1)

V

F

V

V

F

V

2) n! > 2^(n) = 2^(n+1) = (3/2)^(n) > n-n^(2)+5n^(3) = n^(3) = n^(3)+log(n) > 4^(log(n)) = n^(2) > nlog(n) > n = 42 = raiz de n = log(n)

3) Ele faz duas iterações que terminam com s tendo o valor de todos os números de 1 a n somados. A complexidade é $n^2$. Sim, é possível melhorar o código diminuindo a complexidade para n:

```
for(i = 1; i <= n; i++) {
    s = s + i;
}
```

4) A complexidade é da ordem de $n^2$, com n sendo o tamanho das palavras (caso tenham o mesmo tamanho). O que o código faz é verificar se ambas as palavras compartilham as mesmas letras. Sim, existe um modo de melhorar a complexidade do código, mas só encontrei um interferindo na ordem n, então não é muito relevante. Mas é basicamente adicionando um contador no código e posteriormente verificando se este tem o mesmo valor do tamanho da palavra, acabando com o último for:

```
int i, j, cont;
char s1[] = "roma";
char s2[] = "amor";
if (strlen(s1) != strlen(s2)) {
    return 0;
}
for (i = 0; i < strlen(s1); i++) {
    for (j = 0; j < strlen(s2); j++) {
        if (s1[i] == s2[j]) {
            s2[j] = ' ';
            cont++;
            break;
        }
```

```
    }
  }
  if (cont == strlen(s2)){
    return 0;
  }
```

5) O código serve para verificar se o número n é primo. É da ordem de n.

6) Sejam T1 (n)=100· n + 15, T2 (n)=10 ·$n^2$+ 2 · n e T3 (n)=0,5 · $n^3$+ $n^2$+ 3 as equações que descrevem a complexidade de tempo dos algoritmos Alg1, Alg2 e Alg3, respectivamente, para entradas de tamanho n. A respeito da ordem de complexidade desses algoritmos, pode-se concluir que:

A complexidade de T1 é da ordem n, de T2 da ordem $n^2$ e T3 da ordem $n^3$. Isso se dá ao fato que são as variáveis com maior grandeza nas equações.