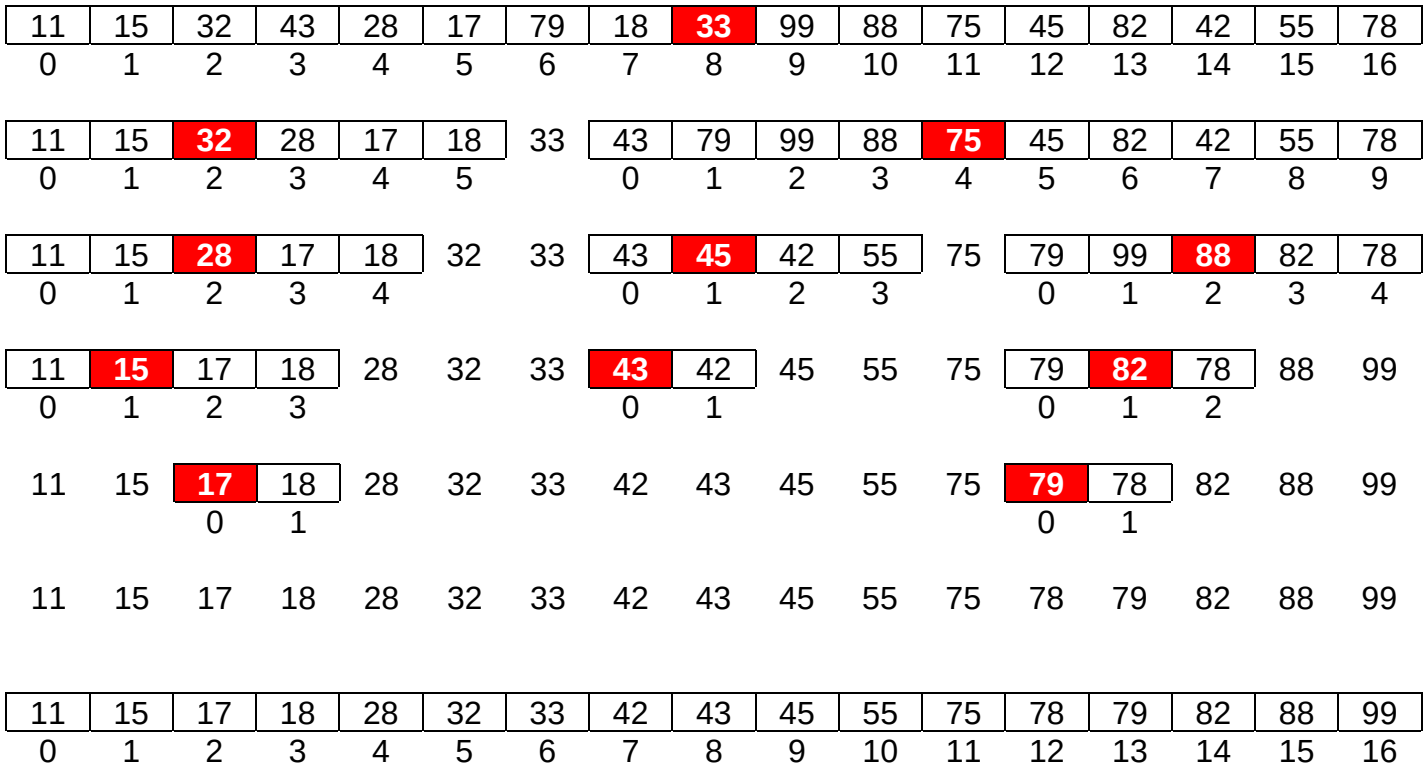
**EXERCICIOS DE ESTRUTURAS DE DADOS**
Prof. Fabio Pereira da Silva
Ana Beatriz Barbosa Alves RA: 1110482113012
Faculdade de Tecnologia da Zona Leste
Tecnologia em Análise e Desenvolvimento de Sistemas – Noite

**Lista 7 – Lista Duplamente Encadeada, Quick Sort, Merge Sort e Heap Sort**

2) Dado o algoritmo Quick Sort, explique o funcionamento de cada método abaixo e simule a sua execução para o seguinte domínio de entrada: [11, 15, 32, 43, 28, 17, 79, 18, 33, 99, 88, 75, 45, 82, 42, 55, 78], realizando a ordenação escolhendo como pivô o elemento central.

```
public static void quickSort (int vet[], int ini, int fim){
    int divisao;
    if (ini < fim) {
        divisao = particao(vet, ini, fim);
        quickSort (vet, ini, divisao-1);
        quickSort (vet, divisao+1, fim);
    }}
public static int particao (int vet[], int ini, int fim){
    int pivo = vet[ini],i = ini+1, f = fim, aux;
    while (i<=f) {
        while (i <= fim && vet[i] <= pivo)
            ++i;
        while (pivo < vet[f])
            --f;
        if (i < f){
            aux = vet[i];
            vet[i] = vet[f];
            vet[f] = aux;
            ++i;
            --f;
        }
    }
    if (ini != f){
        vet[ini] = vet[f];
        vet[f] = pivo;
    }
    return f;
}
```

