O algoritmo de ordenação QUICKSORT

Um dos algoritmos de ordenação mais eficiente, utilizado amplamente é o QuickSort. O algoritmo se vale da técnica de “DIVIDIR PARA CONQUISTAR” e da chamada em recursão (chamada da própria função dentro da mesma) para obtenção de um dos mais rápidos resultados em ordenação. Os algoritmos que usam estas estratégias permitem que vetores sejam reorganizados de forma mais rápida e precisa.

Funcionamento do QuickSort:

1. Escolha do Pivô:

Um elemento do vetor (array) é escolhido como pivô. Uma estratégia comum é escolher o último elemento do array ou o elemento do meio (para maior compreensão). A escolha aleatória do pivô também é considerada quando o algoritmo aumenta de tamanho e se busca agilidade na resolução da ordenação dos piores cenários.

1. **2. Particionamento:**

   O array é reorganizado de forma que todos os elementos menores ou iguais ao pivô sejam movidos para a esquerda do pivô, e todos os elementos maiores ou iguais ao pivô sejam movidos para a direita.

2. **3. Chamadas Recursivas:**
   O algoritmo é chamado recursivamente para ordenar as sub-listas à esquerda e à direita do pivô. O processo continua até que as sub-listas tenham tamanho 0 ou 1, pois listas com um ou nenhum elemento já estão ordenadas.

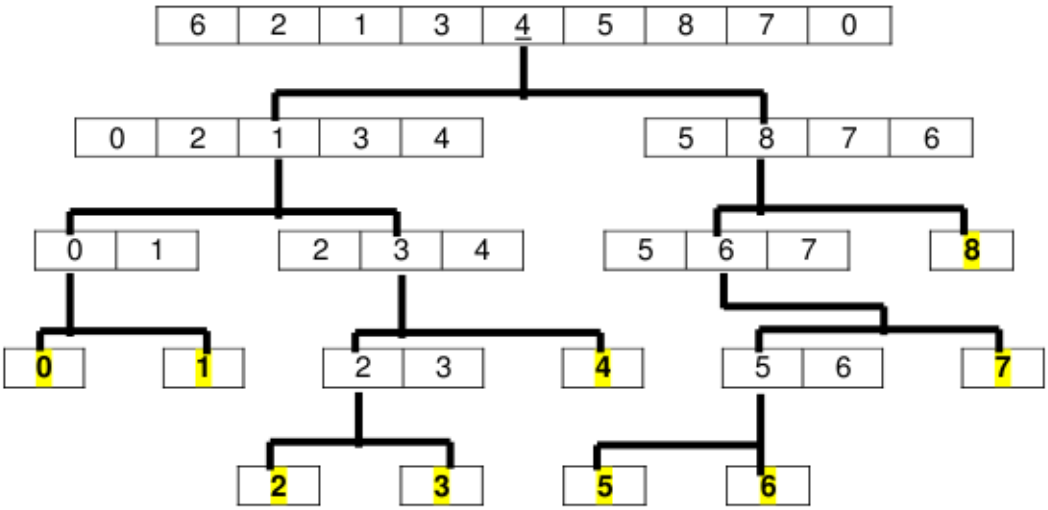
Exemplo de código:

```
public class QuickSort
{

   public static void main(String[] args) {

      int[] vetor = new int[10];

      for(int i = 0; i < vetor.length; i++){
         vetor[i] = (int) Math.floor(Math.random() * vetor.length);
      }

      System.out.println("Desordenado");
      for(int i = 0; i < vetor.length; i++){
         System.out.print(vetor[i] + " ");
      }

      quicksort(vetor, 0, vetor.length - 1);
```

