CLASSIFICAÇÃO DE DADOS

No nosso dia-a-dia nem nos damos conta da importância da classificação das informações na hora de fazermos uma busca. Imagine se você fosse fazer uma pesquisa no catálogo telefônico ou em um dicionário se os dados não seguissem algum tipo de classificação. Daí, notamos a grande importância da classificação para localizarmos uma determinada informação.

Classificação de dados é o processo pelo qual é determinada a ordem que as entradas de uma tabela (os registro de um arquivo) serão apresentadas de forma que obedeça a uma organização previamente determinada por um ou mais campos. O campo pelo qual o arquivo é ordenado é chamado de chave de classificação.

O ambiente de classificação é determinado pelo meio onde estão os dados a serem classificados. Uma classificação é considerada interna se os registros que ela classificar estiverem na memória principal, e externa se alguns dos registros que ela classificar estiverem no armazenamento auxiliar.

A classificação interna apresenta como principais características classificar apenas o que se encontra na memória principal, gastar o mesmo tempo de acesso para buscar qualquer dos endereços e o tempo médio de acesso não é afetado pela seqüência dos dados.

Na classificação externa, os dados são transferidos em blocos para a memória principal para, só então, serem manipulados. Esses dados transferidos num acesso a disco influenciarão na eficiência do processamento e os dados são manipulados bloco a bloco.

É importante notar que os métodos de classificação externa envolvem a aplicação de métodos de classificação interna, tomando a cada vez um subconjunto de dados a classificar. Por isso, nos deteremos ao estudo dos métodos de Classificação Interna.

Há diferentes formas de apresentação do resultado de classificação, as três formas clássicas são: contiguidade física (as entradas são fisicamente rearranjadas, de forma que no final da classificação a ordem lógica é igual a ordem física dos dados), vetor indireto de ordenação – VIO (as entradas são mantidas fisicamente com as mesmas posições originais, sendo a seqüência ordenada determinada por um vetor[VIO] que é gerado durante o processo de classificação e que possui a seqüência dos endereços ordenada pelos valores classificados da tabela associada) e encadeamento (as entradas não sofrem alterações em suas posições físicas, então, é formada uma lista encadeada que inclui todas as entradas da tabela ordenada pelo valor da chave de classificação).

Métodos de Classificação Interna

Os métodos de classificação interna são divididos em cinco grupos de acordo com a técnica empregada, são eles:

1. Classificação por Inserção (inserção direta e shell)
2. Classificação por Troca (bubblesort e quicksort)
3. Classificação por Seleção (seleção direta e heapsort)
4. Classificação por Distribuição (distribuição de chaves e radixsort)
5. Classificação por Intercalação (mergesort)

1 – Classificação por Inserção

Na classificação por inserção, a classificação de um conjunto de registros é feita inserindo registros num sub-arquivo classificado anteriormente, ou seja, a inserção de um elemento é feita na posição correta dentro de um sub-arquivo classificado.

Método da Inserção Direta

Dentre os métodos de inserção, o método da Inserção Direta é o mais simples, porém, é o mais rápido entre os outros métodos considerados básicos – BubbleSort e Seleção Direta, que serão estudados posteriormente. Ele deve ser utilizado para pequenos conjuntos de dados devido a sua baixa eficiência.

A principal característica deste método consiste em ordenarmos nosso arquivo utilizando um sub-arquivo ordenado localizado em seu início, e a cada novo passo, acrescentamos a este sub-arquivo mais um elemento na sua posição correta, até chegar ao último elemento do arquivo gerando um arquivo final ordenado. Como este é um método difícil de ser descrito, faremos a análise do algoritmo através do exemplo.

Iniciando com um arquivo qualquer desordenado que segue:

370	250	210	330	190	230

Consideraremos o seu primeiro elemento como se ele fosse um sub-arquivo ordenado:

370	250	210	330	190	230

O passo seguinte é inserir o próximo elemento ao nosso sub-arquivo ordenado, no nosso exemplo o número 250, que é copiado para uma variável auxiliar. Caminharemos pelo o sub-arquivo a partir do último elemento para o primeiro. Assim poderemos encontrar a posição correta da nossa variável auxiliar dentro do sub-arquivo.

Notamos no nosso exemplo que a variável auxiliar, 250, é menor que o último elemento do nosso sub-arquivo ordenado (o nosso sub-arquivo só possui por enquanto um elemento, o número 370). O número 370 é então copiado uma posição para a direita.

370	370	210	330	190	230

Nossa variável auxiliar com o número 250, é colocada emsua posição correta no sub-arquivo ordenado.

250	370	210	330	190	230

Vale notar que nosso sub-arquivo já possui dois elementos e está ordenado. Todo esse processo é repetido para o elemento seguinte, o terceiro elemento. O próximo elemento, 210, é copiado na nossa variável auxiliar e será inserido no sub-arquivo ordenado. Então faremos a comparação da variável auxiliar com os elementos de nosso sub-arquivo, sempre a partir do último elemento para o primeiro.

Na primeira comparação notamos que a variável auxiliar é menor que o último elemento de nosso sub-arquivo. Assim, copiamos este elemento para a direita e continuamos com nossas comparações.

250	370	370	330	190	230

Novamente, a nossa variável auxiliar é menor que o elemento do sub-arquivo que estamos comparando. Por isso ele deve ser copiado para a direita, abrindo espaço para que a variável auxiliar seja colocada em sua posição correta.

250	250	370	330	190	230

A variável auxiliar com o número 210, é colocada em sua posição correta.

210	250	370	330	190	230

Veja que nosso sub-arquivo ordenado possui 3 elementos. Continuaremos o processo de ordenação copiando mais uma vez o elemento imediatamente superior, 330, ao nosso sub-arquivo na variável auxiliar. Vamos, então, comparar essa variável auxiliar com os elementos do nosso sub-arquivo a partir do último elemento.

Observe que nossa variável auxiliar é menor que o elemento que está sendo comparado no nosso sub-arquivo. Então ele deve ser copiado para a direita para que continuemos com nossas comparações.

210	250	370	370	190	230

Nessa comparação notamos que a variável auxiliar é maior que o elemento do sub-arquivo que estamos comparando (250<330). Portanto, encontramos a posição correta para a inserção de nossa variável auxiliar. Vamos inseri-la na sua posição correta, isto quer dizer que vamos copiá-la para o elemento imediatamente superior ao elemento que estava sendo comparado.