O algoritmo Quicksort utiliza o paradigma Dividir para conquistar para particionar o vetor recursivamente até que tenhamos subsequências ordenadas. Trata-se de uma implementação recursiva do Quicksort utilizando o método de particionamento para quebrar o problema de ordenação em problemas de ordenação menores. No código acima, primeiramente é particionado o vetor em duas partes. Temos então a posição do pivô dentro do vetor. Em seguida é chamado o método quicksort() recursivamente passando a metade à esquerda e em seguida a metade à direita. Ao final da execução, teremos o vetor ordenado.

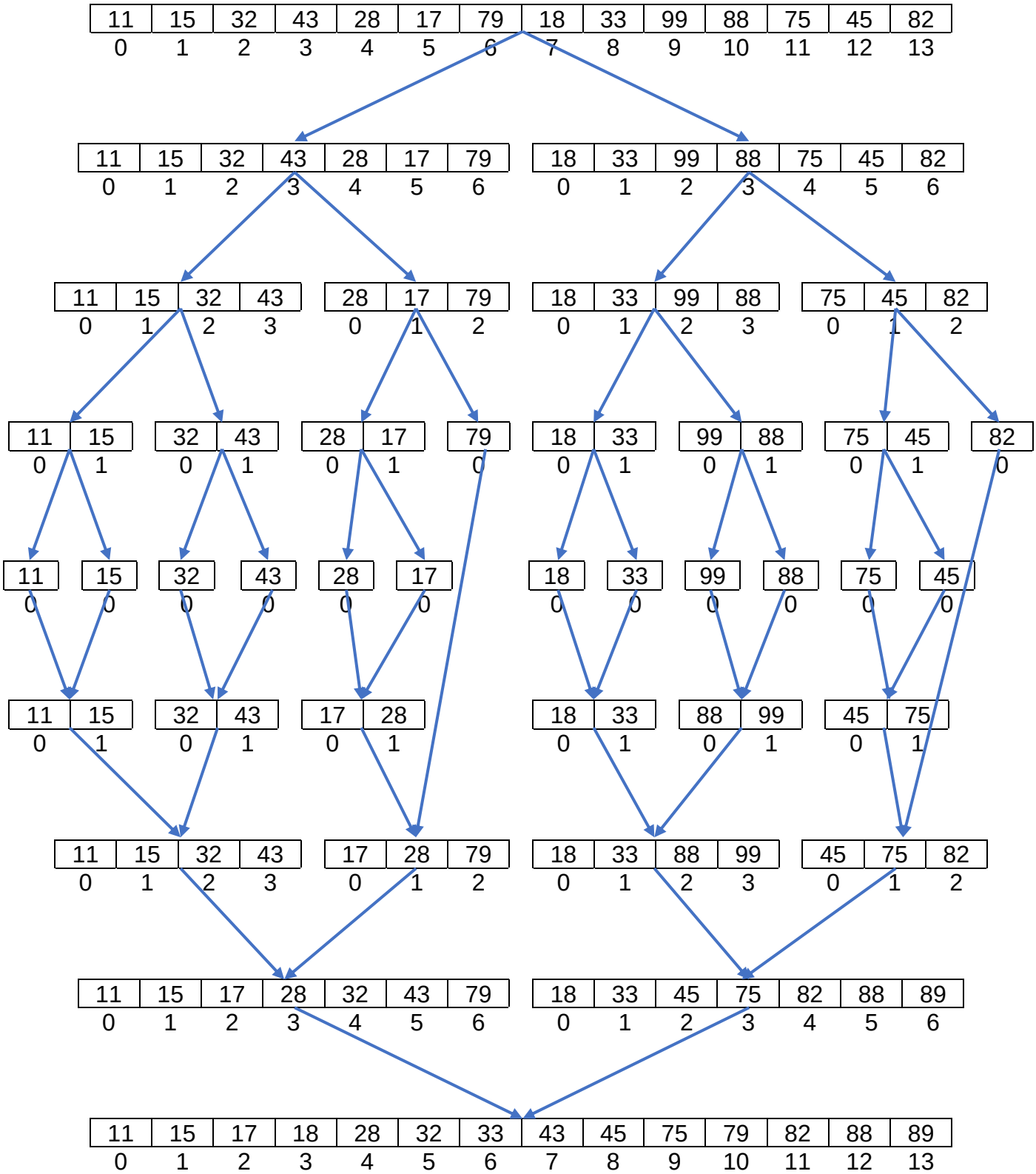

11 15 32 43 28 17 79 18 33 99 88 75 45 82 42 55 78
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
11 15 32 28 17 18 33 43 79 99 88 75 45 82 42 55 78
0 1 2 3 4 5 0 1 2 3 4 5 6 7 8 9
11 15 28 17 18 32 33 43 45 42 55 75 79 99 88 82 78
0 1 2 3 4 0 1 2 3 0 1 2 3 4
11 15 17 18 28 32 33 43 42 45 55 75 79 82 78 88 99
0 1 2 3 0 1 0 1 2
11 15 17 18 28 32 33 42 43 45 55 75 79 78 82 88 99
0 1 0 1
11 15 17 18 28 32 33 42 43 45 55 75 78 79 82 88 99
11 15 17 18 28 32 33 42 43 45 55 75 78 79 82 88 99
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

3) Dado o algoritmo Merge Sort, explique o funcionamento de cada método abaixo e simule a sua execução para o seguinte domínio de entrada: [11, 15, 32, 43, 28, 17, 79, 18, 33, 99, 88, 75, 45, 82].

```
public static void mergeSortRecursivo(int lista[], int inicio, int fim){
    if (inicio < fim){
        int meio = (inicio + fim) / 2;
        mergeSortRecursivo(lista,inicio, meio);
        mergeSortRecursivo(lista,meio + 1, fim);
        mesclar(lista,inicio, meio, meio+1,fim);
