

Estrutura de Dados II



Unidade 6 – Técnicas de Ordenação

Prof. Sandro T. Pinto



*Como preza a estratégia
algorítmica: “Primeiro coloque
os números em ordem. Depois
decidimos o que fazer.”*

www.treinaweb.com.br/blog/












Unidade 6

Técnicas de Ordenação - Introdução

Imagine que a agenda de seu celular não apresenta função de **busca por contatos** e, para piorar, seus dados estão completamente desordenados. Não seria muito mais fácil organizar seus contatos em ordem alfabética para que nós, humanos, possamos utilizar a agenda de maneira mais adequada?

Classificar

? X

 Adicionar Nível			 Excluir Nível			 Copiar Nível			 		 Opções...		<input checked="" type="checkbox"/> Meus dados contêm cabeçalhos	
Coluna				Classificar em				Ordem						
Classificar por 				Valores 				De A a Z 						

Unidade 6

Técnicas de Ordenação - Introdução

Imagine, agora, que você tem a lista de várias compras e vendas realizadas diariamente em uma organização. Sua lista encontra-se **ordenada de acordo com as datas** nas quais cada compra ou venda ocorreu. Seu líder lhe solicita que organize a lista de forma a considerar o nome do comprador/vendedor, em ordem alfabética. Como faríamos para ordenar sua lista?

EMPRESA TESTE LTDA

www.futurasistemas.com.br

06/04/2018 14:38:20

Página 1 de 1

ESTOQUE FINANCEIRO

Classificado por Fornecedor

37

CAROLINE MESQUITA

Cód	Referência	Descrição	Cód. de Barra	Estoque	Custo	Tl. Custo	Venda	Tl. Venda
206	206-12	ASASAS - M - AMARELO - ALGODÃO	2100002018855	0,00	0,00	0,00	0,00	0,00
108	55	CADERNO TESTE 1	7891321046821	187,00	2,00	374,00	5,00	935,00
21303	21303	CADERNO TESTE JC	21303	9,00	5,00	45,00	3,00	27,00
21603	21603-4E	CAMISETA POLO MASCULINA - G - AZUL - ALC 21603		0,00	0,00	0,00	0,00	0,00
21703	21703-4E	CAMISETA POLO MASCULINA - G - BRANCO - 21703		0,00	0,00	0,00	0,00	0,00
21403	21403-4E	CAMISETA POLO MASCULINA - M - AZUL - ALC 21403		0,00	0,00	0,00	0,00	0,00
21503	21503-4E	CAMISETA POLO MASCULINA - M - BRANCO - 21503		0,00	0,00	0,00	0,00	0,00
20703	00007	CANETA ESFEROGRAFICA	20703	0,00	0,50	0,00	1,00	0,00
110	110	CRISOIDINA PO - CERTIFICADO DE ANALISE (2100002017711		500,00	0,00	0,00	8,59	4.297,39
186	186-10	DSOSSFFFD - M - AZUL - ALGODÃO	2100002018657	0,00	0,00	0,00	0,00	0,00
187	187-10	DSOSSFFFD - M - AZUL - LYCRA	2100002018664	0,00	0,00	0,00	0,00	0,00
182	182-10	DSOSSFFFD - M - BRANCO - ALGODÃO	2100002018619	0,00	0,00	0,00	0,00	0,00
183	183-10	DSOSSFFFD - M - BRANCO - LYCRA	2100002018626	0,00	0,00	0,00	0,00	0,00

Unidade 6

Técnicas de Ordenação - Introdução

Por isso, veremos agora alguns algoritmos de ordenação de implementação mais simplificada. Todos os métodos apresentados aqui são capazes de ordenar um conjunto de dados armazenados em um vetor, de maneira exata. Veremos os algoritmos:

- Bubblesort;
- Selectionsort;
- Insertionsort;
- Shellsort.



Unidade 6

Técnicas de Ordenação - Introdução

Para testar as técnicas faz-se necessário que seja criado um programa em que esses algoritmos sejam executados.


[Link do programa](#)

Unidade 6

Técnicas de Ordenação - Introdução

O Programa em questão, mostra a declaração das bibliotecas, constantes e variáveis. As bibliotecas **stdio** e **stdlib** são velhas conhecidas de qualquer estudante ou programador habituado na linguagem C. A diferença está na biblioteca **time**, ela foi incluída para que possamos fazer uso da máquina de geração de números aleatórios que veremos adiante.

```
//Bibliotecas  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>
```




Unidade 6

Técnicas de Ordenação - Introdução

Existe uma única constante chamada **tamanho**. Ela servirá de parâmetro em praticamente todas as funções de ordenação. Ela define o **tamanho máximo do vetor** em que os nossos dados estarão armazenados. Para ficar mais fácil, fixamos o tamanho em 10, mas podemos e devemos alterar esse número para um valor maior durante as suas investigações.

```
//Constantes  
#define tamanho 10
```



Unidade 6

Técnicas de Ordenação - Introdução

Criamos dois vetores, a variável **lista** guardará os dados na ordem original, e a variável **ordenado** conterá os valores após a aplicação das técnicas de ordenação.