210	250	330	370	190	230

Note que nosso sub-arquivo possui quatro elementos ordenados. Iremos, então, repetir o processo de ordenação, copiando o próximo elemento do arquivo para uma variável auxiliar. Repetiremos as comparações de trás para frente até encontrarmos a posição correta para inserirmos o nosso elemento. Sempre que necessário, copiaremos na sua posição imediatamente à direita os elementos que forem maiores que a variável auxiliar até que se encontre um elemento menor ou até que se chegue ao início do arquivo que indicará a posição procurada. O passo-a-passo é mostrado a seguir.

210	250	330	370	190	230

Aux = 190

210	250	330	370	370	230

210	250	330	330	370	230

210	250	250	330	370	230

210	210	250	330	370	230

190	210	250	330	370	230

Mais uma vez chegamos a um sub-arquivo ordenado. Os passos seguintes mostram a última etapa de ordenação do nosso arquivo.

190	210	250	330	370	230

Aux = 230

190	210	250	330	370	370

190	210	250	330	330	370

190	210	250	250	330	370

190	210	230	250	330	370

Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

Método dos Incrementos Decrescentes (Método de Shell)

Esse método foi proposto por Ronald Shell em 1959. Diferente do Método de Inserção Direta que considera apenas um segmento, nesse método vários segmentos são considerados e feita uma inserção seletiva do elemento em um dos segmentos. Os segmentos são formados pegando-se os elementos que se encontram nas posições múltiplas de um determinado valor chamado incremento. Veja o exemplo a seguir para um incremento igual a 4.

1º Segmento: Array[0], Array[4], Array[8], ...
2º Segmento: Array[1], Array[5], Array[9], ...
3º Segmento: Array[2], Array[6], Array[10], ...
4º Segmento: Array[3], Array[7], Array[11], ...

A cada passo os segmentos são classificados isoladamente por inserção direta e o incremento para o passo subsequente passa a ser menor que no passo anterior, podendo ser a metade do anterior. A cada passo, o número de segmentos passa a ser menor, porém a quantidade de elementos em cada seguimento passa a ser maior. Após o passo onde o incremento foi igual a 1, o vetor estará totalmente ordenado! Nesse caso é feita uma classificação por inserção direta, porém, o arquivo estará quase ordenado.

Dado o exemplo abaixo, vamos seguir a idéia do método Shell.

18	15	12	20	5	10	25	8

Acima está o nosso arquivo que queremos ordenar. Nosso primeiro passo é dividir o arquivo em segmentos definidos pelos elementos do arquivo original que estejam nas posições múltiplas do incremento definido, no nosso caso 4.

18	15	12	20	5	10	25	8
1º seg	2º seg	3º seg	4º seg	1º seg	2º seg	3º seg	4º seg

Em seguida ordenaremos cada seguimento, utilizando o método da inserção direta. Observe que inicialmente temos muitos segmentos pequenos e que a cada passo teremos menos segmentos, porém, com mais elementos e eles estarão a cada passo mais próximo da ordenação final. Veja abaixo o resultado após o primeiro passo.

5	10	12	8	18	15	25	20
1º seg	2º seg	3º seg	4º seg	1º seg	2º seg	3º seg	4º seg

Veja que cada segmento se encontra ordenado. O passo seguinte refaz os seguimentos e todo o processo é repetido de forma similar, porém, o incremento passa a ser a metade do valor anterior. Sendo agora o nosso incremento igual a 2.

5	10	12	8	18	15	25	20
1º seg	2º seg	1º seg	2º seg	1º seg	2º seg	1º seg	2º seg

Após a ordenação dos seguimentos chegamos ao seguinte arquivo:

5	8	12	10	18	15	25	20
1º seg	2º seg	1º seg	2º seg	1º seg	2º seg	1º seg	2º seg

Mais uma vez, veja que cada segmento se encontra ordenado. O passo seguinte refaz os seguimentos e todo o processo é repetido de forma similar, porém, como o nosso incremento passa a ser igual a 1 será utilizado o método da inserção direta. Va le lembrar que o arquivo está quase ordenado!

5	8	12	10	18	15	25	20

Após a ordenação pela inserção direta chegamos ao seguinte arquivo:

5	8	10	12	15	18	20	25

Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

2 – Classificação por Troca

Na classificação por troca, a classificação de um conjunto de registros é feita através de comparações entre os elementos e trocas sucessivas desses elementos entre posições no arquivo.

Método BubbleSort

É certamente o método mais simples, de entendimento e programação mais fáceis. O Bubblesort é um dos mais conhecidos e utilizados métodos de ordenação de arquivos, principalmente por principiantes em programação e professores que procuram desenvolver o raciocínio nas disciplinas de algoritmos e programação.

Entretanto, o Bubblesort não apresenta um desempenho satisfatório se comparado com os demais.

O Bubblesort se baseia em trocas de valores entre posições consecutivas, levando os valores mais altos (ou mais baixos) para o final do arquivo.

Vamos acompanhar o exemplo para colocar o arquivo em ordem crescente de valores.

Dado o exemplo abaixo com o arquivo inicial que desejamos ordenar, vamos seguir a idéia do método Bubblesort.

18	15	20	12

Iniciaremos comparando os dois primeiros elementos (o primeiro com o segundo). Como eles estão desordenados, então, trocamos as suas posições.

15	18	20	12

Comparamos agora o segundo com o terceiro. Como eles estão ordenados, então, passaremos para a comparação seguinte (o terceiro com o quarto). Como eles estão desordenados, então, trocamos as suas posições.

15	18	12	20

Chegamos agora ao final do arquivo! Note que após essa primeira fase de comparações e trocas o maior elemento, 20, está na sua posição correta e final.

Na próxima fase, todo o processo será repetido, porém com a diferença que agora as comparações e trocas não serão mais feitas até o final do arquivo, mais sim até o último número que antecede aquele que chegou a sua posição correta no nosso caso o penúltimo elemento. Vamos mais uma vez iniciar o processo, comparando os dois primeiros elementos do arquivo. Verificamos que como os valores estão ordenados, não será necessária a troca de posições. Continuaremos as comparações e notamos que precisamos trocar o terceiro pelo segundo elemento.

15	12	18	20

Note que as comparações e trocas dessa fase chegaram ao final, uma vez que o último elemento já está em sua posição correta e que levamos o segundo maior elemento para a sua posição correta. Vamos mais uma vez iniciar o processo, comparando os dois primeiros elementos do arquivo. Verificamos que como os valores não estão ordenados, será necessária a troca de posições.