    }
}

public static void mesclar(int lista[], int inicioA, int fimA,
        int inicioB, int fimB){
    int i1 = inicioA;
    int i2 = inicioB;
    int iaux = inicioA;
    int aux[] = new int[lista.length];
    while (i1 <= fimA && i2 <= fimB){
        if(lista[i1] <= lista[i2])
            aux[iaux++]=lista[i1++];
        else
            aux[iaux++]=lista[i2++];
    }
    while (i1 <=fimA)
        aux[iaux++]=lista[i1++];
    while (i2 <=fimB)
        aux[iaux++]=lista[i2++];
    for (int i=inicioA;i<=fimB;i++)
        lista[i] = aux[i];
}
```

A ideia por trás do Mergesort é simples. Para um vetor A de n números, devemos dividir o vetor em duas metades, ordenar recursivamente cada metade e mesclar as duas metades do vetor, isto é, combinar as duas metades para formar um único arranjo. O método mergeSortRecursivo simplesmente divide o vetor de entrada em duas metades e faz a chamda do procedimento mesclar. O procedimento mesclar organiza e agrupada as metades novamente.

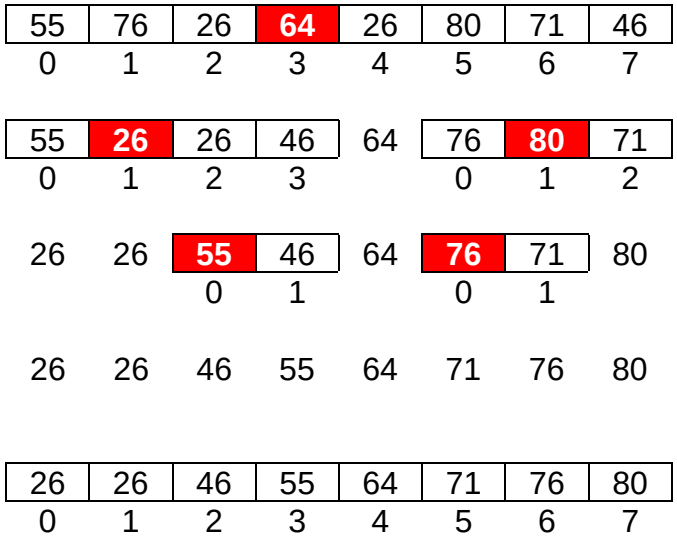

11 15 32 43 28 17 79 18 33 99 88 75 45 82
0 1 2 3 4 5 6 7 8 9 10 11 12 13
11 15 32 43 28 17 79
0 1 2 3 4 5 6
18 33 99 88 75 45 82
0 1 2 3 4 5 6
11 15 32 43
0 1 2 3
28 17 79
0 1 2
18 33 99 88
0 1 2 3
75 45 82
0 1 2
11 15
0 1
32 43
0 1
28 17
0 1
79
0
18 33
0 1
99 88
0 1
75 45
0 1
82
0
11
0
15
0
32
0
43
0
28
0
17
0
18
0
33
0
99
0
88
0
75
0
45
0
11 15
0 1
32 43
0 1
17 28
0 1
18 33
0 1
88 99
0 1
45 75
0 1
11 15 32 43
0 1 2 3
17 28 79
0 1 2
18 33 88 99
0 1 2 3
45 75 82
0 1 2
11 15 17 28 32 43 79
0 1 2 3 4 5 6
18 33 45 75 82 88 89
0 1 2 3 4 5 6
11 15 17 18 28 32 33 43 45 75 79 82 88 89
0 1 2 3 4 5 6 7 8 9 10 11 12 13

