Quick Sort

É um algoritmo de ordenação. O algoritmo escolhe um elemento pivô e reorganiza os elementos da matriz para que todos os elementos menores que o elemento pivô escolhido se movam para o lado esquerdo do pivô e todos os elementos maiores se movam para o lado direito. Finalmente, o algoritmo classifica recursivamente os subarrays à esquerda e à direita do elemento pivô. Ele também apresenta a vantagem da ordenação local e funciona bem até mesmo em ambientes de memória virtual.

Funcionalidade

O algoritmo funciona da seguinte forma:

1. O algoritmo recebe um array preenchido, e escolhe um pivô, que pode ser selecionado aleatoriamente.

2. Duas variáveis j e i, são posicionadas nos extremos do array. A variável j começa a percorrer da esquerda a direita comparando os valores aos do pivô. Se o valor de j for maior a constante para, e a variável i começa a percorrer da direita a esquerda. Se o valor de i for menor que o pivô, a constante para e é feita uma troca com a posição em que a variável j parou.

3. As duas variáveis andam uma casa, e o processo repete-se até o momento em que elas encontram-se durante o percurso. Quando isso acontece o array é dividido, e é feita uma chamada recursiva para cada nova repartição.

4. O processo repete-se até o momento em que só houver um único elemento em cada repartição gerada. E por fim todos os valores são unidos novamente.

Nota: A tecnica Dividir para Conquistar (DAC), que está dividida em 3 fases:
1. Dividir: envolve dividir o problema em subproblemas menores.
2. Conquer: Resolva subproblemas chamando recursivamente até serem resolvidos.
3. Combine: Combine os subproblemas para obter a solução final de todo o problema.

Complexidade

O tempo gasto pelo QuickSort, em geral, pode ser escrito da seguinte maneira:

T(n) = T(k) + T(nk-1) + (theta)(n)

Os primeiros dois termos são para duas chamadas recursivas, o último termo é para o processo de partição. O k é o número de elementos menores que o pivô.

Pior caso

Quando o processo de partição sempre escolhe o maior ou o menor elemento como pivô, ou quando o array já estivesse classificado em ordem crescente ou decrescente.

T(n) = T(n-1) + T(0) + (theta)(n) - Solução (theta)(n^2)

Melhor caso

Quando o processo de partição sempre escolhe o elemento do meio como pivô.

T(n) = 2T(n/2) + (theta)(n) - Solução (theta)(nLogn)