Com o vetor original intacto, podemos aplicar diferentes técnicas uma após a outra sem precisar “bagunçar” os dados no vetor antes de testar um novo algoritmo.

```
//Variáveis  
int lista[tamanho]; ←  
int ordenado[tamanho]; ←  
int opt=-1;  
int qtd;
```

Unidade 6

Técnicas de Ordenação - Introdução

A variável **opt** é usada para fazer o controle do menu e a variável **qtd** é usada para calcular “o esforço computacional” despendido pela técnica de ordenação.

```
//Variáveis  
int lista[tamanho];  
int ordenado[tamanho];  
int opt=-1; ←  
int qtd; ←
```

Unidade 6

Técnicas de Ordenação - Introdução

Esta parte é representado como a região de **prototipação**. Essa área é usada para colocar a lista de funções existentes no programa. Isso permite que as funções possam ser chamadas de qualquer parte do arquivo.

Quando for adicionar um algoritmo de ordenação desse programa, **adicione nesse bloco o nome da função** que irá executar a técnica escolhida.

```
//Prototipação ←  
void menu_mostrar(void);  
void lista_mostrar(void);  
void lista_gerar(void);  
void lista_ler(void);  
void lista_limpar(void);  
void lista_mostrar_ordenado(void);
```

Unidade 6

Técnicas de Ordenação - Introdução

Agora vejamos a função principal desse Programa.

O comando **srand** está incluído na biblioteca `time` e serve para inicializar a máquina geradora de números aleatórios, que usaremos mais adiante.

```
//Função Principal
int main(void) {
    srand(time(NULL)); ←
    do {
        system("cls");
        lista_mostrar();
        lista_mostrar_ordenado();
        menu_mostrar();
        scanf("%d", &opt);
        switch (opt) {
            case 1:
                lista_gerar();
                break;
            case 2:
                lista_ler();
                break;
        }
    } while (opt != 0);
    system("pause");
    return(0);
}
```

Unidade 6

Técnicas de Ordenação - Introdução

O laço principal **limpa a tela(cls)**,
desenha o vetor
original(**lista_mostrar**), o vetor
ordenado(**lista_mostra_ordenad**
o) e mostra ao usuário quais são
as opções disponíveis no
programa(**menu_mostrar**).

```
//Função Principal
int main(void){
    srand(time(NULL));
    do {
        system("cls");
        lista_mostrar();
        lista_mostrar_ordenado();
        menu_mostrar();
        scanf("%d",&opt);
        switch (opt){
            case 1:
                lista_gerar();
                break;
            case 2:
                lista_ler();
                break;
        }
    }while(opt!=0);
    system("pause");
    return(0);
}
```

Unidade 6

Técnicas de Ordenação - Introdução

Por meio da entrada na variável **opt**, a estrutura **case** chama a função desejada.

Quando estiver incluindo uma nova função programa, adicione **um novo caso na estrutura case** para que o usuário possa estar aplicando a técnica de ordenação desejada.

```
// Função Principal
int main(void) {
    srand(time(NULL));
    do {
        system("cls");
        lista_mostrar();
        lista_mostrar_ordenado();
        menu_mostrar();
        scanf("%d", &opt);
        switch (opt) {
            case 1:
                lista_gerar();
                break;
            case 2:
                lista_ler();
                break;
        }
    } while (opt != 0);
    system("pause");
    return(0);
}
```

Unidade 6

Técnicas de Ordenação - Introdução

O Programa mostra três funções criadas para desenhar a tela do programa. Será necessário alterar apenas a `menu_mostrar`. Originalmente ela contém três opções:

- 1) gerar lista aleatoriamente;
- 2) criar uma lista manualmente;
- 3) sair.

Para cada algoritmo de ordenação incluído no programa, uma nova opção deverá ser criada na função `menu_mostrar`.

Unidade 6

Técnicas de Ordenação - Introdução

```
//Mostra o conteúdo da lista
void lista_mostrar(void) {
    printf("[ ");
    for (int i = 0; i < tamanho; i++) {
        printf("%d ", lista[i]);
    }
    printf("]\n\n");
}

//Mostra o menu
void menu_mostrar(void) {
    printf("1 - Gerar lista aleatoriamente\n");
    printf("2 - Criar lista manualmente\n");
    printf("0 - Sair...\n\n");
}

//Mostra o conteúdo da lista ordenada
void lista_mostrar_ordenado(void) {
    printf("[ ");
    for (int i = 0; i < tamanho; i++) {
        printf("%d ", ordenado[i]);
    }
    printf("] Tempo = %d iteracoes\n\n", qtd);
}
```