4) Explique o funcionamento dos algoritmos de ordenação Quick Sort, Merge Sort e Heap Sort, detalhe as principais diferenças entre os três algoritmos de ordenação e apresente um exemplo de teste de mesa para simulação de cada um dos três algoritmos em um conjunto de entrada com no mínimo 8 elementos.

- Quick Sort

É um algoritmo de comparação que emprega a estratégia de “divisão e conquista”. A ideia básica é dividir sua lista de entrada em duas sub-listas a partir de um pivô, para em seguida realizar o mesmo procedimento nas duas listas menores até uma lista unitária. Os problemas menores são ordenados independentemente e os resultados são combinados para produzir a solução final.

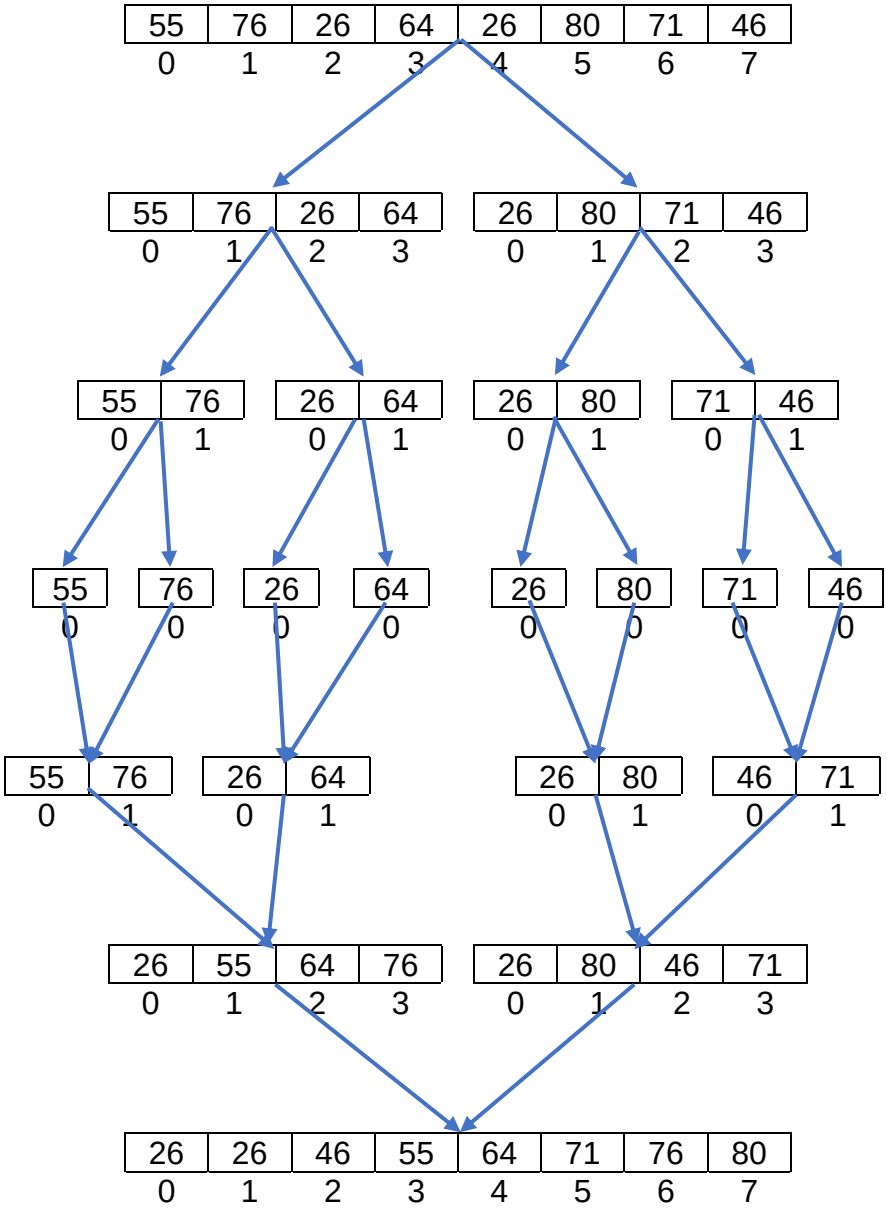
[55, 76, 26, 64, 26, 80, 71, 46]