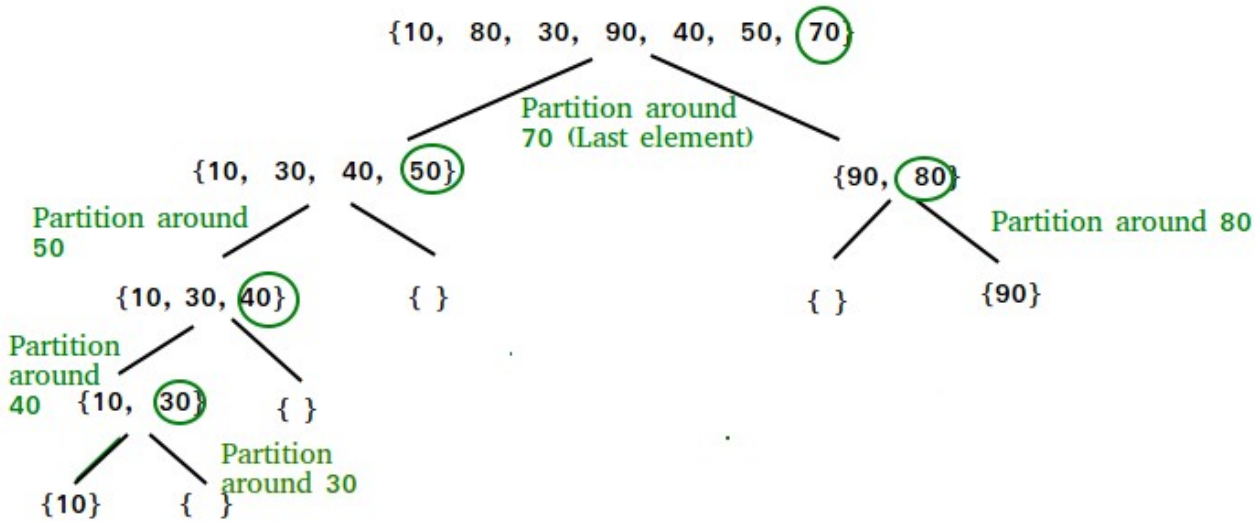
```
        System.out.println("\n\nOrdenado");
        for(int i = 0; i < vetor.length; i++){
            System.out.print(vetor[i] + " ");
        }
    }

    static void quicksort(int[] vetor, int esquerda, int direita){
        if (esquerda < direita){
            int p = particao(vetor, esquerda, direita);
            quicksort(vetor, esquerda, p);
            quicksort(vetor, p + 1, direita);
        }
    }

    static int particao(int[] vetor, int esquerda, int direita){

        int meio =  (int)(Math.random() * ((direita - esquerda) + esquerda));
        // esquerda + (int)(Math.random() * ((direita - esquerda) + 1));
        /*  int meio = (int) (esquerda + direita) / 2; */
        int pivot = vetor[meio];
        int i = esquerda - 1;
        int j = direita + 1;
        while(true){
            do{
                i++;
            }while(vetor[i] < pivot);
            do{
                j--;
            }while(vetor[j] > pivot);
            if (i >= j){
                return j;
            }
            int aux = vetor[i];
            vetor[i] = vetor[j];
            vetor[j] = aux;
        }
    }
}
```

Partição: reorganiza o vetor em valores menores que o ponto pivô a esquerda e valores maiores a direita.
Quicksort: a função reorganiza o vetor ao chamar separadamente e recursivamente enquanto os valores são passados do menor para o maior. Os vetores a esquerda são menores que o valor pivô e os valores à direita que são maiores que o mesmo pivô.