12	15	18	20

Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

Método QuickSort

O método QuickSort, também chamado Método por Troca de Partição é o método que apresenta melhor desempenho se comparado com os métodos estudados até o momento. Ele se baseia na idéia “dividir para conquistar”, ou seja, é mais rápido ordenar dois arquivos menores do que um grande.

O particionamento do arquivo é feito através da escolha de um determinado elemento, chamado pivô, e em seguida todos os elementos menores que o pivô ficam a sua esquerda enquanto aqueles maiores ficam do seu lado direito. O processo continua de forma recursiva ordenando-se o sub-arquivo particionado do lado esquerdo e o sub-arquivo particionado do lado direito.

A escolha do pivô pode ser definida usando uma estratégia. Algumas das estratégias mais utilizadas são: escolha do primeiro elemento do arquivo, escolha do elemento mediano do arquivo, escolha do elemento mediano entre o primeiro, o do meio e o último elemento do arquivo, etc.. A estratégia de escolha do pivô adotada influenciará na eficiência do algoritmo.

Dado o exemplo abaixo, vamos seguir a idéia do método Quicksort.

10	20	12	5	8	15

Acima está o nosso arquivo que queremos ordenar. Nosso primeiro passo é escolher o pivô (através de uma das técnicas), no nosso caso escolheremos o primeiro elemento do arquivo, 10. Seguiremos, colocando um apontador no início(que pesquisa os elementos maiores que o pivô e será chamado de menor) e outro no final (que pesquisa os elementos menores que o pivô e será chamado de maior) do arquivo.

10	20	12	5	8	15
menor					maior

Incrementaremos menor até acharmos um elemento maior que pivô.

10	20	12	5	8	15
	menor				maior

Incrementaremos maior até acharmos um elemento menor que pivô

10	20	12	5	8	15
	menor			maior	

Como a posição para a qual o apontador maior é maior que a posição de menor, então faremos a troca entre os valores.

10	8	12	5	20	15
	menor			maior	

Mais uma vez, incrementaremos menor até acharmos um elemento maior que pivô.

10	8	12	5	20	15
		menor		maior	

Incrementaremos maior até acharmos um elemento menor que pivô

10	8	12	5	20	15
		menor	maior		

Como a posição para a qual o apontador maior é maior que a posição de menor, então faremos a troca entre os valores.

10	8	5	12	20	15
		menor	maior		

Mais uma vez, incrementaremos menor até acharmos um elemento maior que pivô.

10	8	5	12	20	15
			maior menor		

Incrementaremos maior até acharmos um elemento menor que pivô

10	8	5	12	20	15
		maior	menor		

Como a posição para a qual o apontador maior é menor que a posição de menor, então não faremos a troca entre os valores, mas sim trocaremos o valor de pivô pelo valor de maior.

5	8	10	12	20	15
		maior	menor		

Agora, reiniciaremos todo o processo para ordenar o sub-arquivo a esquerda (5,8) e para ordenar o sub-arquivo à direita (12, 20, 15) do nosso pivô (10). Ao final de todas as ordenações dos sub-arquivos gerados, teremos o nosso arquivo totalmente ordenado.

3 – Classificação por Seleção

Na classificação por seleção, a classificação de um conjunto de registros é feita através de sucessivas seleções do menor elemento de um arquivo ou sub-arquivo, durante o processo o menor valor encontrado é colocado na sua posição correta final no arquivo e o processo é repetido para o sub-arquivo que contém os elementos que ainda não foram selecionados.

Método Seleção Direta

O metódo de Seleção Direta possui melhor desempenho que o método Bubblesort, porém, só deve ser utilizado em pequenos arquivos. Assim como o Bubblesort, esse método também é bastante usado durante o estudo de algoritmos e desenvolvimento do raciocínio lógico.

A Seleção Direta se baseia na fixação da primeira posição e na busca, por todo o arquivo, do menor elemento quando então é feita a troca dos valores. No final, teremos o menor valor (ou o maior, conforme a comparação) na primeira posição do arquivo.

Este primeiro passo nos garante que o menor elemento fique na primeira posição. Continuamos, assim, a buscar os demais elementos, comparando-os com a segunda posição do arquivo (já desconsiderando a primeira posição, que foi anteriormente ordenada em relação ao arquivo como um todo).

Dado o exemplo abaixo, vamos seguir a idéia do método Seleção Direta.

10	20	12	5	8	15

Vamos fixar a primeira posição e, utilizando uma variável auxiliar, procurar o menor elemento do arquivo e trocá-lo pelo elemento que ocupava a primeira posição.

5	20	12	10	8	15

Agora, fixaremos a segunda posição e repetiremos o processo acima.

5	8	12	10	20	15

Passaremos para a terceira posição e repetiremos o processo. Acompanhe os passos para conclusão do algoritmo.

5	8	10	12	20	15

5	8	10	12	20	15

5	8	10	12	15	20

Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

Método HeapSort

Possui a mesma eficiência do Quicksort para a maioria dos casos. É dividido em duas etapas:

1. Preparação do monte inicial (heap): organiza o arquivo a ser ordenado em forma de árvore binária (não árvore de busca binária) na qual o valor do nó terá que ser obrigatoriamente maior que seus sucessores;