- Merge Sort

Esse algoritmo divide o problema em pedaços menores, resolve cada pedaço e depois junta (mescla) os resultados. O vetor será dividido em duas partes iguais, que serão cada uma divididas em duas partes, e assim até ficar um ou dois elementos. Para juntar as partes ordenadas os dois elementos de cada parte são separados e o menor deles é selecionado e retirado de sua parte. Em seguida os menores entre os restantes são comparados e assim se prossegue até ter toda a sequência ordenada.

[55, 76, 26, 64, 26, 80, 71, 46]

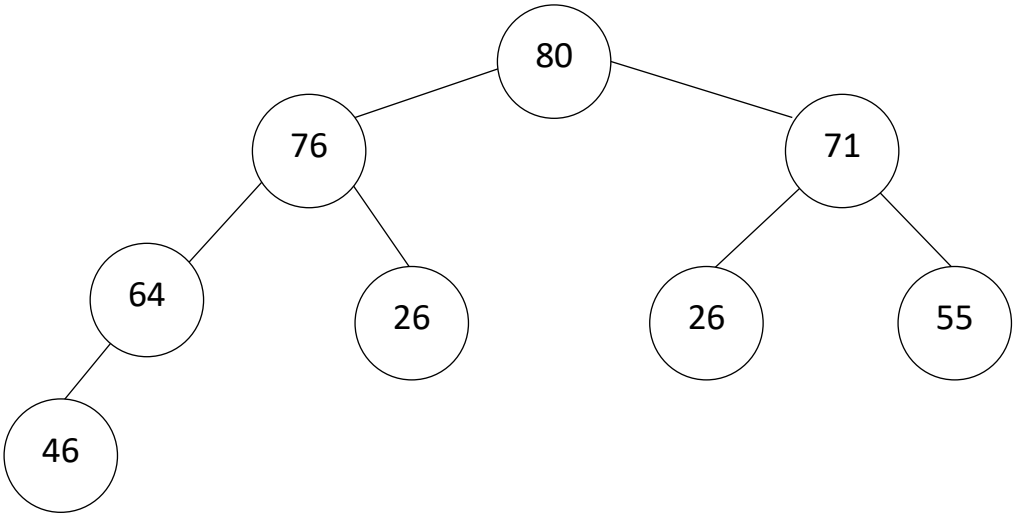


Heap Sort

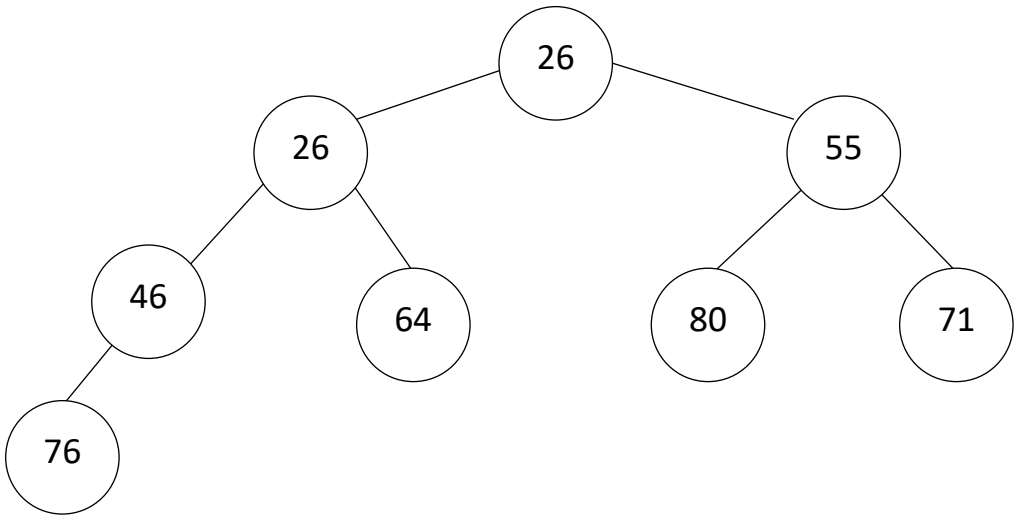
O heapsort utiliza uma estrutura de dados chamada heap binário para ordenar os elementos a medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada. Um heap binário é uma árvore binária mantida na forma de um vetor.

