Gabriel Catrojan Faria Oliveira - 20.1.4004

1) a(n) = (5n² + 15) → O(n²)

= (10n) → O(n)

b(n) = 200n → O(n)

= (30 + 10n) → O(n)

Escolheria o algoritmo b(n), já que sua ordem de complexidade, tanto no pior e melhor caso, é igual a O(n). Isso, o n cresce menos que n², como no pior caso da a(n). Como n cresce menos a medida que seu valor aumenta, o seu tempo de execução é menor comparado o n², que cresce muito mais rapido. Lembrando que para ver a ordem de complexidade, desconsidere as constantes.

2) a) typedef struct {
int x, y;
} TPonto;

typedef struct {
int raio;
TPonto *centro;
} TCirculo

b) void cria Circulo (TCirculo **c, int x, int y, int raio) {
c = (TCirculo) malloc (sizeof (TCirculo));
c -> centro = (TPonto*) malloc (sizeof (TPonto));
if (c == NULL) {
printf ("Falta memória");
exit (1);
}
if (c -> centro == NULL) {
printf ("Falta de memória");
exit (1);
}
c -> centro -> x = x;
c -> centro -> y = y;
c -> raio = raio;
}

c) void liberaCirculo (TCirculo **c){

free ((*c) -> centro);

free (*c));

(*c) = NULL;

d) #include <math.h> // Deve aparecer para usar o pow

int checaPontoCirculo (TCirculo *c, TPonto *ponto)

double distancia;

// Distancia entre X

distancia = sqrt ((pow (c -> centro -> x) - (ponto -> x)) +

(pow (c -> centro -> y) - (ponto -> y)));

// Distancia entre Y

if (distancia <= c -> raio) {

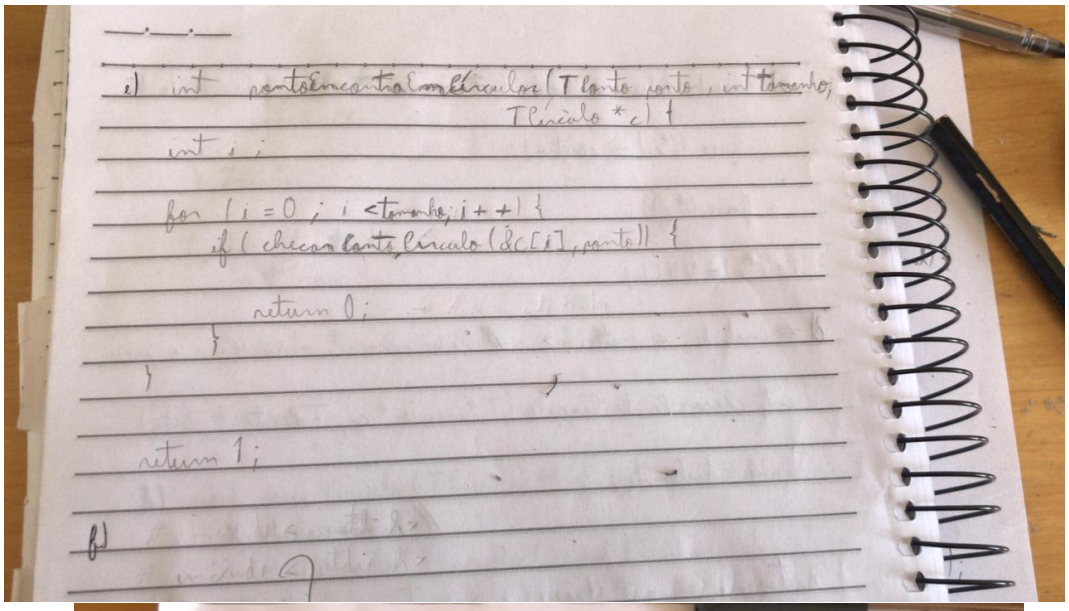
return 1;

