```c
/*
    Matheus Peixoto Ribeiro Vieira - 22.1.4104
*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

//!-----------------------------------------------------------------
//!   a) Encontrar o maior valor em um vetor.

void swap(int v[], int i, int j){                       // O(1)
    int temp = v[i];                                    // O(1) - Atribuição
    v[i] = v[j];                                        // O(1) - Atribuição
    v[j] = temp;                                        // O(1) - Atribuição
}

int partition(int v[], int pivo, int inicio, int fim){  // O(n)
    if(pivo != fim)                                     // O(1) - Comparação entre valores
        swap(v, pivo, fim);                             // O(1) - Chamada de função de custo O(1)

    /*
        O valor de i deve começar uma posição antes
        do início do vetor pois já será incrementado
        no início do loop de repetição.
    */
    int i = inicio - 1;                                 // O(1) - Atribuição de valores
    int j = fim;                                        // O(1) - Atribuição de valores

    while(i < j){                                       // O(1) - Comparação
        do{ i++; }while(i != j && v[i] < v[fim]);        // O(n) - Percorre o vetor
        do{ j--; }while(j != 1 && v[j] > v[fim]);        // O(n) - Percorre o vetor

        if(i < j)                                       // O(1) - Comparação
            swap(v, i, j);                              // O(1) - Chamada de função de custo O(1)
    }
    if(i != fim)                                        // O(1) - Comparação
        swap(v, i, fim);                                // O(1) - Chamada de função de custo O(1)

    return i;                                           // O(1) - Retorno de valor
}

/*
    A chamada recursiva tem custo de T(3N/4) pois está sendo considerado que o valor do pivô estará
    no segundo ou terceiro quartil, reduzindo, assim, o tamanho do vetor em 3/4
```

```
    T(N) = T(3N/4) + O(n)
    a = 1; b = 4/3; d = 1
    log_{3/4} 1   1
    0 < 1
    O(n^1)
    O(n)
*/
int selection(int v[], int k, int inicio, int fim){                         // O(n) - Custo da função a partir do teorema mestre
    // Verifica se há somente um valor no vetor e o retorna
    if (inicio == fim) return v[inicio];                                    // O(1) - Comparação e retorno

    // Escolhe um pivo aleatoriamente que está entre o início e o fim
    int pivo = inicio + rand() % (fim - inicio + 1);                        // O(1) - Atribuição e chamada de função O(1)

    int posicaoDoPivo = partition(v, pivo, inicio, fim);                    // O(n) - Chamada de função O(n)

    int tamanhoEsquerda = posicaoDoPivo - inicio + 1;                       // O(1) - Atribuição

    // Vai para a esquerda do vetor
    if (k < tamanhoEsquerda)                                                // O(1) - Comparação
        return selection(v, k, inicio, posicaoDoPivo - 1);                  // T(3n/4) no caso médio

    // Retorna o valor do pivo
    else if (k == tamanhoEsquerda)                                          // O(1) - Comparação
        return v[posicaoDoPivo];                                            // O(1) - Retorno de valor

    // Vai para a direita do vetor
    else
        return selection(v, k - tamanhoEsquerda, posicaoDoPivo + 1, fim); // T(3n/4) no caso médio
}

void encontrar_maior_valor(){
    srand(time(NULL));
    int v []= {15, 13, 107, 56, 78, 1, 23, 45, 99, 35};
    int n = 10;
    int k = n; // Procurar o k-ésimo maior valor, onde k é igual a n
    int valor = selection(v, k, 0, n - 1);
    printf("Maior valor do vetor: %d\n", valor);
}
/*
    Output: "Maior valor do vetor: 107"
*/
```

```c
//!-----------------------------------------------------------------
//!   b) Encontrar o maior e o menor elemento em um vetor.

typedef struct MaiorMenor{
    int maior, menor;
}MaiorMenor;

/*
    São feitas duas chamadas recursivas para cada chamada recursiva

    T(n) = 2T(n/2) + O(1)
    a = 2; b = 2, d = 0
    log_{2} 2   0
    1 > 0
    O(n^(log_{2} 2))
    O(n^1)
    O(n)
*/
MaiorMenor minMax(int v[], int esq, int dir, MaiorMenor maiorMenor){ // O(n)
    if(esq == dir) return maiorMenor;                           // O(1) - comparação e retorno

    int meio = (esq + dir) / 2;                                 // O(1) - Operações aritméticas básicas

    if( v[meio] > maiorMenor.maior)                             // O(1) - Comparação
        maiorMenor.maior = v[meio];                             // O(1) - Atribuição
    if( v[meio] < maiorMenor.menor)                             // O(1) - Comparação
        maiorMenor.menor = v[meio];                             // O(1) - Atribuição

    maiorMenor = minMax(v, esq, meio, maiorMenor);              // T(n/2) - Diminui o problema pela metade
    maiorMenor = minMax(v, meio+1, dir, maiorMenor);            // T(n/2) - Diminui o problema pela metade

    return maiorMenor;                                          // O(1) - Retorno do valor
}

void encontrarMinMax(){
    MaiorMenor maiorMenor;
    maiorMenor.maior = INT_MIN;
    maiorMenor.menor = INT_MAX;

    int v []= {123456, 15, 13, 107, 56, 78, 1, 23, 45, 99, 35, -50};
    int n = 12;

    maiorMenor = minMax(v, 0, n, maiorMenor);

    printf("Maior valor: %d\nMenor valor: %d\n", maiorMenor.maior, maiorMenor.menor);
```

```c
}
/*
    Output: "
        Maior valor: 123456
        Menor valor: -50
    "
*/









//!-----------------------------------------------------------------
//!   c) Exponenciação.
/*
    T(N) = T(N/2) + O(1)
    a = 1; b = 2; d = 0
    log_{2} 1   0
    0 = 0
    O(n^0 * log n)
    O(log n)
*/
int exponenciacao(int base, int expoente){              // O(log n)
    if(expoente == 1) return base;                      // O(1) - Comparação e retorno

    if (expoente % 2 == 0){                             // O(1) - Comparação
        int exp = exponenciacao(base, expoente/2);      // T(N/2) - Divide o problema na metade
        return exp * exp;                               // O(1) - Multiplicação e retorno
    }
    else{
        int exp = exponenciacao(base, (expoente-1)/2);  // T(N/2) - Divide o problema na metade
        return base * exp * exp;                        // O(1) - Multiplicação e retorno
    }

}
void exponenciacao_divisao_e_conquista(){
    int base, expoente;                                 // O(1)
    printf("Digite o valor da base: ");                 // O(1)
    scanf("%d", &base);                                 // O(1)
    printf("Digite o valor do expoente: ");             // O(1)
    scanf("%d", &expoente);                             // O(1)

    int exp = exponenciacao(base, expoente);            // O(log n)

    printf("%d ^ %d = %d\n", base, expoente, exp);      // O(1)
```

```
}
/*
    Digite o valor da base: 5
    Digite o valor do expoente: 9
    5 ^ 9 = 1953125
*/

int main(){
    encontrar_maior_valor();
    encontrarMinMax();
    exponenciacao_divisao_e_conquista();
    return 0;
}
```