[55, 76, 26, 64, 26, 80, 71, 46]

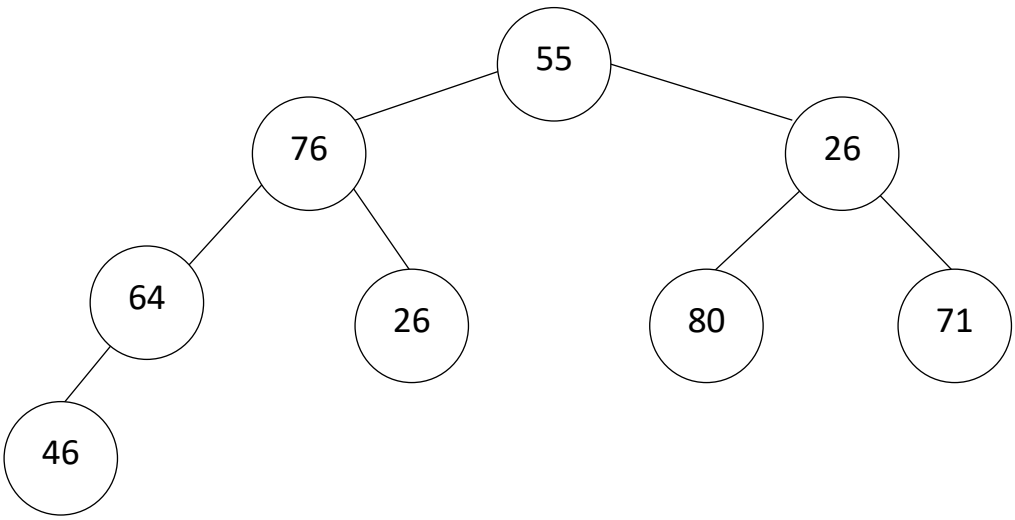
Simulação do procedimento de inserção no heap sort de cada elemento, considerando o método de ordenação no heap máximo

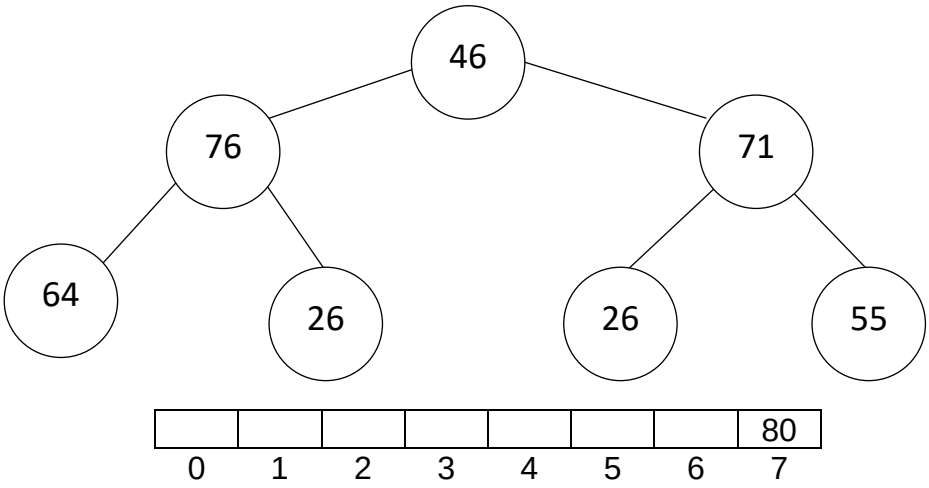



Simulação do procedimento de inserção no heap sort de cada elemento, considerando o método de ordenação no heap minimo

