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.

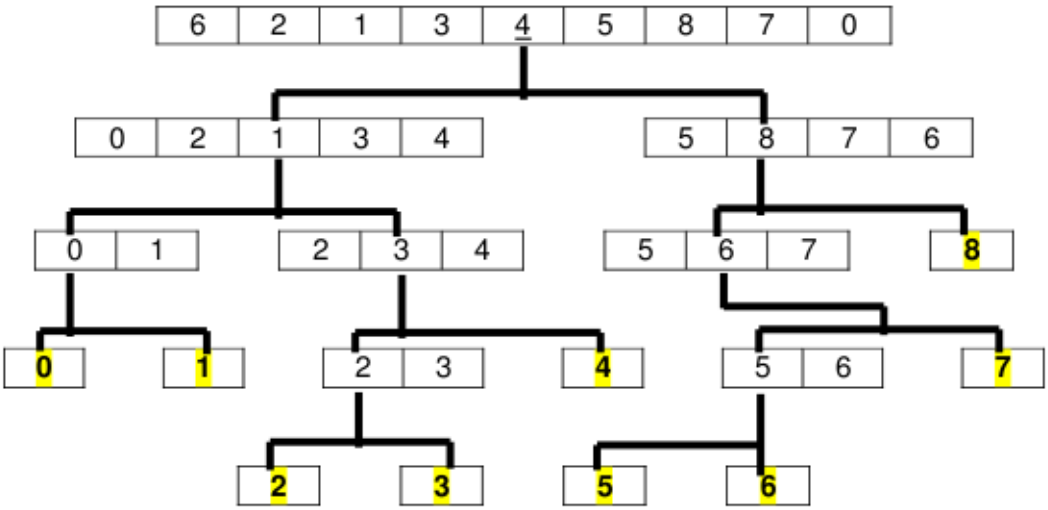
Exemplo de código:

```
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);
```

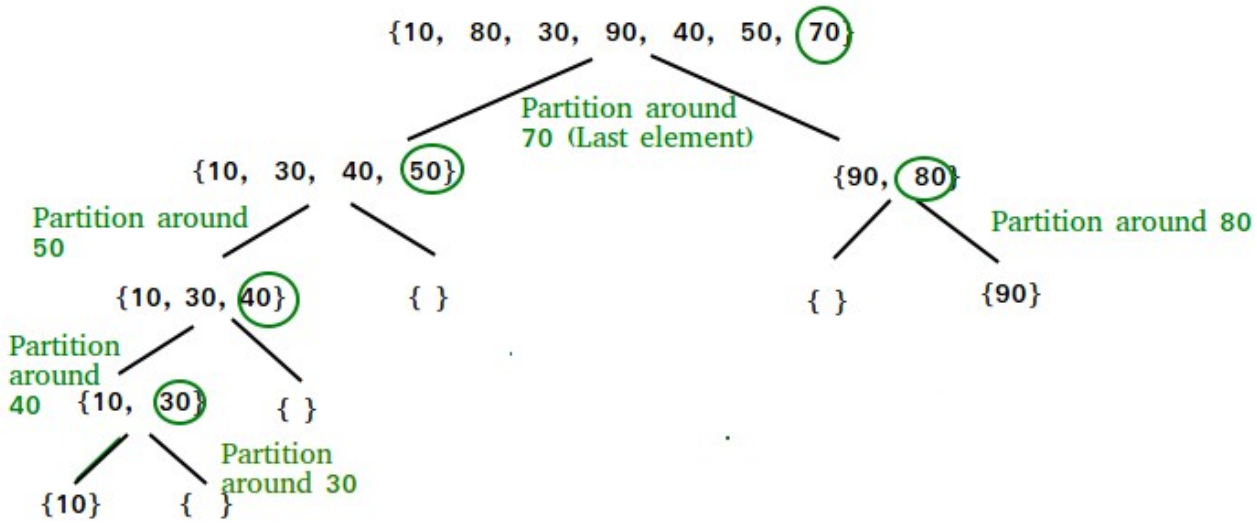
```
        System.out.println("\n\nOrdenado");
        for(int i = 0; i < vetor.length; i++){
            System.out.print(vetor[i] + " ");
        }
    }

    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.