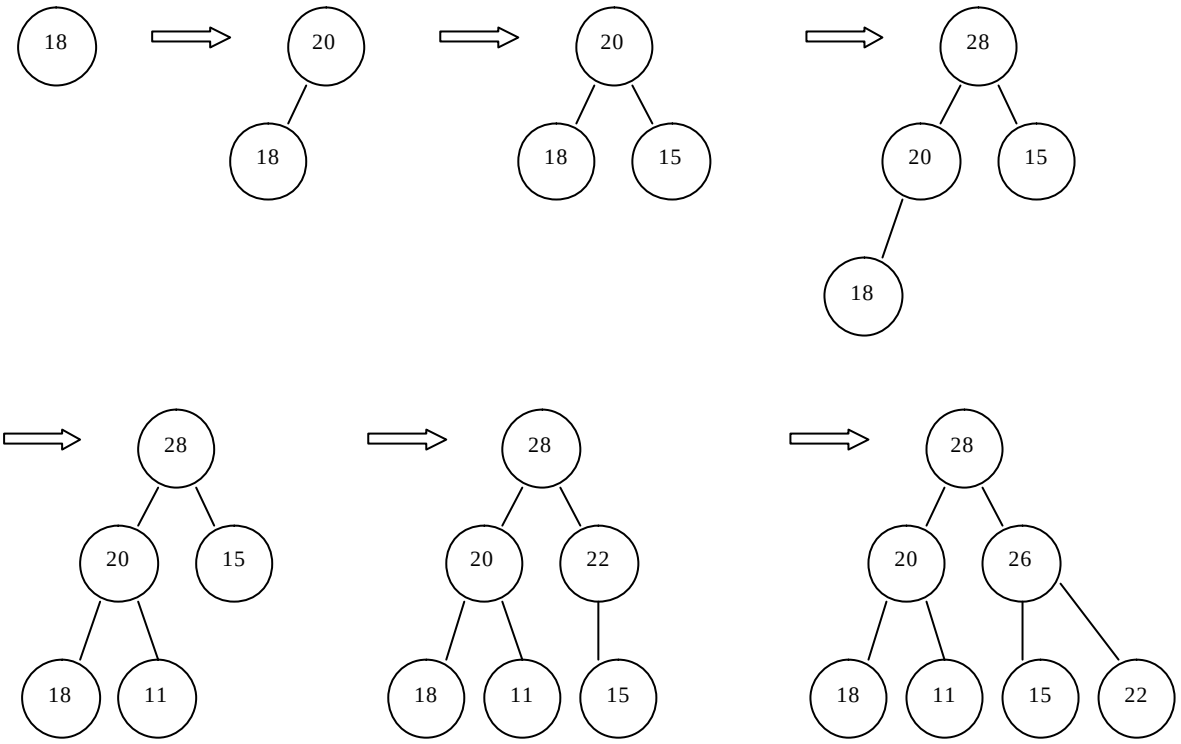
2. Ordenação do heap que gerará o arquivo final ordenado.

A ordenação do heap é feita retirando-se a raiz da árvore, que é o elemento de maior valor do arquivo, colocando-a na posição de maior índice do arquivo e reinserindo o elemento que estava nessa última posição no heap.

Dado o exemplo abaixo, vamos seguir a idéia do método Heapsort.

18	20	15	28	11	22	26
1	2	3	4	5	6	7

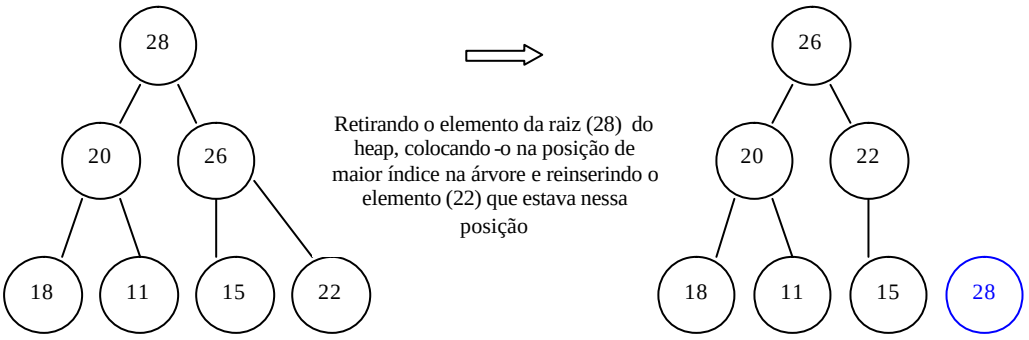
O primeiro passo é a criação do heap (árvore, monte inicial), não esqueça que cada nó tem que ter valor maior que os de seus filhos. Veja a seguir a criação do nosso heap.