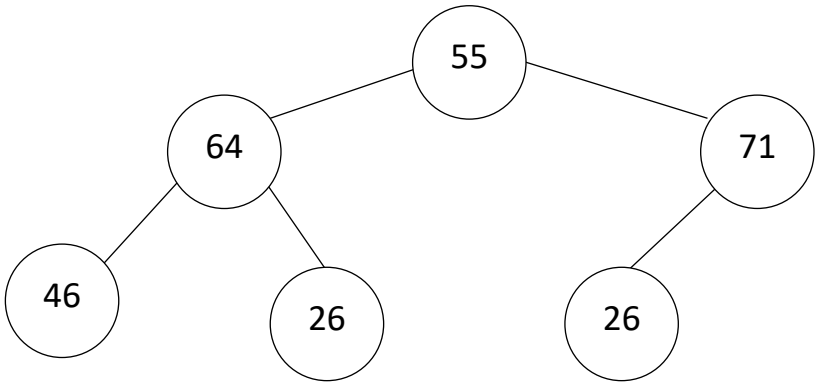


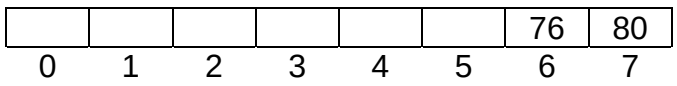

---

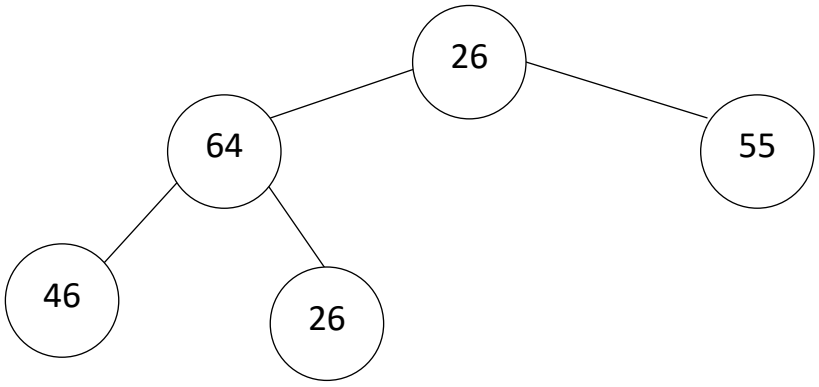


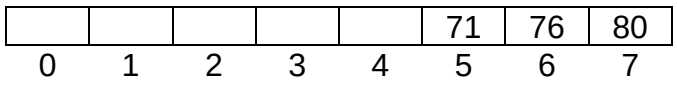

46
76
71
64
26
26
55

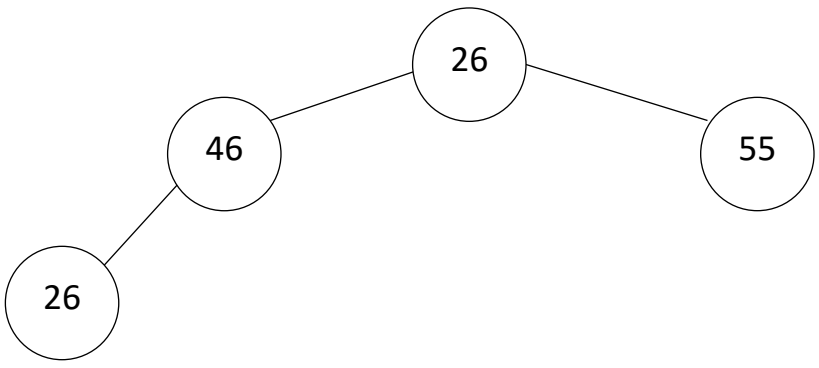

80
0 1 2 3 4 5 6 7

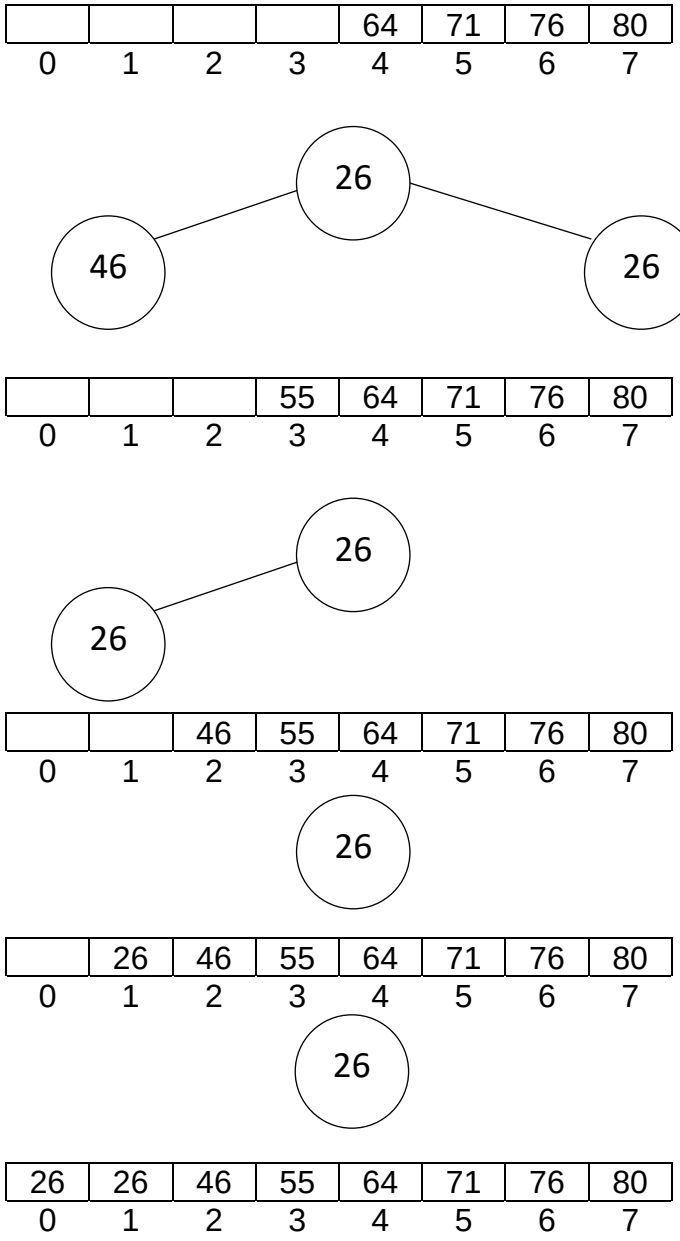

55
64
71
46
26
26


76
80
0 1 2 3 4 5 6 7


26
64
55
46
26


71
76
80
0 1 2 3 4 5 6 7


26
46
55
26




5) Explique qual algoritmo de ordenação se aplica a afirmação abaixo, justifique sua resposta. “Método de Ordenação que utiliza-se do método da divisão e conquista para ordenação do vetor. Em sua técnica, escolhe um elemento denominado de pivô (um dos elementos a serem ordenados) e separa os elementos em 2 partes, de modo que os elementos menores que o pivô ficam à esquerda e os elementos maiores que o pivô ficam à direita. Esse processo é repetido recursivamente até que todos os elementos estejam ordenados.”

R: Trata-se de um QuickSort, esse algoritmo é um método de ordenação muito rápido e eficiente. Ele baseia-se na técnica "dividir e conquistar", onde a ideia é reduzir um problema em problemas menores, resolver cada um destes subproblemas e combinar as soluções parciais para obter a solução do problema original.

6) Realize um resumo do artigo “Algoritmos de Ordenação: Um Estudo Comparativo”, disponível no Link abaixo: https://periodicos.ufersa.edu.br/index.php/ecop/article/view/7082/6540

**Resumo**

**Algoritmos de Ordenação: Um Estudo Comparativo**

Este artigo tem como objetivo mostrar os experimentos e resultados produzidos por meio de algoritmos de ordenação (Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort e Shell Sort) e comparar sua eficiência em diferentes situações.

O Bubble Sort é é considerada o algoritmo mais simples, mas o menos eficiente porque exibe o pior desempenho e realiza um grande número de comparações e trocas. Seu uso é apenas para fins educacionais.

