Universidade Federal de Ouro Preto
Departamento de Computação - DECOM
BCC241 - Projeto e Análise de Algoritmos
Prof. Anderson Almeida Ferreira

Nome: Caio Silas de Araujo Amaro
Matrícula: 21.1.4111

1) Usando o algoritmo que multiplica dois números binários a um custo $\Theta(n^{1.585})$, multiplique 1001 por 0110.

function multiply(x, y)
Input: Positive integers x and y, in binary
Output: Their product

$n =$ max(size of x, size of y)
if $n = 1$: return xy

x_L, $x_R =$ leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of x
y_L, $y_R =$ leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of y

$P_1 = \text{multiply}(x_L, y_L)$
$P_2 = \text{multiply}(x_R, y_R)$
$P_3 = \text{multiply}(x_L + x_R, y_L + y_R)$
return $P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2$

```
multiply(1001, 0110)
|       n = max(size of 1001, size of 0110) = max(4, 4) = 4
|       n != 1
|       xl = 10; xr = 01
|       yl = 01; yr = 10
|       P1 = multiply(xl, yl) = multiply(10, 01)
|       |       n = max(size of 10, size of 01) = max(2, 2) = 2
|       |       n != 1
|       |       xl = 1; xr = 0
|       |       yl = 0; yr = 1
|       |       P1 = multiply(xl, yl) = multiply(1, 0)
|       |       |       n = max(size of 1, size of 0) = max(1, 1) = 1
|       |       |       n == 1
|       |       └       return 1*0 = 0
|       |       P2 = multiply(xr, yr) = multiply(0, 1)
|       |       |       n = max(size of 0, size of 1) = max(1, 1) = 1
|       |       |       n == 1
|       |       └       return 0*1 = 0
|       |       P3 = multiply(xl+xr, yl+yr) = multiply(1+0, 0+1) = multiply(1, 1)
|       |       |       n = max(size of 1, size of 1) = max(1, 1) = 1
|       |       |       n == 1
|       |       └       return 1*1 = 1
|       └       return 0*2^2 + (1-0-0)*2^(2/2) + 0 = 0 + 10 + 0 = 10
|       P2 = multiply(xr, yr) = multiply(01, 10)
|       |       n = max(size of 01, size of 10) = max(2, 2) = 2
|       |       n != 1
|       |       xl = 0; xr = 1
|       |       yl = 1; yr = 0
```

```
|   |   P1 = multiply(xl, yl) = multiply(0, 1)
|   |   |   n = max(size of 0, size of 1) = max(1, 1) = 1
|   |   |   n == 1
|   |   └   return 0*1 = 0
|   |   P2 = multiply(xr, yr) = multiply(1, 0)
|   |   |   n = max(size of 1, size of 0) = max(1, 1) = 1
|   |   |   n == 1
|   |   └   return 0*1 = 0
|   |   P3 = multiply(xl+xr, yl+yr) = multiply(0+1, 1+0) = multiply(1, 1)
|   |   |   n = max(size of 1, size of 1) = max(1, 1) = 1
|   |   |   n == 1
|   |   └   return 1*1 = 1
|   └   return 0*2^2 + (1-0-0)*2^(2/2) + 0 = 0 + 10 + 0 = 10
|   P3 = multiply(xl+xr, yl+yr) = multiply(10+01, 01+10) = multiply(11, 11)
|   |   n = max(size of 11, size of 11) = max(2, 2) = 2
|   |   n != 1
|   |   xl = 1; xr = 1
|   |   yl = 1; yr = 1
|   |   P1 = multiply(xl, yl) = multiply(1, 1)
|   |   |   n = max(size of 1, size of 1) = max(1, 1) = 1
|   |   |   n == 1
|   |   └   return 1*1 = 1
|   |   P2 = multiply(xr, yr) = multiply(1, 1)
|   |   |   n = max(size of 1, size of 1) = max(1, 1) = 1
|   |   |   n == 1
|   |   └   return 1*1 = 1
|   |   P3 = multiply(xl+xr, yl+yr) = multiply(1+1, 1+1) = multiply(10, 10)
|   |   |   n = max(size of 10, size of 10) = max(2, 2) = 2
|   |   |   n != 1
|   |   |   xl = 1; xr = 0
|   |   |   yl = 1; yr = 0
|   |   |   P1 = multiply(xl, yl) = multiply(1, 1)
|   |   |   |   n = max(size of 1, size of 1) = max(1, 1) = 1
|   |   |   |   n == 1
|   |   |   └   return 1*1 = 1;
|   |   |   P2 = multiply(xr, yr) = multiply(0, 0)
|   |   |   |   n = max(size of 0, size of 0) = max(1, 1) = 1
|   |   |   |   n == 1
|   |   |   └   return 0*0 = 0;
|   |   |   P3 = multiply(xl+xr, yl+yr) = multiply(1+0, 1+0) = multiply(1, 1)
|   |   |   |   n = max(size of 1, size of 1) = max(1, 1) = 1
|   |   |   |   n == 1
|   |   |   └   return 1*1 = 1;
|   |   └   return 1*2^2 + (1-1-0)*2^(2/2) + 0 = 100 + 0 + 0 = 100
|   └   return 1*2^2 + (100-1-1)*2^(2/2) + 1 = 100 + 100 + 1 = 1001
└   return 10*2^4 + (1001-10-10)*2^(4/2) + 10 = 100000 + 10100 + 10 = 110110
```

2-

a) T(n) = 5*T(n/2) + O(n)

   Teorema mestre:
   a = 5
   b = 2
   d = 1

   $\log_b a = \log_2 5 > 1 = d$

   Logo, T(n) = $O(n^{\log_2 5})$

b) T(n) = 2*T(n-1) + O(1)
   T(n) = 2(2*T(n-2) + O(1)) + O(1) = 2^2*T(n-2) + 2*O(1) + O(1)
   T(n) = 2^2(2*T(n-3) + O(1)) + 2*O(1) + O(1) = 2^3*T(n-3) + 2^2*O(1) + 2*O(1) + O(1)
   .
   .
   .
   T(n) = 2^k*T(n-k) + $\sum_{i=0}^{k-1} O(1)$ n - k = 0 -> n = k
   T(n) = 2^n*T(n-n) + $\sum_{i=0}^{n-1} O(1)$ = 2^n*T(0) + $\sum_{i=0}^{n-1} O(1)$ = 2^n*O(1) + n*O(1)
   T(n) = O(2^n) + O(n)
   T(n) = O(2^n)

c) T(n) = 9*T(n/3) + O(n^2)

   Teorema mestre:
   a = 9
   b = 3
   d = 2

   $\log_b a = \log_3 9 = 2 = d$

   Logo, T(n) = O(n^2logn)

   Como $\lim_{n \to \infty} \frac{n^2 logn}{n^{\log_2 5}} = 0$, logo $n^2 logn$ = $O(n^{\log_2 5})$

Como $\lim_{n \to \infty} \frac{n^{log_2 5}}{2^n} = 0$, logo $n^{log_2 5} = O(2^n)$

Como $n^2 logn = O(n^{log_2 5})$ e $n^{log_2 5} = O(2^n)$, temos que $n^2 logn = O(2^n)$

Portanto, n^2 logn é assintoticamente dominada pelas outras duas funções, e o algoritmo C é o que executa no menor tempo. Logo, eu escolheria o algoritmo C.