Questões teóricas

1) Complexidade de Algoritmos de Busca

A busca sequencial e a busca binária são algoritmos com propósitos semelhantes, mas com características distintas. A busca sequencial percorre todos os elementos de forma linear e tem complexidade linear, sendo ideal para listar desordenadas. Já a busca binária é muito mais eficiente, com complexidade logarítmica, porém só pode ser usada se os dados estiverem previamente ordenados. A ordenação, portanto, é um pré-requisito fundamental para a aplicação da busca binária, o que pode implicar num custo adicional caso os dados estejam desorganizados.

A busca sequencial é mais apropriada quando há poucos dados ou quando não há vantagem em ordenar a estrutura. Por outro lado, a busca binária é recomendada quando se trabalha com grandes volumes de dados e buscas frequentes, desde que os elementos estejam ordenados ou possam ser mantidos ordenados.

Por exemplo, a busca ordenada pode ser a melhor para encontrar o primeiro ano bissexto após o aniversário de alguém em um input onde os dados não estão ordenados ou estão ordenados de forma inversa (decrescente), mas caso esses dados estivessem ordenados de forma crescente de forma correta, é evidente que uma busca binária poderia ser superior, principalmente se o input de anos for de larga escala com uma base de centenas de anos.

2) Impacto da ordenação dos algoritmos de busca

Se várias buscas precisam ser feitas sobre o mesmo conjunto de dados, geralmente vale a pena ordenar os dados previamente, principalmente se forem usados algoritmos de busca mais eficientes como a busca binária. Embora a ordenação inicial tenha um custo, esse custo pode ser compensado pelo ganho de desempenho nas buscas subsequentes, que passam a ter tempo de execução logarítmica em vez de linear, como ocorre na busca sequencial.

A longo prazo, essa escolha impacta positivamente o desempenho do sistema. O custo da ordenação é pago uma única vez, enquanto o benefício de buscas mais rápidas se repete a cada operação. Isso resulta em menor tempo total de execução quando o número de buscas é grande, reduzindo o consumo de recursos e melhorando a eficiência geral do sistema.

3)

A recursão consiste em uma função que chama a si mesma para resolver subproblemas menores. Já a iteração usa estruturas de repetição, como for e while, para repetir instruções até que uma condição seja satisfeita.

A recursão tem como principal vantagem a clareza e elegância em problemas que podem ser definidos com o escopo recursivo, facilitando a implementação de algoritmos complexos, como percursos de ordenação ou em linked lists. No entanto, seu principal

contra é o maior consumo de memória, pois cada chamada recursiva consome espaço na pilha de execução, o que pode levar a estouro de pilha (stack overflow) se o número de chamadas for grande demais.

Iteração geralmente é mais eficiente em termos gerais de espaço e tempo, pois evita o custo adicional de chamadas recursivas. No entanto, pode gerar implementações confusas para certo tipo de problema.

Portanto, a recursão se mostra vantajosa em algoritmos de sorting de dividir e conquistar como quicksort, problemas com estruturas hierárquicas e algoritmos clássicos de recursão como fatorial e sequência de fibonacci.

Tome como exemplo uma implementação de sequência de Fibonacci em C++ nos dois casos e perceba o quão mais intuitiva (e correta em sentido de aplicação) a recursão é:

Versão recursiva:

```
int fibonacci(int n)
{
   if (n <= 1)
   {
      return n;
   }
   else
   {
   return fibonacci(n-1) + fibonacci(n-2);
   }
}
```

Simples e reflete diretamente a definição matemática da sequência, porém consome muita memória devido suas chamadas repetidas na stack de chamadas.

Versão iterativa:

```
int fibonacci(int n)
{
   if (n <= 1)
   {
      return n;
   }
   int a = 0, b = 1, temp;
   for(int i = 2; i <= n; i++)
   {
      temp = a + b;
      a = b;
      b = temp;
   }
   return b;
}
```

Mais eficiente, com complexidade linear, porém muito mais verboso e contra intuitivo em relação à definição matemática.

4) Análise de um cenário real

Sabendo que os códigos de rastreio chegam em tempo real e desordenados, a busca linear é a mais adequada para esse cenário.

A busca binária não pode ser usada diretamente, pois ela exige que os dados estejam previamente ordenados. Como os códigos chegam fora de ordem e o sistema precisa responder rapidamente, não há tempo para ordenar os dados antes de cada busca, o que torna a busca binária inviável nesse contexto.

A busca linear, por outro lado, funciona em qualquer conjunto de dados, ordenados ou não. Embora sua complexidade temporal seja de fato inferior a outros tipos de busca, seu custo computacional e de tempo é menor no curto prazo, pois evita o tempo gasto com ordenação.

Agora, se fosse possível pré-processar os dados, ou seja, ordená-los uma vez antes de começar as buscas, a busca binária seria claramente uma melhor alternativa.