}

else {

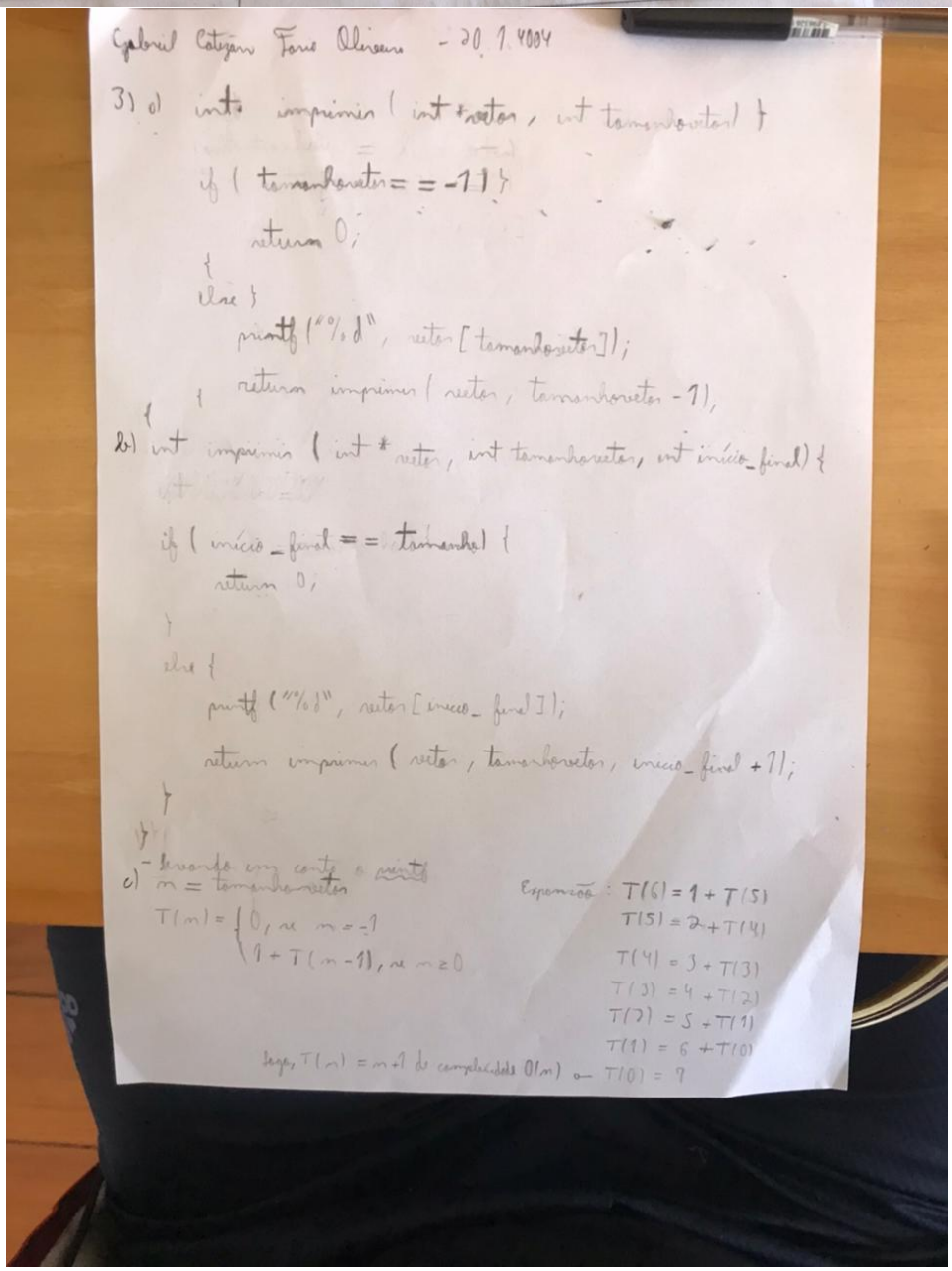
return 0;

}



a) int pontoEncontraEmCirculos(T_ponto ponto, int tamanho, T_circulo *c) {
int i;
for (i = 0; i < tamanho; i++) {
if (checaPontoCirculo(&c[i], ponto)) {
return 0;
}
}
return 1;

b)



Gabriel Catejon Faria Oliveira - 20.1.4004

3) a) int imprimir(int *vetor, int tamanhovetor) {
if (tamanhovetor == -1) {
return 0;
}
else {
printf("%d", vetor[tamanhovetor]);
return imprimir(vetor, tamanhovetor - 1);
}

b) int imprimir(int *vetor, int tamanhovetor, int início_final) {
if (início_final == tamanho) {
return 0;
}
else {
printf("%d", vetor[início_final]);
return imprimir(vetor, tamanhovetor, início_final + 1);
}
}

c) - levando em conta o print
n = tamanhovetor

$$T(n) = \begin{cases} 0, \text{ se } n = -1 \\ 1 + T(n-1), \text{ se } n \geq 0 \end{cases}$$

Expansão: T(6) = 1 + T(5)
T(5) = 2 + T(4)
T(4) = 3 + T(3)
T(3) = 4 + T(2)
T(2) = 5 + T(1)
T(1) = 6 + T(0)
T(0) = 7

Logo, T(n) = n+1 de complexidade O(n)

c) Caso implemente-lo de outra forma, usando for / loop de repetição, ou seja, de uma forma iterativa, que além de ser mais fácil de implementá-lo e entendê-lo. Com o código iterativo, o custo do espaço de memória do algoritmo vai reduzir, já não empilha conteúdos num mesmo espaço.

d) $\sum_{y=0}^{n-1} \left(\sum_{x=0}^{n-2} 3 \right) \rightarrow \sum_{y=0}^{n-1} (n-2)+1-0 \,|\, 3 = \sum_{j=0}^{n-1} |3n-3| =$

$= n\,|3n-3| = 3n^2 - 3n$

↳ $T(n) = F(n^2)$

$F(n^2) = O(n^2)$

e) ? ? ?

4) d $T(n)$ $\begin{cases} T(1)=0 \\ T(n)=n+T\left(\frac{n}{3}\right) \end{cases}$

b) $T(n) = n + (T/3)$

$= n + n/3 + T(n/3/3)$

$= n + n/3 + n/3/3 + T(n/3/3/3)$

$\vdots$

$= n + n/3 + n/3/3 + \dots n/3/3/3/\dots/3 + T(n/3/3/\dots/3)$

Como, $n = 3^k$, assim: $T(1) = T\left(\frac{n}{3^k}\right)$.

Logo, $T(n) = \sum_{i=0}^{k-1}\left(\frac{n}{3^i}\right) + T(1) = n\sum_{i=0}^{k-1}\left(\frac{1}{3^i}\right) + 1$

c) $T(n) = O(n)$