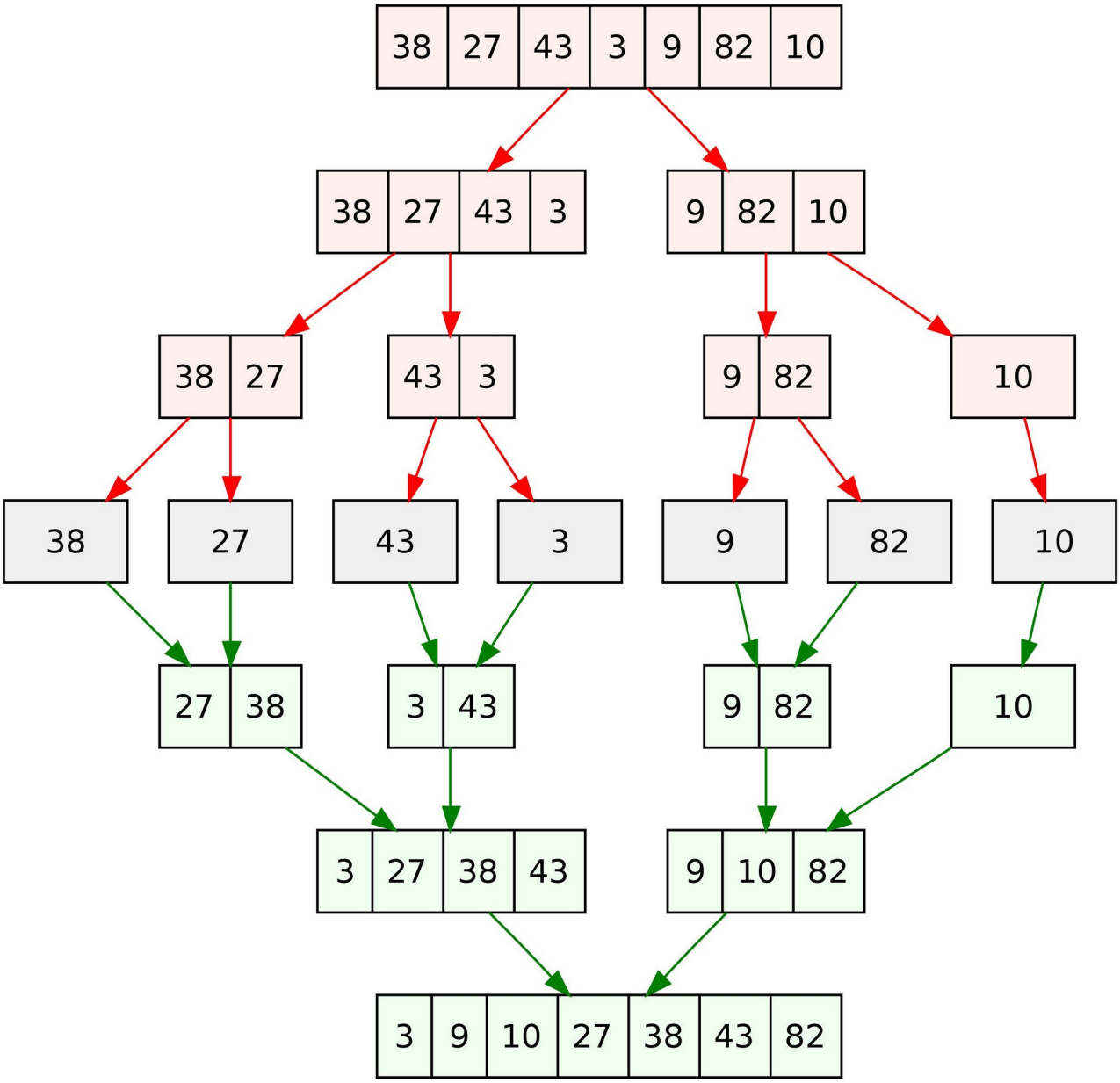
Exemplo ilustrando de forma visual como o programa funciona.



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.

Entre os algoritmos o quicksort sempre leva vantagem devido a sua rapidez. Agora entre a forma de particionar o quicksort, será que há diferenças?

Um teste foi realizado para quantificar estas diferenças. Foi realizado testes de tempo, em milisegundos, entre a execução do algoritmo com o pivô central, no início, no fim, e com a escolha aleatória. O vetor, desordenado, foi escolhido entre os valores de 10, 100, 1000, 100000, 1000000 e 100000000 unidades. Qual o método de quicksort realizará a tarefa em menor tempo?

