1. Seja o programa a seguir. Determine o valor do ponteiro y no programa:

```
#include <iostream>

using namespace std;

int r, *y, s;

int main()
{
    r = 5;
    s = r + 2;
    y = &s;
    *y = ++(*y) + (s)++ + (r)++;
    cout << *y << endl;
    return 0;
}

```

Solução:

A precedência entre os operadores vai definir o resultado deste programa.
O $++$ pré-fixado é executado antes da atribuição e o $++$ pos-fixado é executado posteriormente.
Ex:

$$y = ++a\ +\ b$$

sendo a = 1 e b =2 inicialmente.
Após executar temos o valor 4 para y.
O $++$ pré-fixado incrementa o valor de a, que inicialmente é 1, e fica dois, que é somado a variável b que contém o valor 2, resultando em 4.

Quando o $++$ é pós-fixado é feita a soma dos valores, primeiro, para posteriormente incrementar o valor. Seja em:

$$y = a++\ +\ b$$

Neste caso ele soma os valores de a e b e incrementa ao final, porém, em um momento posterior, fazendo com que a atribuição receba os valores sem o incremento. Se o mesmo for feito, como a seguir, o resultado ao invez de ser 3 será 4. Neste caso, como não tem atribuição ele aproveita o incremento pós-fixado.

$$cout << a++\ +\ b << endl;$$

Baseado neste critérios o resultado da questão é 21. .

2. Seja o programa a seguir. Determine o valor da variável b.

```
#include <iostream>

using namespace std;

int b, *a, *c;

int main()
{
```

```
    b = 10;
    a = &b;
    c = a;
    *a = *c + b++;
    b = (*a)++ + ++(*c);
    cout << b << endl;
    return 0;
}
```

Solução:

Baseado na mesma lógica aplicada a questão 1, temos que o resultado é 42.

3. Determine a ordem assintótica do (pior caso) algoritmo a seguir:

```
    int achou = -1;               1+
    for (int i=0; i < n; i++) {   n+
        if (v[i] == alvo){        n+
         achou=i;                 1+
         break;                   1+
        }
    }                       Total:2n + 3
```

Solução:

No código acima temos a soma da quantidade de vezes em que cada linha é executada. Gerando o valor $2n + 3$ cuja grandeza está em n. Portanto temos $O(n)$.

4. Para o mesmo algoritmo acima, determine o melhor caso.

Solução:

O melhor caso é apresentado quando o algoritmo acha o valor na primeira tentativa. Assim a distribuição da execução de linhas fica:

```
    int achou = -1;               1+
    for (int i=0; i < n; i++) {   1+
        if (v[i] == alvo){        1+
         achou=i;                 1+
         break;                   1+
        }
    }                       Total: 5
```

A grandeza de 5 é n^0, portanto 1. Todo valor elevado a zero tem como resultado 1. Ou seja: $O(1)$ é o resultado da questão 4.

5. Ordene em ordem decrescente as ordens assintóticas a seguir:

- $O(n\ logn)$
- $O(1)$
- $O(n^2)$

- $O(n)$

Solução:
$O(1) > O(n) > O(n\ logn) > O(n^2)$

No geral temos a seguinte ordem:

$O(1) > O(log\ n) > O(n) > O(n\ logn) > O(n^2) > O(n^3) > O(n^k) > O(n!)$

6. Determine a ordem assintótica do (pior caso) algoritmo a seguir:

```
    for (int i=0; i < n; i++) {
        for (int j=0; j < n; j++) {
            if (m[i][j] = alvo)
              return m[i][j];
        }
    }
```

Solução:
Para resolver esta questão iremos criar um conjunto cujos elementos são os pares i e j que são testados junto com o alvo. Assim temos, supondo que n = 4:

$$C_1 = \{(0,0);(0,1);(0,2);(0,3)\}$$

$$C_2 = \{(1,0);(1,1);(1,2);(1,3)\}$$
$$C_3 = \{(2,0);(2,1);(2,2);(2,3)\}$$
$$C_4 = \{(3,0);(3,1);(3,2);(3,3)\}$$

Sabendo que cada conjunto possui 4 elementos e possuimos 4 linhas, temos 4x4, ou seja, n x n. Assim a ordem é $O(n^2)$

7. Determine a ordem assintótica do (pior caso) algoritmo a seguir:

```
    for (int i=n-1; i > 0; i--) {
        for (j=0;j<i;j++) {
            if (v[j] > v[j+1]) {
                int temp = v[j];
                v[j] = v[j+1];
                v[j+1] = temp;
            }
        }
    }
```

Solução:
Os conjunto para este algoritmo é formado de elementos que representam as trocas ocorridas no vetor. Assim temos que cada conjunto é formado por (considerando que os dados iniciais são [4, 3, 2, 1]):
$$C_1 = \{(1,4);(2,4);(3,4)\}$$

$$C_2 = \{(1,3);(2,3)\}$$
$$C_3 = \{(1,2)\}$$

Assim, temos três, depois duas, e finalmente, uma troca. Isso gera uma PA (Progressão Aritmética) com os seguintes elementos: $1 + 2 + 3$

Sabendo que a sequência é uma P.A temos a seguinte relação: invertendo os elementos em duas linhas subsequentes, verifica-se que a soma dos termos será sempre igual a n.

$$\begin{array}{ccccc} 1 & + & 2 & + & 3 \\ 3 & + & 2 & + & 1 \\ \hline 4 & + & 4 & + & 4 \end{array}$$

Ou seja, n = 4. Assim podemos concluir que o n (4) é somado três vezes (n-1). Portanto, a somatória dos elementos abaixo da linha é:

$$(n-1).n$$

Também considerando que temos a soma invertida de duas sequências (1,2,3) e (3,2,1) e que queremos generalizar a soma de somente uma sequência, temos que dividir $(n-1).n$ por 2. ficando então:

$$\frac{n(n-1)}{2}$$

Essa formula representa a somatória dos elementos. Resolvendo, chegamos ao seguinte termo:

$$\frac{n^2 - n}{2}$$

Como n^2 representa a maior grandeza, temos $O(n^2)$ para o algoritmo bubblesort.

8. Para o algoritmo recursivo a seguir, determine o valor da chamada da função na segunda iteração.

```
int soma(int a, int b){
   if (a == b)
      return a;
   else
      return a + soma(a,b-1);
}
```

a) Para os valores de 2 até 4

Solução:

Evolua o algoritmo até o final e verifique qual é o valor calculado quando ocorrer a segunda iteração (refazendo os passos recursivamente até alcançar a segunda chamada da função soma).

Resposta: 4.

9. Para o algoritmo recursivo a seguir, determine o valor da chamada da função na segunda iteração.

```
int multiplicacao(int a, int b){
   if (a == b)
      return a;
```

```
        else
            return (1+a) * multiplicacao(a,b-1);
    }
```

a) Para os valores de 1 até 3

Solução:

Evolua o algoritmo até o final e verifique qual é o valor calculado quando ocorrer a segunda iteração (refazendo os passos recursivamente até alcançar a segunda chamada da função multiplicação).

Resposta: 2