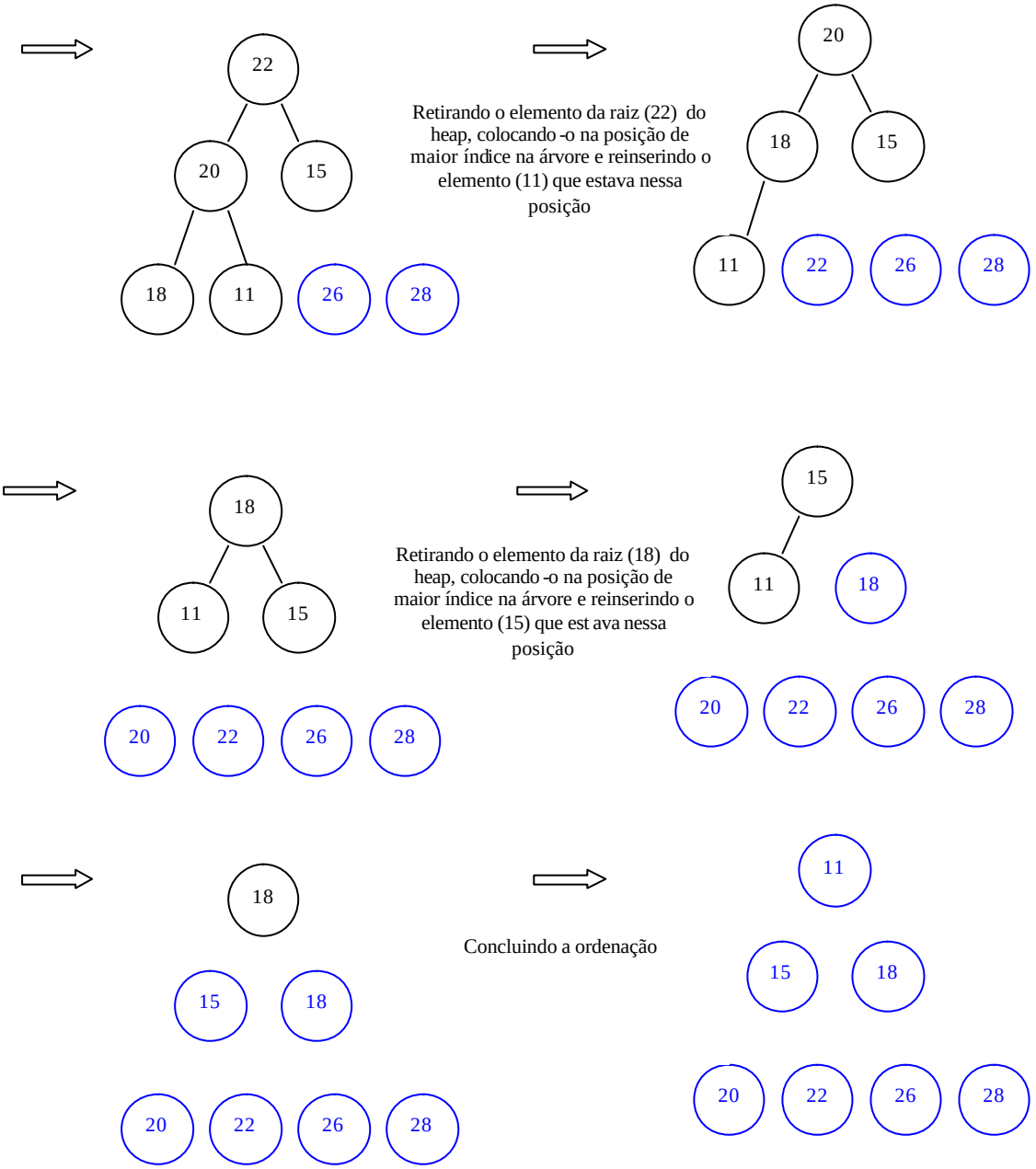


Após a formação do nosso heap, o arquivo estará dessa forma:

28	20	26	18	11	15	22
1	2	3	4	5	6	7

Os passos de ordenação do heap podem ser acompanhados a seguir.





Após a ordenação teremos o arquivo classificado dessa forma:

11	15	18	20	22	26	28
1	2	3	4	5	6	7

Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

4 – Classificação por Distribuição

Na classificação por distribuição encontramos, em geral, um arquivo com números repetidos diversas vezes. Para cada valor repetido no vetor iremos definir em outro vetor o número de repetições do valor, em seguida, recolocamos os valores no arquivo ordenando por uma parte da chave.

A idéia é dividir a chave em partes e para cada uma dessas partes será executada uma etapa de ordenação. O arquivo é ordenado de acordo com a parte da chave correspondente ao passo.

Classificação por Distribuição de Chave

Dado o exemplo abaixo do arquivo que queremos ordenar, vamos seguir a idéia do método de classificação por Distribuição de Chave.

567	361	852	495	689	752	117	659	427	795	342
1	2	3	4	5	6	7	8	9	10	11

O primeiro passo é definir o(s) dígito(s) que será(ão) analisados, no nosso caso iremos analisar o último dígito da chave, ou seja, o dígito das unidades. Em seguida, geraremos a tabela de freqüência do dígito analisado no arquivo a ser ordenado, inicialmente, esta tabela tem todos os seus elementos iguais a zero e depois todo o arquivo original é percorrido pegando a parte do elemento a ser analisada e, a cada ocorrência do valor, a freqüência é acrescida de 1. Veja a seguir a tabela de freqüência gerada.

Dígitos avaliados	0	1	2	3	4	5	6	7	8	9
Freqüência do dígito na parte da chave	0	1	3	0	0	2	0	3	0	2

Isto quer dizer que para o dígito avaliado 2, no nosso caso o último dígito de cada valor de chave, há 3 ocorrências de chaves que possuem o último dígito igual a 2.

Agora, geraremos a tabela de freqüência acumulada, ou seja, a quantidade de ocorrências de chaves com o último dígito sendo igual a qualquer um dos dígitos anteriores. Para entender melhor, verifique, a seguir, a tabela de freqüência acumulada produzida no nosso exemplo.

Dígitos avaliados	0	1	2	3	4	5	6	7	8	9
Freqüência acumulada dos dígitos anteriores (indica o endereço anterior do elemento no arquivo auxiliar)	0	0	1	4	4	4	6	6	9	9

A freqüência acumulada de elementos com último dígito igual a um dos anteriores, quando acrescido de 1, indica, na verdade, a posição no arquivo auxiliar do próximo elemento do arquivo desordenado cujo último dígito seja o avaliado naquele instante.

Para finalizar essa primeira fase, que correspondente a análise do último dígito da chave, utilizaremos um arquivo auxiliar para iniciar a classificação.

Percorrendo o arquivo desordenado inicial, para cada elemento encontrado o colocaremos no arquivo auxiliar na posição seguinte àquela indicada pela tabela de freqüência acumulada analisando-se o dígito avaliado. Ao inserirmos um elemento no arquivo auxiliar, obrigatoriamente, atualizaremos a tabela de freqüência acumulada, somando 1 ao endereço que ela indicava ao elemento de dígito avaliado.

O arquivo auxiliar será criado como segue.

361	852	752	342	495	795	567	117	427	689	659
1	2	3	4	5	6	7	8	9	10	11

A tabela de freqüência acumulada produzida ao final da execução da primeira fase do nosso exemplo está logo abaixo.

Dígitos avaliados	0	1	2	3	4	5	6	7	8	9
Freqüência acumulada dos dígitos anteriores (indica o endereço anterior do elemento no arquivo auxiliar)	0	~~0~~ 1	~~1~~ ~~2~~ ~~3~~ 4	4	~~4~~ ~~5~~ 6	4	~~6~~ ~~7~~ ~~8~~ 9	6	9	~~9~~ ~~10~~ 11

Método RadixSort

O metódo RadixSort (Classificação de Raízes) é outro método baseado na distribuição de chaves. Começamos pegando cada dígito menos significativo (dígito das unidades) e posicionando-o em uma das dez filas, dependendo do valor do dígito. Em seguida, restauramos cada fila para o arquivo original, começando pela fila de números com o dígito 0 e terminando com a fila de números com o dígito 9. Quando essas ações tiverem sido executadas para cada dígito, começando com o menos significativo e terminando com o mais significativo, o arquivo estará classificado.

Dado o exemplo abaixo do arquivo que queremos ordenar, vamos seguir a idéia do método RadixSort.

25	57	48	37	12	92	86	33
1	2	3	4	5	6	7	8

Distribuindo cada número em sua fila correspondente ao dígito da unidade, teremos:

Fila 0	Fila 1	Fila 2	Fila 3	Fila 4	Fila 5	Fila 6	Fila 7	Fila 8	Fila 9
		12 92	33		25	86	57 37	48	

Restaurando as filas para o arquivo original, teremos então:

12	92	33	25	86	57	37	48
1	2	3	4	5	6	7	8

Repetimos a distribuição dos números nas filas, porém agora se baseando no dígito da dezena.