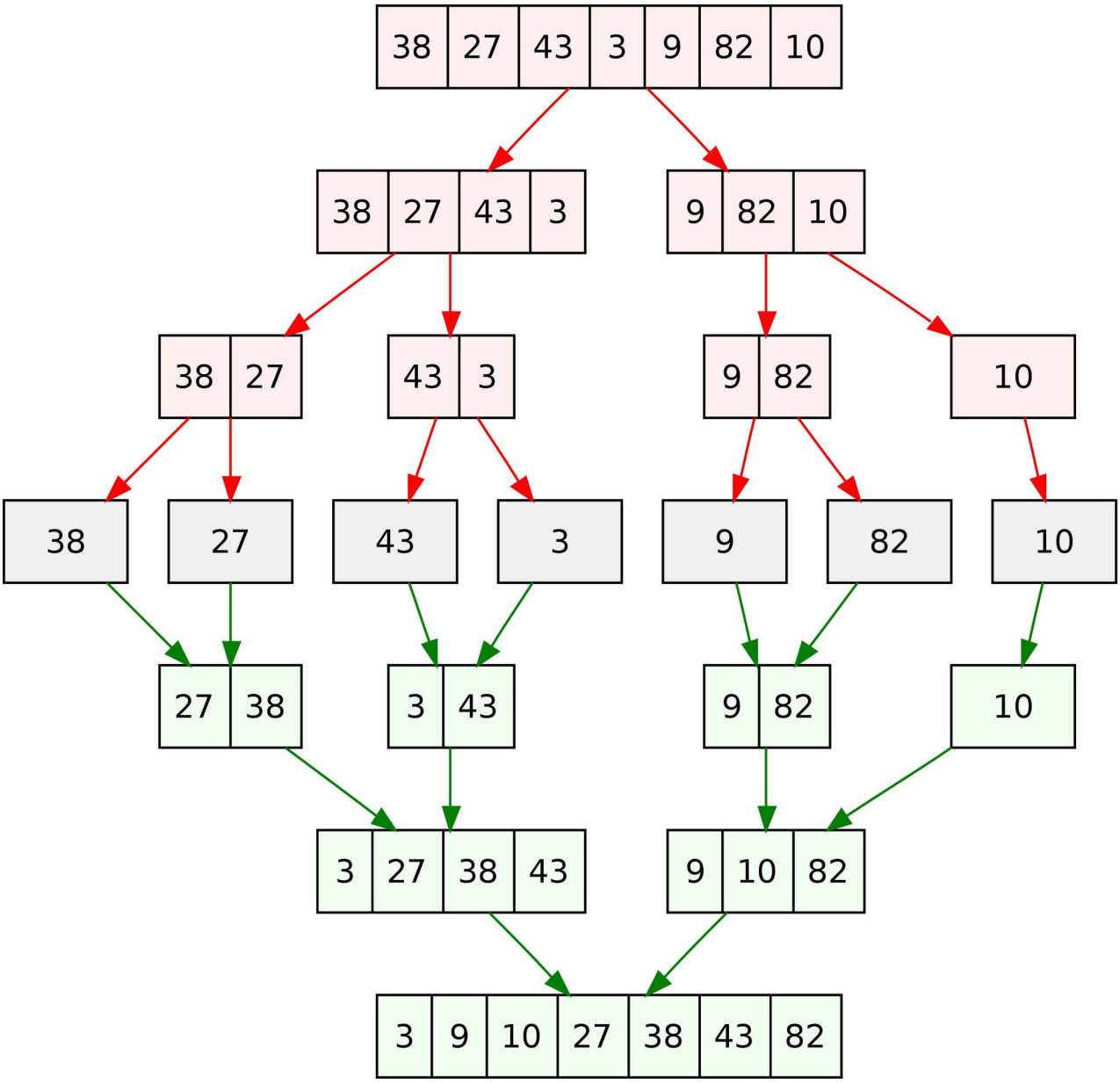
Exemplo ilustrando de forma visual como o programa funciona.



Outro exemplo visual:



Mais um exemplo visual para melhor compreensão:



Complexidade na resolução dos problemas:

- Complexidade Pior Caso: $O(n^2)$
- Complexidade Caso Médio: (nlogn)
- Complexidade Melhor Caso: (nlogn)

O QuickSort é um dos melhores algoritmos de ordenação e seu emprego é bastante comum ao utilizar métodos de ordenação incluídos no sistema.