Universidade do Estado de Santa Catarina
Departamento de Ciência da Computação
Projeto e Análise de Algoritmos

1) Resolva as relações de recorrência:

a) $T(n) = T(n/2) + n$
$T(1) = 1$

b) $T(n) = 2T(n-1) + n$
$T(1) = 1$

c) $T(n) = 4T(n/2) + n$
$T(1) = 1$

d) $T(n) = T(n/2) + \log_2 n$
$T(1) = 1$

2) Mostre a relação de recorrência para o tempo de execução da função *foo* abaixo e calcule a complexidade de tempo. Na primeira chamada à função *foo* é passado o número de elementos do vetor como valor dos parâmetros *n* e *t*.

```
void foo(char *v, int n, int t)
{
    int i;

    if (n > 0)
    {
            v[t-n]='0';
            foo(v, n-1, t);
            v[t-n]='1';
            foo(v, n-1, t);
    }
    else
    {
            for (i=0; i < t; i++)
                    printf("%c", v[i]);
            printf("\n");
    }
}
```

3) Para a função ***potencia*** definida abaixo: Mostre qual a relação de recorrência que descreve o tempo de execução da função. Resolva essa relação de recorrência. Calcule a complexidade de tempo e a complexidade espaço para essa função.

```
unsigned int potencia (unsigned int b, unsigned int e)
{
    unsigned int r;
    if (e == 0)
            return 1;
    r = potencia(b, e/2);
    if (e % 2 == 0)
            return r*r;
    else
            return r*r*b;
}
```

4) Implemente funções que retornem o *n*-ésimo elemento da sequencia de *Fibonacci*. Uma versão da função deve ser recursiva e uma versão da função deve ser iterativa. Faça a medição de tempo de cada implementação e apresente na forma de um gráfico. Calcule a complexidade de tempo e espaço de cada implementação.

5) Implemente uma função que multiplique duas matrizes, mostre a complexidade da implementação.

6) Considerando uma árvore binária de pesquisa balanceada, mostre relação de recorrência que determina a complexidade de tempo para a função ***imprimir*** definida abaixo. Resolva essa relação de recorrência. Mostre a complexidade de tempo e espaço para essa função.

```
struct Arvore
{
    int elem;
    struct Arvore *esq, *dir;
};

void imprimir(struct Arvore *r)
{
    if (r != NULL)
    {
            imprimir (r->esq);
            printf("%d ", r->elem);
            imprimir (r->dir);
    }
}
```

Reescreva a função ***imprimir*** sem usar recorrência.