Fila 0	Fila 1	Fila 2	Fila 3	Fila 4	Fila 5	Fila 6	Fila 7	Fila 8	Fila 9
	12	25	33 37	48	57			86	92

Após restaurar novamente as filas, o arquivo estará classificado.

12	25	33	37	48	57	86	92
1	2	3	4	5	6	7	8

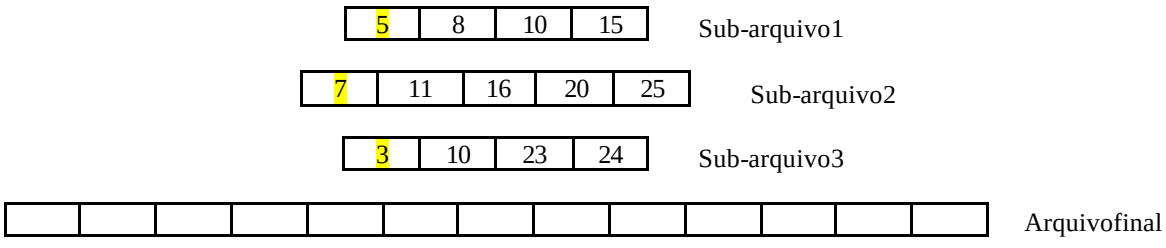
Deixaremos, como exercício, a codificação da rotina que implementa o método RadixSort.

5 – Classificação por Intercalação

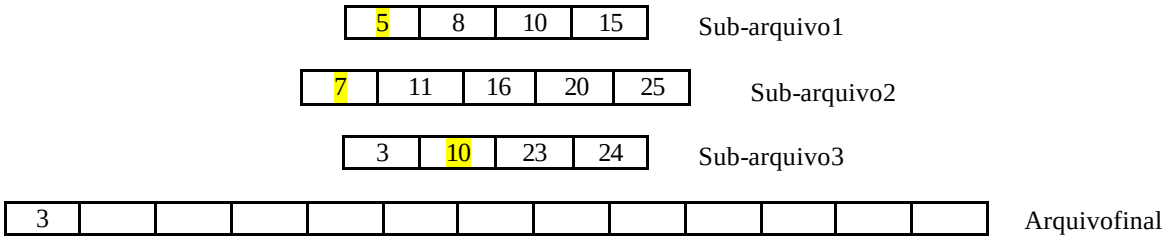
Na classificação por intercalação utilizamos dois ou mais arquivos anteriormente ordenados para gerar um outro arquivo ordenado. É comum a utilização de outros métodos de classificação para ordenação dos sub-arquivos que gerarão o arquivo final ordenado. O processo de intercalação é feito percorrendo-se simultaneamente os sub-arquivos, movendo-se a cada passo o menor entre os elementos da posição analisada dos sub-arquivos para o arquivo resultado final.

É comum dividir o arquivo que queremos ordenar em sub-arquivos ordenados (ou pelo método de Intercalação ou através de qualquer outro) e depois compor o arquivo final através da intercalação simultânea dos sub-arquivos, sendo a cada etapa inserido o menor dos elementos no arquivo que queremos gerar ordenadamente. Vale lembrar que o tamanho do arquivo final é a soma dos tamanhos dos sub-arquivos envolvidos.

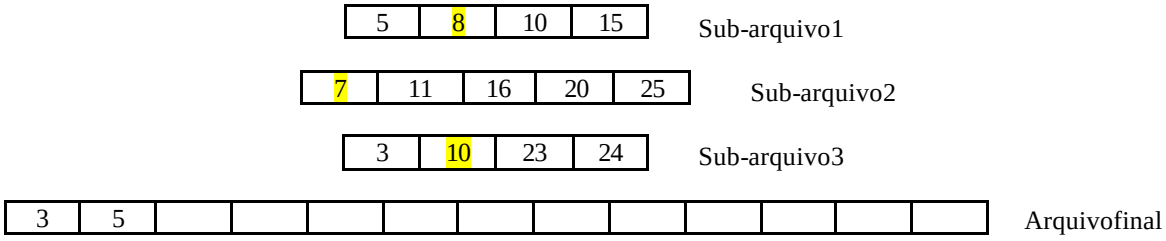
Siga o exemplo abaixo para acompanhar a idéia da classificação por intercalação, utilizando três sub-arquivos para gerar o arquivo final ordenado.

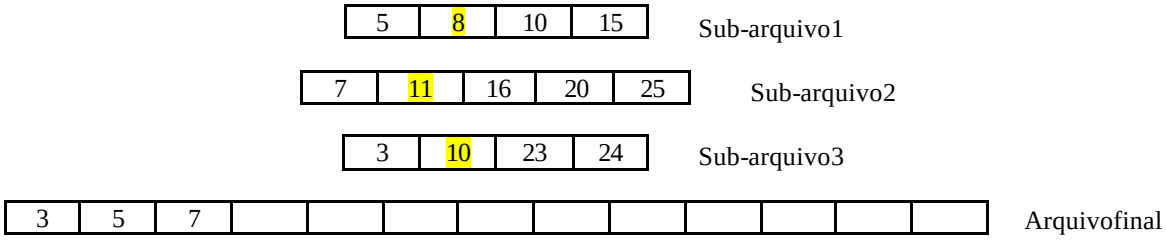


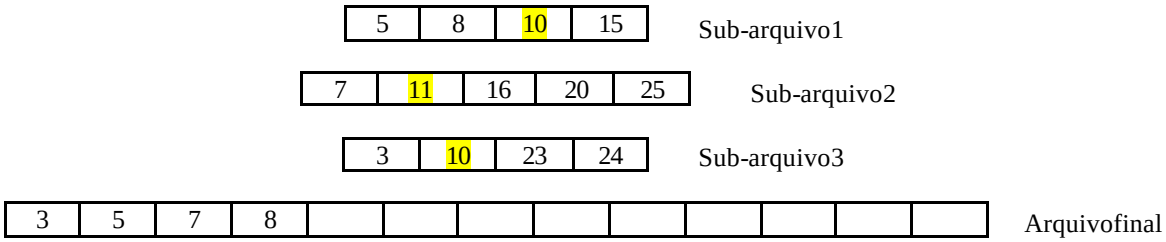
Após o primeiro passo (pega o menor entre os elementos analisados nos sub-arquivos e passa a analisar o próximo apenas no sub-arquivo que possuía o menor entre os elementos) teremos:

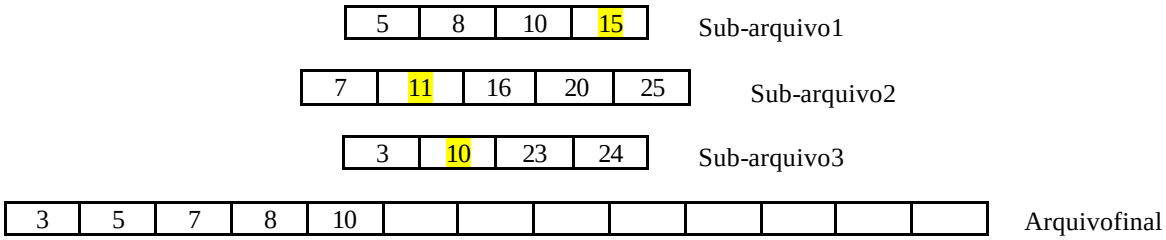


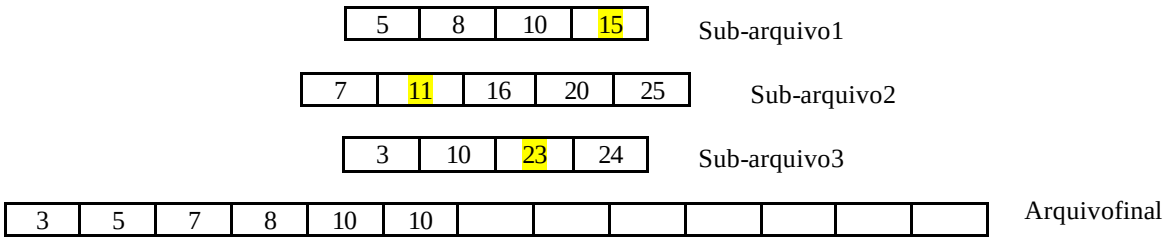
Siga os passos seguintes.

