Quick Sort:
Algorítimo de organização de Arrays

Algoritmos e técnicas de programação II
Trabalho prático de programação 02

Rafael Bassi Rosseto // 191251968
Vinicius Polachini Mayer Freitas de Jesus // 191251852

Paradigma
Divide and Conquer

Divide and Conquer

O *Divide and Conquer* é um paradigma de programação de algoritmos que visa dividir um problema grande em dois problemas menores recursivamente, de forma que cada problema seja simples de resolver. Tipicamente este paradigma se divide em três fases:

- Divisão
- Conquista
- Combinação

Divide and Conquer - Divisão

O problema principal é dividido em sub-problemas menores, então verifica-se se estes sub-problemas são simples de resolver. Caso não sejam, cada sub-problema é dividido novamente até que se atinja o nível maximo de simplicidade.

Divide and Conquer - Conquista

Cada sub-problema criado na fase de divisão é resolvido recursivamente, isto é, o ultimo problema da pilha é o primeiro a ser resolvido. É importante ressaltar que após a criação de uma pilha de problemas recursivo, a mesma é resolvida na fase de conquista antes da criação de uma segunda pilha de um mesmo problema.

Divide and Conquer - Combinação

Cada resultado obtido na fase de conquista é propriamente combinado, ao combinar os resultados de todos os sub-problemas resolve-se o problema principal.

Divide and Conquer – Exemplos

Estes são alguns exemplos de algoritmos que utilizam o paradigma de Divide and Conquer:

- Busca Binária;
- Merge Sort;
- Quick Sort;
- Closest Pair Points
- Algorítimo de Strassen

Algoritmo Quick Sort

Quick Sort - Introdução

O Quick Sort é um algorítimo recursivo de organização de arrays que utiliza o paradigma de Divide and Conquer. O algorítimo utiliza um elemento pivot e particiona os elementos menores e maiores que ele à sua esquerda e direita respectivamente, de forma que o pivot fique na posição correta em um array.

Quick Sort - Pseudocódigo

```
QuickSort(arranjo[], inicio, fim){
 se (baixo < alto){
  part_index = Partição(arranjo, inicio, fim);
  QuickSort(arranjo, inicio, part_index - 1);
  QuickSort(arranjo, partição + 1, fim);
 }
}
```

Quick Sort - Pseudocódigo

```
Partição(arranjo, inicio, fim){
 pivot = arranjo[fim];
 i = (inicio - 1);
 para(j=inicio; j<=fim; j++){
  se(arranjo[j] < pivot){
   i++;
   Troca(arranjo[i] e arranjo[j]);
  }
 }
 Troca(arranjo[i+1] e arranjo[fim]);
 retorno (i+1);
}
```

Quick Sort - Pseudocódigo

Os elementos inicio e fim são a posição do primeiro e do ultimo elemento do array:

```
Arranjo[] = {20, 50, 10, 40, 70, 60, 30}
Posição =     0   1   2   3   4   5   6
```

Quick Sort - Algoritmo

1. Selecionamos um elemento pivot, pode-se utilizar o primeiro, o ultimo, o mediano ou um elemento aleatório do array.

2. Procuramos o primeiro elemento à esquerda que é maior do que o pivot, e o primeiro elemento a direita que é menor que o pivot.

3. Trocamos estes elementos de posição, e repetimos a etapa 2 a partir dos elementos que foram até que os contadores se encontrem no array.

4. Trocamos de posição o pivot e o maior elemento da ultima troca. Neste momento o pivot estará em sua posição correta no array.

```
void QuickSort(int array[], int inicio, int fim){
  int i, j, pivot, temp;
  if (inicio < fim){
    pivot = inicio;
    i = inicio;
    j = fim;
    while(i < j){
      while(array[i] <= array[pivot] && i < fim)
      i++;
      while(array[j] > array[pivot])
      j--;
      if(i < j){
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
      }
    }
    temp = array[pivot];
    array[pivot] = array[j];
    array[j] = temp;
    QuickSort(array, inicio, j-1);
    QuickSort(array, j+1, fim);
  }
}
```

Como o pivot interfere no algoritmo

Pivot

A escolha do pivot no algoritmo influencia no seu funcionamento, existem quatro opções de escolha para se utilizar:

Utilizar o primeiro elemento como pivot;

Utilizar o ultimo elemento como pivot;

Utilizar o elemento mediano como pivot;

Utilizar um elemento aleatório como pivot;

Pivot

A escolha do pivot está diretamente relacionada com o pior caso possível de complexidade do Quick Sort $O(n^2)$, este caso acontece em diferentes situações dependendo de qual elemento foi utilizado como pivot.

Se utilizarmos o Quick Sort em um array que já está ordenado utilizando o ultimo elemento como pivot, cairemos no caso de complexidade $O(n^2)$. Caso o array esteja ordenado de forma decrescente, cairemos no caso de complexidade $O(n^2)$ se o pivot utilizado for o primeiro elemento.

Bibliografia

https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/

https://medium.com/brandons-computer-science-notes/divide-and-conquer-algorithms-4e83d9999ffa

https://www.geeksforgeeks.org/quick-sort/

https://hackr.io/blog/quick-sort-in-c