Caso médio

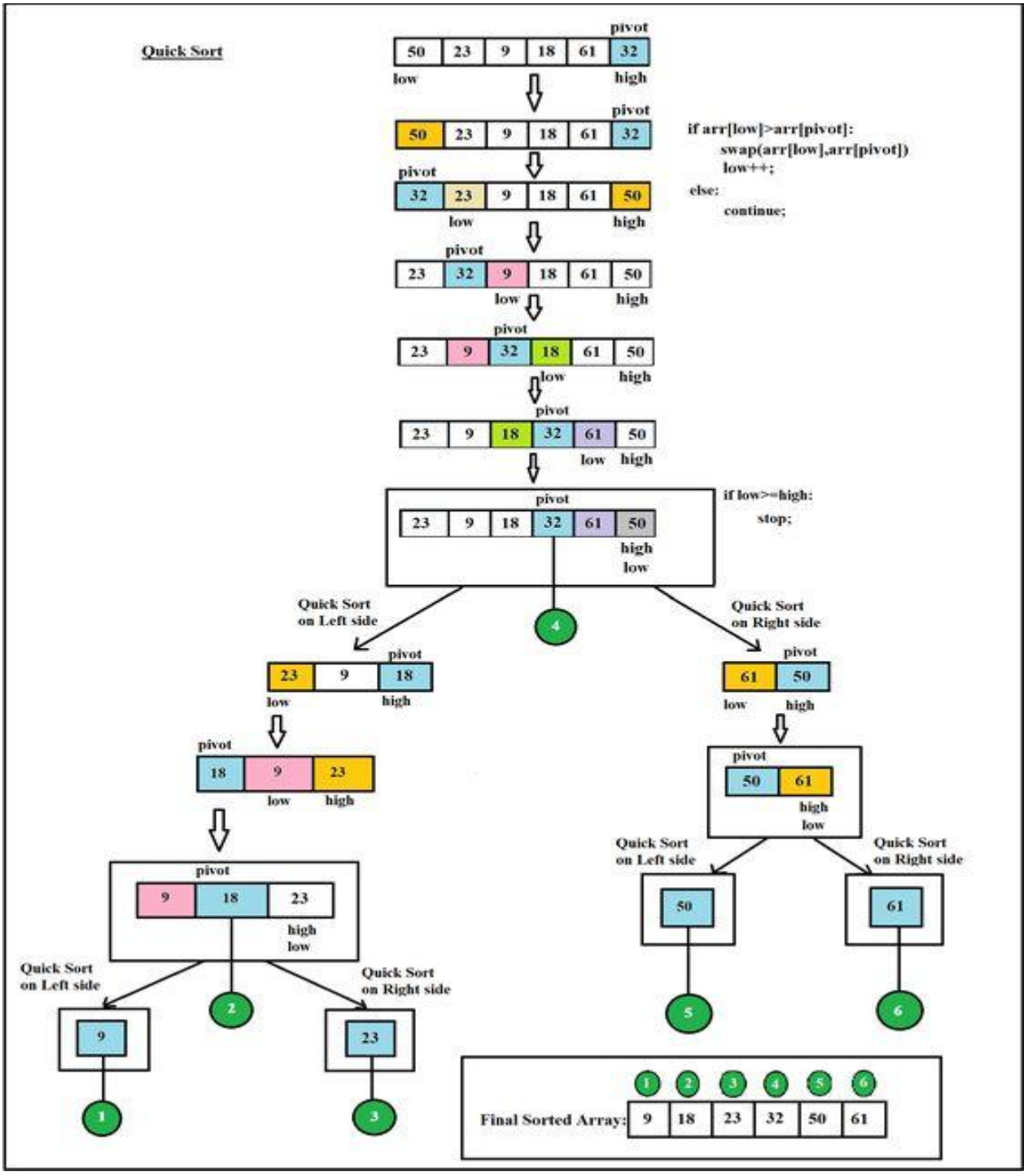
Exemplo - suponha que o algoritmo de particionamento sempre produza uma divisão proporcional de 9 para, obtemos a recorrência:

T(n) = T(n/10) + T(9n/10) + (theta)(n) - Solução (theta)(nLogn)

Estabilidade

A implementação padrão não é estável. No entanto, qualquer algoritmo de ordenação pode se tornar estável considerando índices como parâmetro de comparação.

Representação gráfica



Implementação em C

```c
#include <stdio.h>

// function to swap elements
void swap(int *a, int *b) {
 int t = *a;
 *a = *b;
 *b = t;
}

// function to find the partition position
int partition(int array[], int low, int high) {
  // select the rightmost element as pivot
 int pivot = array[high];
  // pointer for greater element
 int i = (low - 1);

 // traverse each element of the array
 // compare them with the pivot
 for (int j = low; j < high; j++) {
   if (array[j] <= pivot) {

     // if element smaller than pivot is found
     // swap it with the greater element pointed by i
     i++;

     // swap element at i with element at j
     swap(&array[i], &array[j]);
   }
 }

 // swap the pivot element with the greater element at i
 swap(&array[i + 1], &array[high]);
  // return the partition point
 return (i + 1);
}
```

```c
void quickSort(int array[], int low, int high) {
 if (low < high) {

   // find the pivot element such that
   // elements smaller than pivot are on left of pivot
   // elements greater than pivot are on right of pivot
   int pi = partition(array, low, high);

   // recursive call on the left of pivot
   quickSort(array, low, pi - 1);

   // recursive call on the right of pivot
   quickSort(array, pi + 1, high);
 }
}

// function to print array elements
void printArray(int array[], int size) {
 for (int i = 0; i < size; ++i) {
   printf("%d  ", array[i]);
 }
 printf("\n");
}

// main function
int main() {
 int data[] = {8, 7, 2, 1, 0, 9, 6};
  int n = sizeof(data) / sizeof(data[0]);
  printf("Unsorted Array\n");
 printArray(data, n);
  // perform quicksort on data
 quickSort(data, 0, n - 1);
  printf("Sorted array in ascending order: \n");
 printArray(data, n);
}
```

Referências

https://www.youtube.com/watch?v=wU7Q8Z51MUI
https://www.geeksforgeeks.org/quick-sort/
https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm-data-structure-and-algorithm-tutorials/
https://gist.github.com/marcoscastro/1dd65900cc7b188e1ab9
https://www.programiz.com/dsa/quick-sort
https://joaoarthurbm.github.io/eda/posts/quick-sort/