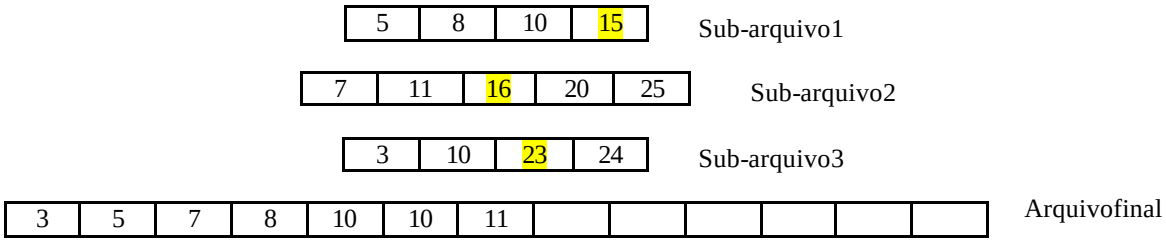


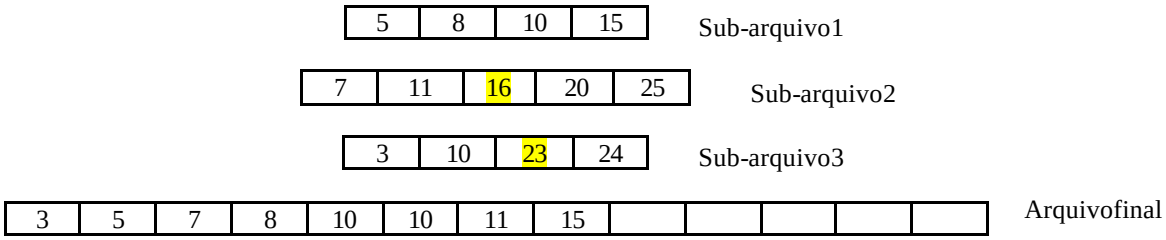


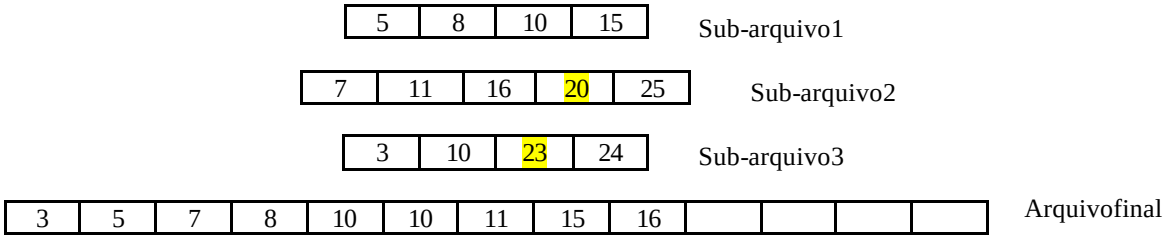


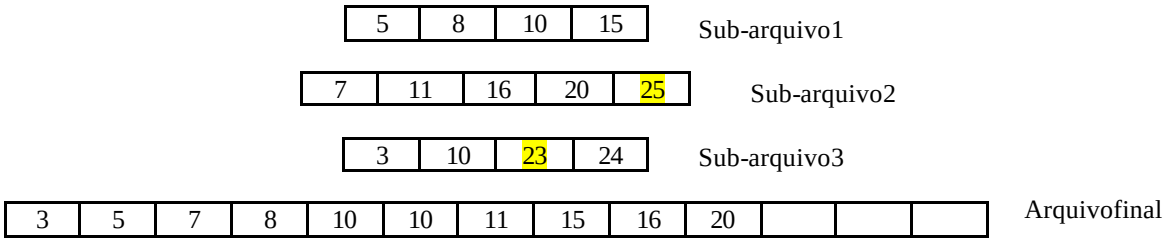


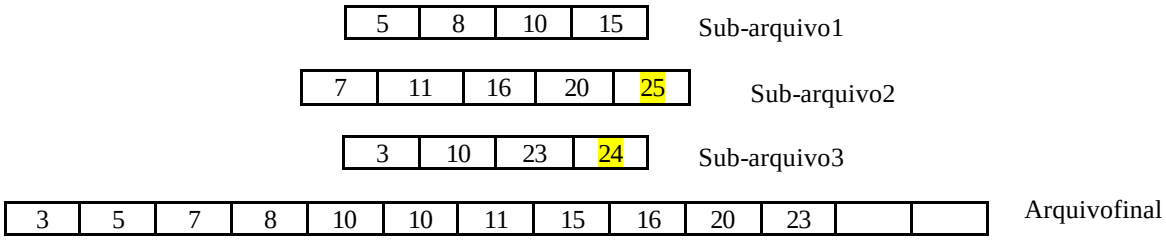


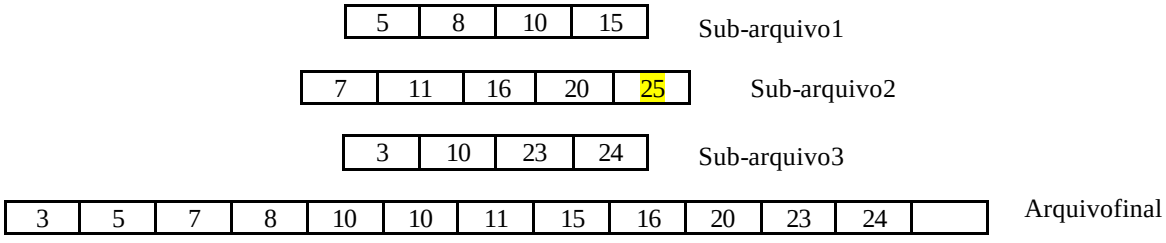

5 8 10 15
Sub-arquivo1
7 11 16 20 25
Sub-arquivo2
3 10 23 24
Sub-arquivo3
3 5 7 8 10 10 11
Arquivofinal

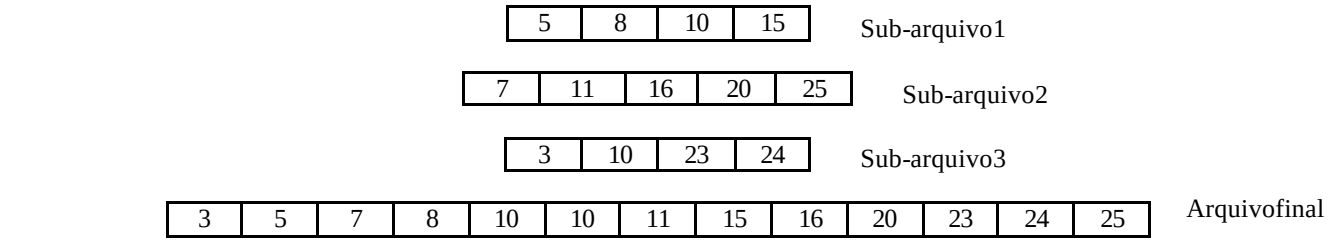

5 8 10 15
Sub-arquivo1
7 11 16 20 25
Sub-arquivo2
3 10 23 24
Sub-arquivo3
3 5 7 8 10 10 11 15
Arquivofinal


5 8 10 15
Sub-arquivo1
7 11 16 20 25
Sub-arquivo2
3 10 23 24
Sub-arquivo3
3 5 7 8 10 10 11 15 16
Arquivofinal


5 8 10 15
Sub-arquivo1
7 11 16 20 25
Sub-arquivo2
3 10 23 24
Sub-arquivo3
3 5 7 8 10 10 11 15 16 20
Arquivofinal


5 8 10 15
Sub-arquivo1
7 11 16 20 25
Sub-arquivo2
3 10 23 24
Sub-arquivo3
3 5 7 8 10 10 11 15 16 20 23
Arquivofinal


5 8 10 15
Sub-arquivo1
7 11 16 20 25
Sub-arquivo2
3 10 23 24
Sub-arquivo3
3 5 7 8 10 10 11 15 16 20 23 24
Arquivofinal

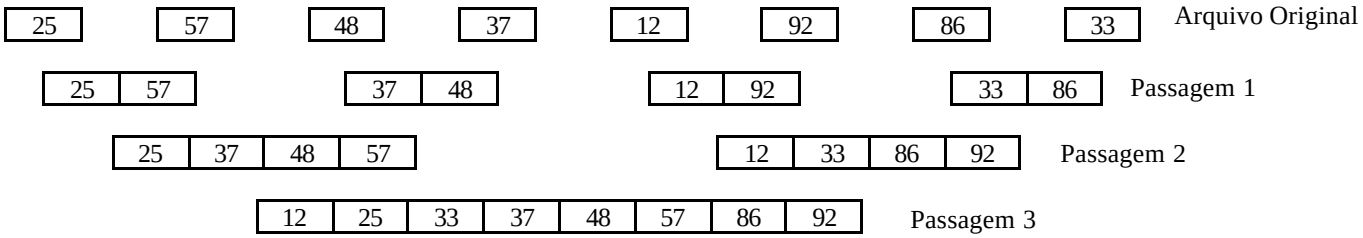


Chegamos agora ao resultado esperado: nosso arquivo totalmente ordenado.

Método MergeSort

O método MergeSort (intercalação direta) funciona da seguinte maneira: divida o arquivo em n sub-arquivos de tamanho 1 e intercale pares de arquivos adjacentes. Temos, então, aproximadamente n/2 arquivos de tamanho 2. Repita esse processo até restar apenas um arquivo de tamanho n.

Veja o exemplo abaixo:



EXERCÍCIOS:

1. Demonstre graficamente a classificação dos dados 38, 21, 45, 30, 35, 27, usando os seguintes métodos:

a) Bolha

b) Inserção Direta

c) Seleção Direta

d) Shell

e) QuickSort

f) HeapSort

g) MergeSort

h) Distribuição de Chaves

i) RadixSort

2. Implemente um programa em C que construa uma tabela comparativa dos métodos de classificação de dados, relacionados a seguir:

- Inserção direta
- Shell
- Bolha
- QuickSort
- Seleção Direta
- HeapSort
- Distribuição de chaves
- MergeSort
- RadixSort

Para cada método, o programa deve calcular:

a) Número médio de comparações

b) Número médio de trocas

c) Média do tempo gasto para a classificação

d) Número de comparações no melhor caso (vetor já classificado)

e) Número de trocas no melhor caso

f) Tempo gasto para o melhor caso

g) Número de comparações no pior caso (vetor classificado inversamente)

h) Número de trocas no pior caso

i) Tempo gasto para o pior caso

Observações:

- Utilize um vetor de 100 elementos inteiros, gerado com valores aleatórios no intervalo de 0 a 999.
- Para o item (a), (b) e (c), obtenha a média da classificação de 20 vetores distintos.