Seguem os resultados desta contagem abaixo:

QuickSortInicio	Quantidade de elementos no array					
Tempo (ms)	10	100	1000	100000	1000000	100000000
1 Rodada	0	0	1	9	76	9485
2 Rodada	0	0	1	11	72	8961
3 Rodada	0	0	1	9	73	8977
4 Rodada	0	0	1	10	76	9032
5 Rodada	0	0	0	9	73	8948
6 Rodada	1	0	1	9	75	9089
7 Rodada	0	0	1	8	72	9027
8 Rodada	1	1	1	10	75	9387
9 Rodada	0	0	0	8	72	8813
10 Rodada	0	0	0	9	79	9437
Média Aritmética	0,20000	0,10000	0,70000	9,20000	74,30000	9115,60000
Média Geométrica	0,00000	0,00000	0,00000	9,15964	74,26805	9112,92725
Desvio Médio	0,32000	0,18000	0,42000	0,68000	1,90000	192,44000
Desvio Padrão	0,42164	0,31623	0,48305	0,91894	2,31181	233,74402
Desvio Padrão P	0,40000	0,30000	0,45826	0,87178	2,19317	221,74905

QuickSortMeio	Quantidade de elementos no array					
Tempo (ms)	10	100	1000	100000	1000000	100000000
1 Rodada	1	0	0	11	73	9489
2 Rodada	0	0	1	10	76	9478
3 Rodada	0	1	0	9	71	8953
4 Rodada	0	0	1	10	74	9407
5 Rodada	0	0	0	10	73	8977
6 Rodada	0	1	1	10	76	9477
7 Rodada	0	0	1	10	75	9072
8 Rodada	0	0	0	9	73	9045
9 Rodada	0	0	1	10	76	9457
10 Rodada	1	0	1	9	72	9392
Média Aritmética	0,20000	0,00000	0,10000	2,00000	14,50000	1888,10000
Média Geométrica	0,00000	0,00000	0,00000	9,78165	73,88041	9272,10214
Desvio Médio	0,32000	0,32000	0,48000	0,48000	1,50000	210,36000
Desvio Padrão	0,42164	0,42164	0,51640	0,63246	1,79196	230,61322
Desvio Padrão P	0,40000	0,40000	0,48990	0,60000	1,70000	218,77891

QuickSortFim	Quantidade de elementos no array					
Tempo (ms)	10	100	1000	100000	1000000	100000000
1 Rodada	0	0	1	10	72	8859
2 Rodada	0	0	1	9	70	9083
3 Rodada	0	0	1	9	72	9120
4 Rodada	0	1	0	9	72	8818
5 Rodada	0	0	0	10	72	8905
6 Rodada	0	1	0	8	71	8967
7 Rodada	0	0	1	8	70	8901
8 Rodada	0	0	0	8	71	8913
9 Rodada	0	0	0	9	70	9041
10 Rodada	0	0	0	10	70	8946
Média Aritmética	0,00000	0,00000	0,10000	2,00000	14,20000	1780,50000
Média Geométrica	0,00000	0,00000	0,00000	8,96652	70,99437	8954,81898
Desvio Médio	0,00000	0,32000	0,48000	0,60000	0,80000	77,96000
Desvio Padrão	0,00000	0,42164	0,51640	0,81650	0,94281	97,97624
Desvio Padrão P	0,00000	0,40000	0,48990	0,77460	0,89443	92,94843

QuickSortAleatório	Quantidade de elementos no array					
Tempo (ms)	10	100	1000	100000	1000000	100000000
1 Rodada	1	0	1	9	76	9106
2 Rodada	0	1	1	9	74	9230
3 Rodada	0	0	0	10	76	9406
4 Rodada	0	1	1	10	74	9394
5 Rodada	1	1	0	10	78	9307
6 Rodada	0	0	0	9	77	9418
7 Rodada	0	0	1	10	76	9335
8 Rodada	0	0	1	10	76	9261
9 Rodada	0	0	1	10	76	9189
10 Rodada	0	0	0	10	76	9156
Média Aritmética	0,10000	0,00000	0,10000	1,90000	15,20000	1826,20000
Média Geométrica	0,00000	0,00000	0,00000	9,68886	75,89148	9279,61487
Desvio Médio	0,32000	0,42000	0,48000	0,42000	0,76000	91,80000
Desvio Padrão	0,42164	0,48305	0,51640	0,48305	1,19722	109,77937
Desvio Padrão P	0,40000	0,45826	0,48990	0,45826	1,13578	104,14586

Para os maiores valores, o quicksort com o pivô no final do array, é o que realiza a tarefa em menor tempo. O quicksort do início também obteve a segunda média menor, ficando em segundo lugar geral, seguido do pivô no meio e em final o pivô na posição aleatória. Os valores não são tão grandes assim para considerar que há uma grande diferença entre a escolha do pivô. Para valores maiores que o listado, deve-se realizar novos testes.