1)

inserir pilha = 42 = O(1);
busca binaria = lgn = O(log(n));
n^1/2 = O(n^1/2);
4^log(n) = O(2^logn);
n = busca-linear = O(n);
quick-sort = merge sort = = nlogn = O(n logn);
n^2 = insertion = selection = bubble = O(n^2);
multiplicação de matrizes = n^3 + log(n) = n-n^2+5n^3 = O(n^3);
2^n = 3/2 ^n = 2^(n+1) = O(2^n);
n! = O(n!).

2)

O algoritmo faz o somatório de todos os valores de 0 até n. Neste caso a complexidade é de O(n^2).
Podemos melhorar a complexidade para O(1), pois para todo natural este somatório é igual a n(n+1)/2.

3)

O algoritmo recebe duas strings e retorna 1 caso uma seja anagrama da outra, ou 0 caso contrário. Neste caso a complexidade é de n^2+n, O(n^2).
A complexidade pode melhorar para apenas n^2. Cria-se um contador igual ao tamanho dos vetores recebidos, toda vez que encontrar duas letras iguais nos vetores decrementa-se este contador. No fim verifica se esse contador é igual a 0, caso seja retorna 1, do contrário retorna 0.

4)

A função recebe um inteiro e verifica se este valor é primo, caso seja retorna 1, do contrário retorna 0. Sua complexidade é O(n^(1/2)).

5)

F-F-F-V-V-F

6) Buscar e resolver um exercício do conteúdo da aula.

Descreva e explique a complexidade do seguinte código:

```
int i, j, k = 0;
for (i = n / 2; i <= n; i++) {
    for (j = 2; j <= n; j = j * 2) {
        k = k + n / 2;
    }
}
```

Dado um certo n, temos duas variáveis que iteram, i e j.
i itera n/2 vezes.
j itera lg(n) vezes, por exemplo, para qualquer n tal que, 16 <= n < 32, j itera 4 vezes.
j = 2,4,8,16.
Portanto a função itera (n/2)*(lg(n)) vezes. Logo a complexidade é O(n * lgn).