ICMC - Instituto de Ciências Matemáticas e de Computação
Disciplina: Laboratório de Introdução a Ciência da Computação II
Discente: Bernardo Marques Costa
Número USP: 11795551
Docente: Leonardo Pereira

RELATÓRIO FORMAL DE ANÁLISE DE ALGORITMOS

Introdução

Neste relatório serão apresentadas as equações correspondentes a contagem de operações dos seguintes algoritmos: busca sequencial, busca binária iterativa e busca binária recursiva. Serão consideradas e contabilizadas as seguintes operações:

> Acesso a ponteiro: representada pela letra **p**
> Aritmética: representada pela letra **a**
> Atribuição: também representada pela letra **a**
> Comparação: representada pela letra **c**
> Chamada e retorno de função: **r**

Busca Sequencial

Inicialmente, analisando o código e contabilizando as operações.

```
int sequentialSearch(int *array, int key, int n){

    for(int i = 0; i < n; ++i) // 1. a + c
                                 // em diante: n(2a + c)
         // n(c + p)
        if(key == array[i]) return i;
    // r
    return -1;
}
```

A estrutura interna do laço for roda **n** vezes, enquanto o cabeçalho do laço demanda uma análise diferente:

> a primeira execução roda apenas uma vez, sendo responsável por realizar a primeira atribuição (i = 0) e a primeira comparação (i < n);

> Após a primeira iteração, temos a cada laço uma comparação (i < n) e a operação aritmética e de atribuição (i++, ou seja, i = i + 1).

De forma não intuitiva, o laço de repetição itera n vezes, mas executa uma última comparação e o comando i++, portanto o header, de fato, possui n + 1 execuções, sendo a primeira vez em que executa as operações $a + c$ e n vezes em que executa as operações $2a + c$.

Dessa forma, podemos interpretar o laço a partir da equação $a + c + n(2a + c)$.

O código interno realiza a comparação (key == array[i]) o acesso a ponteiro (array[i]) e o retorno i. Entretanto, no pior dos casos, este retorno não será efetuado, portanto podemos desconsiderá-lo.

No fim, temos o retorno da função encerrando o algoritmo.

Desta forma, temos a equação de complexidade :

$$f(n) = a + c + n(2a + c) + n(c+p) + r$$

Ou ainda

$$f(n) = (2a + 2c + p)n + a + c + r$$

Considerando tempo e espaço constante para cada uma das operações, teremos a equação com referência a constante C:

$$f(n) = 5Cn + 3C$$

Podemos plotar um gráfico correspondente a essa função de forma a analisar seu comportamento, que de maneira genérica será linear.

Busca Binária Recursiva

Analisando o código do algoritmo:

```
int binarySearchRecursive(int *array, int key, int start, int end){
    if(start <= end){ // Comparação - c
    // atribuição + 3 aritméticas - 4a
    int middle = start + (end - start) / 2;

    // acesso a ponteiro e comparação - p + a ( + r no caso base)
    if(array[middle] == key) return middle;

    // acesso a ponteiro e comparação - p + c
    else if(array[middle] > key)
        // chamada de função para metade do array R(n/2) e aritmética - a
        return binarySearchRecursive(array, key, start, middle - 1);
    else
        // chamada de função para metade do array R(n/2) e aritmética - a
        return binarySearchRecursive(array, key, middle + 1, end);

    return -1;
```

```
}
```

Temos então, no pior caso, que a chave não está no array, sendo necessário rodar toda a recursão, entrando em uma ou outra condição de comparação. Montando a equação para a busca binária recursiva (percebendo que a cada chamada recursiva, uma ou outra chamada da função é feita, após a operação condicional, reduzindo na metade o tamanho do vetor), teremos:

$$f(n) = c + 4a + p + a + p + c + R(n/2) + a$$

Expandindo sucessivamente R(n/2):

$$f(n) = 2c + 6a + 2p + (2c + 6a + 2p + R(n/4))$$

$$f(n) = 2c + 6a + 2p + [2c + 6a + 2p + (2c + 6a + 2p + R(n/8))]$$

Podemos então generalizar para k chamadas recursivas:

$$f(n) = k(2c + 6a + 2p) + R(n/2^k)$$

A recursão encerra no momento em que entra no caso base, ou seja, quando o tamanho do vetor se aproxima de 1, sendo possível encontrar, no pior caso possível, a equação em termos de n:

$$n/2^k = 1$$
$$2^k = n$$
$$k = log_2 n$$

Dessa maneira, podemos resumir a expressão quando $k = log_2 n$, ou seja, temos o caso base:

$$f(n) = log_2 n\,(2c + 6a + 2p) + c + r$$

Encontramos então, ao substituir todas variáveis para constante C:

$$f(n) = 10\,C\,log_2 n + 2C$$

Dessa maneira, temos uma função cuja tendência de crescimento é consideravelmente menor que na busca sequencial.

