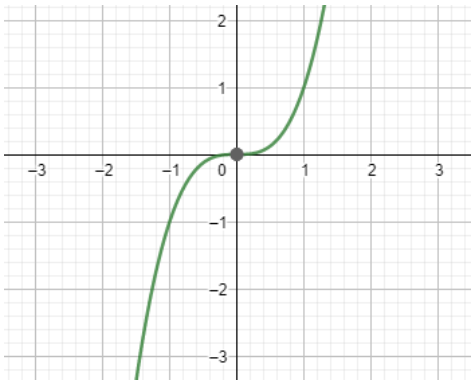
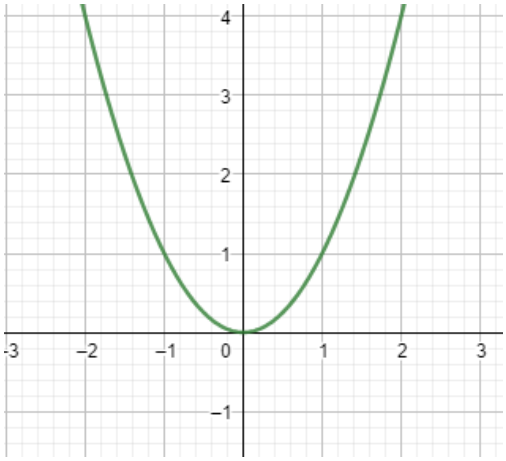
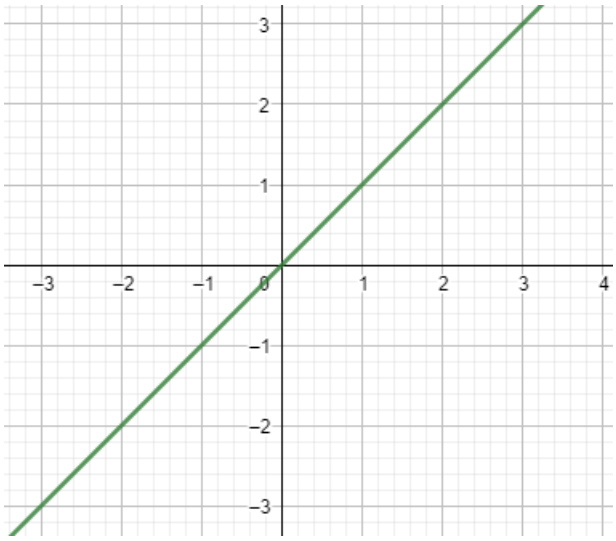
```
/*
TRABALHO TEORICO 6
PONTIFICIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
ALUNO: JULIA VELOSO DIAS ID: 1314675
PROFESSOR: MAX DO VAL MACHADO
CURSO: CIENCIA DA COMPUTAÇÃO // TURNO: MANHÃ // PERIDO: SEGUNDO
*/

//QUESTÃO 1
a) 2^10 = 1024
b) lg(1024) = 10
c) lg(17) = 4,08746284125034
d) TETO lg(17) = 5
e) PISO lg(17) = 4

//QUESTAO 2
n^3
```



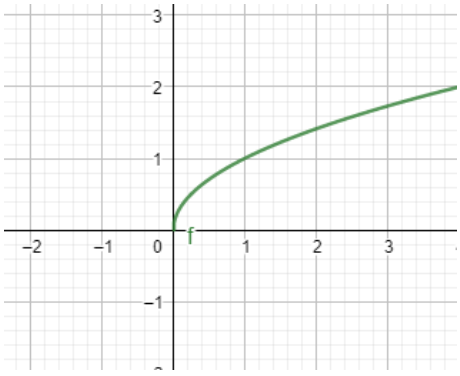
n^2



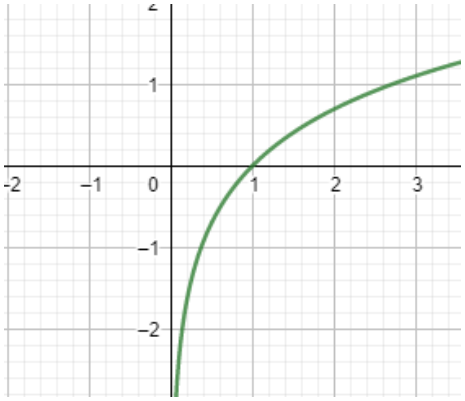
N



Raiz N



Ln2 n



//QUESTAO 3

```
int min = array[0];
for (int i = 1; i < n; i++)
{ if (min > array[i]){
 min = array[i];
}
}
```

Temos n – 1 comparaçoes

A maior comparação ocorre dentro do if do array

É um bom codigo pois testamos tudo para resultado

//

```
boolean resp = false;
for (int i = 0; i < n; i++){
if (array[i] == x){
resp = true; i = n;
} }
```

Mesmo caso da outra. No melhor caso, quando falso, roda apenas uma vez

Quando verdadeiro, roda varias vezes

É bom porque testamos tudo

//

//

//

Um aluno deve procurar um valor em um array de números reais. Ele tem duas alternativas. Primeiro, executar uma pesquisa sequencial. Segundo, ordenar o array e, em seguida, aplicar uma pesquisa binária. O que fazer?

