(*vi*)
```
int sum = 0;
for (int n = N; n > 0; n = 2*n/3)
    for (int j = 0; j < N; j++)
        sum++;
```
Resp.: $\Theta(N \log N)$

Dado um inteiro positivo N, seja $|N|_i$ o número de bits iguais a i na expansão binária de N (assim, $|N|_0 + |N|_1 = \lfloor \log_2 N \rfloor + 1$).

(*vii*) Qual é o valor final (exato) de `sum` após a execução do trecho de código em (*ii*) acima? Dê a respostam en função de N e $|N|_1$. *Resp.*: $2N - |N|_1$

(*viii*) Qual é o valor final (exato) de `sum` após a execução do trecho de código em (*iii*) acima? *Resp.*: $2^{\lfloor \log_2 N \rfloor + 1} - 1$

Q3. [20 pontos]

(i) Seja a um vetor com N inteiros. Uma *inversão* em a é um par (i, j) com $0 \leq i < j < N$ e $a[i] > a[j]$. Quantas inversões tem o vetor `a = {3, 1, 4, 5, 9, 2, 6, 7}`? *Resp.*: 7

(ii) Considere a seguinte implementação do algoritmo de ordenação por inserção:

```
public static void sort(Comparable[] a) {
    int N = a.length;
    for (int i = 1; i < N; i++) {
        for (int j = i; j > 0; j--) {
            if (a[j-1].compareTo(a[j]) > 0)
                exch(a, j-1, j);
            else break;
        }
    }
}
```

Seja `a` o vetor do item (i) acima. Quantas vezes `compareTo()` é executado na chamada `sort(a)`? *Resp.*: 13

(iii) Seja agora `a` um vetor genérico. Quantas vezes `compareTo()` é executado na chamada `sort(a)` no máximo? Dê a resposta em função do número de inversões em `a` e o número de elementos N em `a`. *Resp.*: #inversões $+ N - 1$

(iv) Suponha que em uma certa aplicação queremos ordenar um vetor `a`. Suponha também que sabemos que esse vetor a ser ordenado está sempre quase ordenado, no sentido que ele sempre contém no máximo $2N$ inversões, onde N é o número de elementos em `a` e sabemos que N será em geral bastante grande. Qual algoritmo de ordenação você recomendaria? *Resp.*: Insertion sort

Justificativa:

Sabemos que ordenação por inserção fará no máximo $3N$ comparações (e $2N$ trocas) e assim terá complexidade $O(N)$. Como quicksort e mergesort têm complexidade $\Omega(N \log N)$, para valores grandes de N ordenação por inserção será melhor.

Q4. [20 pontos] Considere a seguinte implementação do mergesort.

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
    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].compareTo(a[i]) < 0) aux[k] = a[j++];
        else                               aux[k] = a[i++];
    }
    for (int k = lo; k < hi; k++) a[k] = aux[k];
}
public static void sort(Comparable[] a, Comparable[] 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(Comparable[] a) {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length);
}
```

(*i*) Suponha que `a = {E, X, M, P, L, O, D, A, R, T}` e que vamos ordenar `a` fazendo a chamada `sort(a)`. Considere o momento em que as chamadas recursivas de `sort()` já foram feitas, e que resta agora executar (a última chamada de) `merge()`. Diga qual é o conteúdo de `a` nesse momento (antes da última execução de `merge()`).

Resp.: E L M P X A D O R T

(*ii*) Suponha agora que a última execução de `merge()` ocorre. Quantas vezes `compareTo()` é executado nessa chamada de `merge()`? *Resp.*: 9

(*iii*) Suponha agora que `a` seja um vetor arbitrário com N elementos, a ser ordenado com uma chamada de `sort(a)`. Considere o momento em que as chamadas recursivas de `sort()` já foram feitas, e que resta agora executar a última chamada de `merge()`. Quantas vezes `compareTo()` é executado nessa última chamada de `merge()`? Esse número depende de `a` ou é univocamente determinado pelo valor de N? Se ele depende de a, diga qual é o valor máximo possível e diga qual é o valor mínimo possível (em função de N).

Resposta:

O número de vezes que `compareTo()` é executado depende de `a`. Esse número é sempre pelo menos $\lfloor N/2 \rfloor$ e no máximo $N - 1$.

Q5. [20 pontos] Considere o algoritmo de partição de Sedgewick abaixo.

```
private static int partition(Comparable[] a, int lo, int hi) {
    int i = lo;  int j = hi + 1;
    Comparable v = a[lo];
    while (true) {
        while (less(a[++i], v))
            if (i == hi) break;
        while (less(v, a[--j])) ;
        if (i >= j) break;
        exch(a, i, j);
    }
    exch(a, lo, j);
    return j;
}
```

(i) Suponha que `a = {M, X, O, P, L, E, D, A, R, T}` e que `exch(a, i, j)` imprime o conteúdo de `a`. Dê a saída da execução do código acima.

Resposta:

```
M A O P L E D X R T
M A D P L E O X R T
M A D E L P O X R T
L A D E M P O X R T
```

Exemplo. Se `a = {4, 7, 1, 8, 3, 2, 5}`, a resposta seria como abaixo

Resposta:

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

(ii) No item (i) acima, quantas vezes `less()` é executado para `a = {M, X, O, ... }`?

Resp.: 11

(iii) Mais geralmente, se $N =$ `hi` $-$ `lo` $+ 1$, quantas vezes `less()` é executado? Esse número depende de `a` ou é univocamente determinado pelo valor de N? Se ele depende de a, diga qual é o valor máximo possível e diga qual é o valor mínimo possível (em função de N).

Resposta:

O número de vezes que `less()` é executado depende de `a`. Valor mínimo: N. Valor máximo: $N + 1$.