Busca binária Iterativa

Analisando o código

```
int binarySearchIterative(int *array, int key, int start, int end){
    int middle;

    while(start <= end){ // comparação - c

       // atribuição + 3 aritméticas - 4a
       middle = start + (end - start) / 2;

       // acesso a ponteiro e comparação - c + p (+ r para caso base)
       if(array[middle] == key) return middle;

       // acesso a ponteiro e comparação - c + p (+ 2a caso entre)
       else if(array[middle] > key)
            end = middle - 1;
       else
             start = middle + 1;
    }
    return -1; // r
}
```

Temos uma análise semelhante à anterior, sendo a equação da contagem de operações feita da seguinte forma:

$$f(n) = (X + 1) * c + X * (4a + 2c + 2p + 2a)$$

Sabemos que X corresponde a $log_2 n$ encontrado previamente, assim:

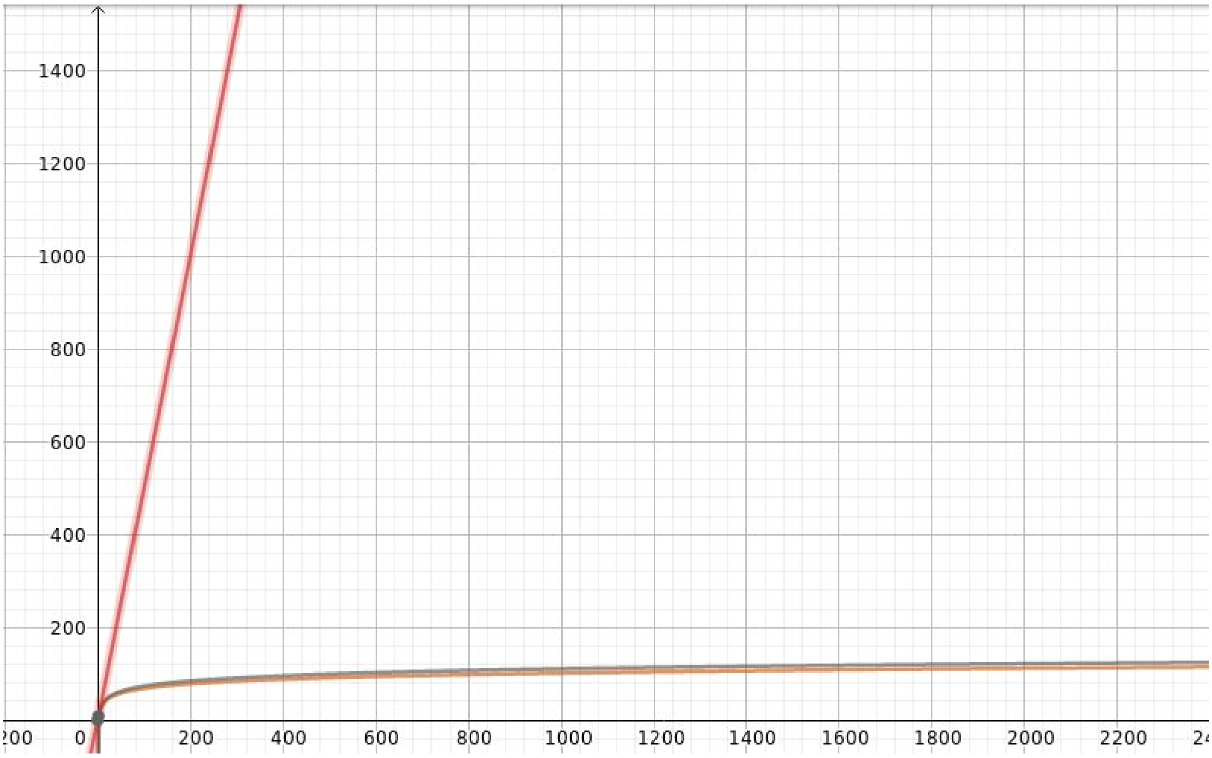
$$f(n) = (log_2 n + 1)\, c + log_2 n(4a + 2c + 2p + 2a)$$

Substituindo então todas as variáveis de operações para uma constante C, teremos uma equação de complexidade:

$$f(n) = 11\, log_2 n + c$$

Temos então uma equação parecida com a busca binária recursiva.

Comparando o comportamento de cada uma das funções encontradas para cada um dos algoritmos, teremos os gráficos. Considerando o eixo X como o sendo o número de inputs e o eixo Y como o número de operações, consequentemente, o tempo de execução, teremos a construção do seguinte gráfico, comparando as funções.



Conclusão

Percebemos que persiste, da intuição feita no relatório anterior, que a busca binária recursiva e iterativa resultam em um tempo de execução e quantidade de operações infinitamente menor conforme aumenta o número de inputs aumenta e tende para infinito.
Desta forma, temos que a aproximação da busca pelo método de divisão e conquista sobressai, que cresce em tendência logarítmica, a aproximação sequencial, que cresce em tendência linear.