A primeira que tem menos custo, theta(n).

//

a) 3n2 + 5n + 1 é O(n): falso

b) 3n2 + 5n + 1 é O(n2): verdadeiro

c) 3n2 + 5n + 1 é O(n3): verdadeiro

d) 3n2 + 5n + 1 é Ω(n): verdadeiro

e) 3n2 + 5n + 1 é Ω(n2): verdadeiro

f) 3n2 + 5n + 1 é Ω(n3): falso

g) 3n2 + 5n + 1 é **Θ**(n): falso

h) 3n2 + 5n + 1 é **Θ**(n2): verdadeiro

i) 3n2 + 5n + 1 é **Θ**(n3): verdadeiro

- O é o limite superior, logo, se um algoritmo é O(f(n)), ele também será O(g(n)) para toda função g(n) tal que "g(n) é maior que f(n)" melhor caso
- Ω é o limite inferior, logo, se um algoritmo é Ω(f(n)), ele também será Ω (g(n)) para toda função g(n) tal que "g(n) é menor que f(n)" pior caso
- **Θ** é o limite justo, logo, g(n) é O(f(n)) and Ω(f(n)) se e somente se g(n) é **Θ** (f(n)) medio

função	O(1)	O(lgn)	O(n)	O(n.lg (n))	$O(n^2)$	$O(n^3)$	$O(n^5)$	$O(n^{20})$
f(n) = lg(n)	f	v	v	v	v	v	v	v
f(n) = n . lg(n)	f	f	f	v	v	v	v	v
f(n) = 5n + 1	f	f	v	v	v	v	v	v
$f(n) = 7n^5 - 3n^2$	f	f	f	f	f	f	v	v
$f(n) = 99n^3 - 1000n^2$	f	f	f	f	f	v	v	v

$f(n) = n^5 - 99999n^4$	f	f	f	f	f	f	v	v

função	Ω (1)	Ω (lgn)	Ω (n)	Ω (n.lg(n))	Ω (n^2)	Ω (n^3)	Ω (n^5)	Ω (n^{20})
$f(n) = lg(n)$	v	v	f	f	f	f	f	f
$f(n) = n . lg(n)$	v	v	v	v	f	f	f	f
$f(n) = 5n + 1$	v	v	v	f	f	f	f	f
$f(n) = 7n^5 - 3n^2$	v	v	v	v	v	v	v	f
$f(n) = 99n^3 - 1000n^2$	v	v	v	v	v	v	f	f
$f(n) = n^5 - 99999n^4$	v	v	v	f	v	v	v	f

função	θ (1)	θ (lgn)	θ (n)	θ (n.lg(n))	θ (n^2)	θ (n^3)	θ (n^5)	θ (n^{20})
$f(n) = lg(n)$	f	v	f	f	f	f	f	f
$f(n) = n . lg(n)$	f	f	f	v	f	f	f	f
$f(n) = 5n + 1$	f	f	v	f	f	f	f	f
$f(n) = 7n^5 - 3n^2$	f	f	f	f	f	f	v	f
$f(n) = 99n^3 - 1000n^2$	f	f	f	f	f	v	f	f
$f(n) = n^5 - 99999n^4$	f	f	f	f	f	f	v	f

//

a) f(n) + g(n) - h(n) -- O(f(n) x g(n)))

b) O(f(n) + O(g(n)) - O(h(n)) -- O(máximo(f(n),g(n)) - O(h(n)))

c) f(n) x g(n) -- O(f(n) x g(n))

d) g(n) x l(n) + h(n) -- O(máximo(f(n),g(n)) - O(h(n))

e) f(n) x g(n) x l(n) -- O(f(n) x g(n))

f) O(O(O(O(f(n))))) -- O(f(n))

//

a) C= 3 m = 6
b) C= 2 m=4
c) N não é limite superior

//

d) C = 2 m = 3
e) C = 1 m = 3
f) N^3 não é inferior ao n^2

//

g) C1 = 2
h) C2 = 3
i) M = 5
j) Não é justo n e sim n2
k) O justo é n2

//

```
void sistemaMonitoramento() {
    if (telefone() == true && luz() == true){
        alarme(0);
    } else {
        alarme(1);
    }
    for (int i = 2; i < n; i++){
        if (sensor(i- 2) == true){
            alarme (i - 2);
        } else if (camera(i- 2) == true){
            alarme (i - 2 + n);
} } }
```

Pior = On

Melhor = O2

//

Pior = O(n) + n

Melhor = O(n)

//

```
if(n<3)  n . 2;
 else{
for(int i=0; i < n; i++) {
N+=34
N = n * 1
}
}
```

| O(n . 2)

Melhor caso: n < 3 | O(1)

//

No Exercício Resolvido (10), verificamos que quando desejamos pesquisar a existência de um elemento em um array de números reais é adequado executar uma pesquisa sequencial cujo custo é $\Theta(n)$. Nesse caso, o custo de ordenar o array e, em seguida, aplicar uma pesquisa binária é mais elevado, $\Theta(n * lg(n)) + \Theta(lg(n)) = \Theta(n * lg(n))$. Agora, supondo que desejamos efetuar n pesquisas, responda qual das duas soluções é mais eficiente.

Já ordena os arrays