O Insertion Sort é semelhante ao Bubble Sort, mas organiza os elementos um a um na posição correta, em que o lado direito do elemento selecionado sempre tem um valor mais alto e o lado esquerdo do elemento selecionado sempre tem um valor mais baixo. A sua maior vantagem é que o número de comparações é menor, mas a desvantagem é que tem mais trocas.

O Selection Sort encontrará o menor elemento e o colocará no início da estrutura linear, então encontrará o segundo menor elemento e o colocará na segunda posição da estrutura, repetindo este processo até que a sequência esteja completamente classificada. Ao contrário do Insertion Sort, o Selection Sort realiza um grande número de comparações, mas, por outro lado, realiza uma pequena quantidade de troca.

O Merge Sort decompõe a estrutura linear em várias partes até que sejam indivisíveis e, em seguida, reconstrói recursivamente a estrutura de maneira intercalada e ordenada. Devido à recursão, o algoritmo é mais eficaz em estruturas lineares aleatórias, e quando usado em estruturas pequenas ou pré-determinadas, é realizada trocas desnecessárias e ocorre o aumento do tempo de processamento.

O Quicksort divide a estrutura linear em duas partes por meio do pivô. Uma parte contém elementos menores que o pivô e a outra parte contém elementos maiores que o pivô. Repete o processo de seleção de pivôs e separação de conteúdo em conteúdo menor e maior do que os dados selecionados até que a estrutura seja classificada. Quando a velocidade de ordenação é priorizada, este algoritmo provou ser uma boa escolha, mesmo se muitos recursos forem usados.

O Shell Sort é considerada uma versão aprimorada do Insertion Sort, que usa um valor (que representa a distância entre os dados) para reorganizar os elementos de uma determinada estrutura. No Insertion Sort, cada elemento é comparado com elementos vizinhos, enquanto que no Shell Sort ele pode ser comparado com um elemento que esteja mais distante. O algoritmo apresentou os melhores resultados durante o experimento, principalmente em grandes estruturas, desorganizadas e quando utilizado em situações onde a escrita é quase desnecessária.