1) Considere o programa, escrito na linguagem C, a seguir.

```c
int Cal(int n){
        if (n < 2) return 1;
        else return Cal(n – 1) + Cal(n – 2);
}

int main(){
int T = 6;
          printf("%d\n", Cal(T));
return 0;
}
```

Quantas chamadas da função Cal ocorrem na execução desse programa.

(X) 1 + 1 + 2 + 3 + 5 + 8 + 5

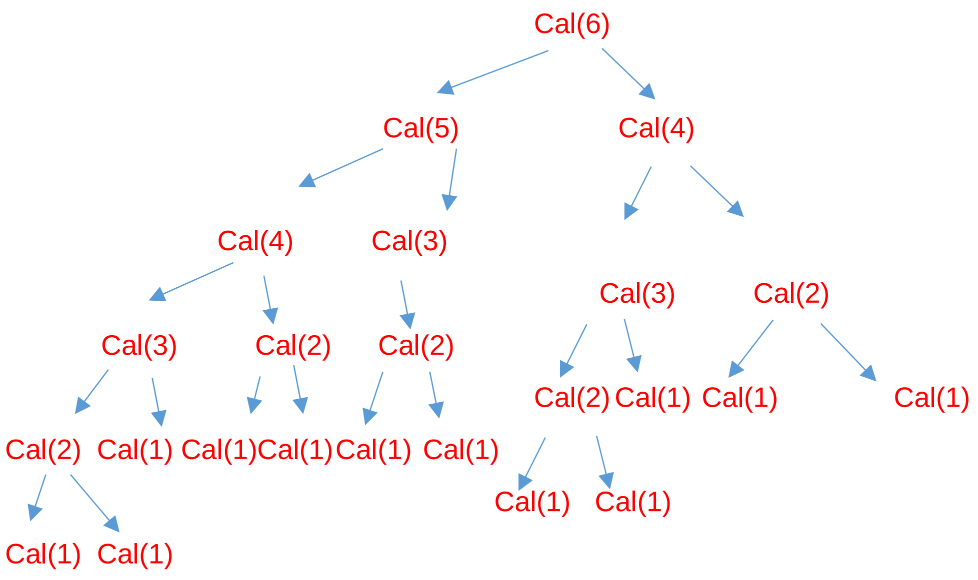
(b) 1 + 1 + 2 + 3 + 5 + 8

(c) 2*6

(d) 6^2

Justifique a resposta.

OBS: A resposta só será aceita se a justificativa estiver correta.



2) Classifique as recorrências quanto ao tipo, 1ª ordem, dividir para conquistar ou nenhum dos dois, e resolva as seguintes relações de recorrência.

a) *S(1) = 1*

S(n) = S(n - 1) + 2n, para *n ≥ 2.*

Primeira ordem

C=1 g(n) = 2n

$$S(n) = 1^{\log n} * S(1) + \sum_{i=1}^{\log n} 1^{(\log n)-i} * 2^{i+1}$$

$$S(n) = 1 + \sum_{i=1}^{\log n} 2^{i+1}$$

$$S(n) = 1 + \sum_{i=1}^{\log n} 2^{i+1} = 1 + 2^{1+logn}$$

b) *A(1) = 1*

A(n) = 2A(n/2) + 1, para *n ≥ 2.*

Dividir para conquistar

C = 2 g(n) = 1

$$A(n) = 2^{\log n} * A(1) + \sum_{i=1}^{\log n} 2^{(\log n)-i}$$

$$A(n) = n * 1 + \sum_{i=1}^{\log n} n$$

$$A(n) = n + \sum_{i=1}^{\log n} n = n * \log n$$

c) *T(1) = 1*

T(n) = nT(n - 1) , para *n ≥ 2.*

Note que não é possível aplicar as fórmulas que conhecemos para recorrências. Explique o motivo? Resolva essa recorrência usando a técnica de expandi, conjecturas e provar.

Nem primeira ordem, nem dividir e conquistar. O motivo é que o número que multiplica na segunda chamada da função T, não é uma constante.

$$T(2) = 2 * 1$$

$$T(3) = 3 * 2 * 1$$

$$T(4) = 4 * 3 * 2 * 1$$

$$T(k) = k!$$

Caso base: $T(1) = 1! = 1$ Ok

Supor que $T(k) = k!$

Mostrar que $T(k+1) = (k+1)!$

$$T(k+1) = (k+1) * T(k)$$

$$T(k+1) = (k+1) * k! = (k+1)!$$

3) Considere o famoso algoritmo de ordenação conhecido por Bubblesort.

1. **Bubblesort**(lista *A*,inteiro positivo *n)*
2.   **Para** *i =1 .. (n – 1)*
3.   **Para** *j =1 .. (n – 1)*
4.   **Se** *A[j]* > *A[j+1]*
5.   **Troca** *A[j]* com *A[j+1]*

a) Apresente, justificando sua resposta, o número de execuções da operação “Troca” no melhor caso e no pior caso para uma lista de *n* elementos.
O melhor caso seria se a lista já estivesse ordenada crescente, então não seria executado. O pior caso seria se a lista estivesse ordenada de forma decrescente, neste caso, o número de execuções seria $n-1+n-2+n-3+\cdots+n-(n-1)$

b) Explique como esse método de ordenação funciona.´
Ele irá pegar uma lista e ir comparando 2 números e trocando o menor com o maior, colocando o maior elemento no final da lista, organizando de forma crescente.

c) Enuncie a invariante do loop na linha 2. Isto é, o que se pode garantir sobre a lista A ao final da cada iteração desse loop.
Após o final de cada interação o maior número será colocado para o final da lista.

d) Prove a correção desse algoritmo usando a invariante do loop.