Unidade 6

Técnicas de Ordenação - Introdução

No caso de o usuário não querer digitar o conteúdo de um vetor para que possa ser ordenado, foi criada uma função **lista_gerar**. Ela percorre todo o vetor lista e para cada posição sorteia um número entre 0 e 50 por meio da função **rand() % 50**.

```
//Gera uma lista aleatória
void lista_gerar(void) {
    for (int i = 0; i < tamanho; i++) {
        lista[i] = rand() % 50; ←
    }
}
```

Unidade 6

Técnicas de Ordenação - Introdução

Por fim, mais duas funções. A primeira, solicita que o usuário preencha uma **lista com valores escolhidos por ele**. Em alguns casos poderá ser interessante testar os diversos algoritmos em vetores ordenados ou parcialmente ordenados, verificando o desempenho nessas situações peculiares.

```
//Permite que o usuário entre com os valores da lista
void lista_ler(void){
    for (int i = 0; i < tamanho; i++){
        system("cls");
        lista_mostrar();
        printf("\nDigite o valor para a posicao %d: ", i);
        scanf("%d", &lista[i]);
    }
}
```

Unidade 6

Técnicas de Ordenação - Introdução

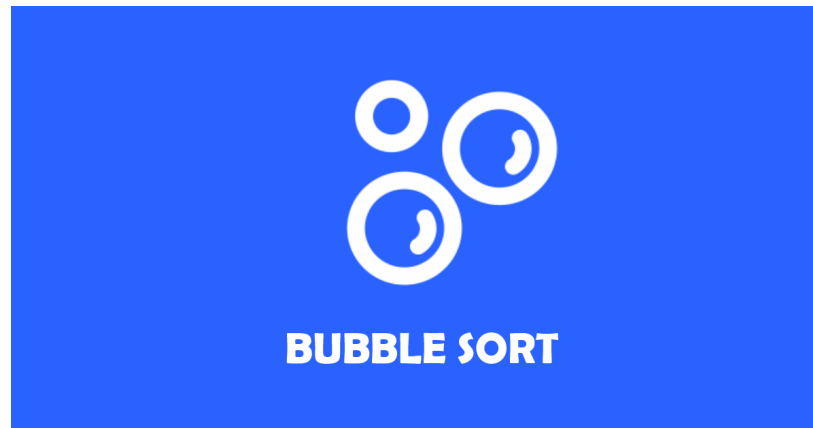
A função **lista_limpar** prepara o vetor auxiliar para a aplicação da técnica de ordenação escolhida pelo usuário. Essa função deve ser incluída no laço do menu principal junto com a função de ordenação em cada uma das novas entradas na estrutura case.

```
//Preparar a lista para ordenação
void lista_limpar(void) {
    for (int i = 0; i < tamanho; i++) {
        ordenado[i] = lista[i];
    }
}
```

Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

A técnica de **ordenação Bubblesort** também é conhecida por ordenação por flutuação ou por **método da bolha**. Ela é de simples implementação e de alto custo computacional.

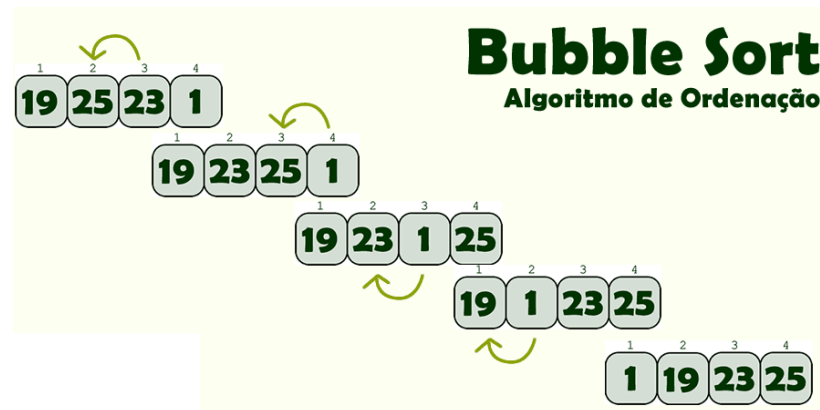


[Dança do bubble sort](#)

Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

Começando na primeira posição do vetor, compara-se o valor dela com todos os demais elementos, trocando caso o valor da posição atual seja maior do que o valor verificado. Os valores mais altos vão flutuando para o final do vetor, criando a ordenação da estrutura. Esse processo se repete para cada uma das posições da tabela.




Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

O Programa traz também uma função chamada **troca**. Ela recebe como parâmetro dois ponteiros e tem como objetivo trocar os seus valores de lugar. Essa função também será utilizada em outros algoritmos de ordenação

```
//Aplica o método do bubbleSort
int bubbleSort(int vec[]) {
    int qtd, i, j, tmp;
    qtd = 0;
    for (i = 0; i < tamanho - 1; i++) {
        for (j = i + 1; j < tamanho; j++) {
            if (vec[i] > vec[j]) {
                troca(&vec[i], &vec[j]);
            }
            qtd++;
        }
    }
    return(qtd);
}

//Função genérica de troca de valores
void troca(int* a, int* b) { 
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

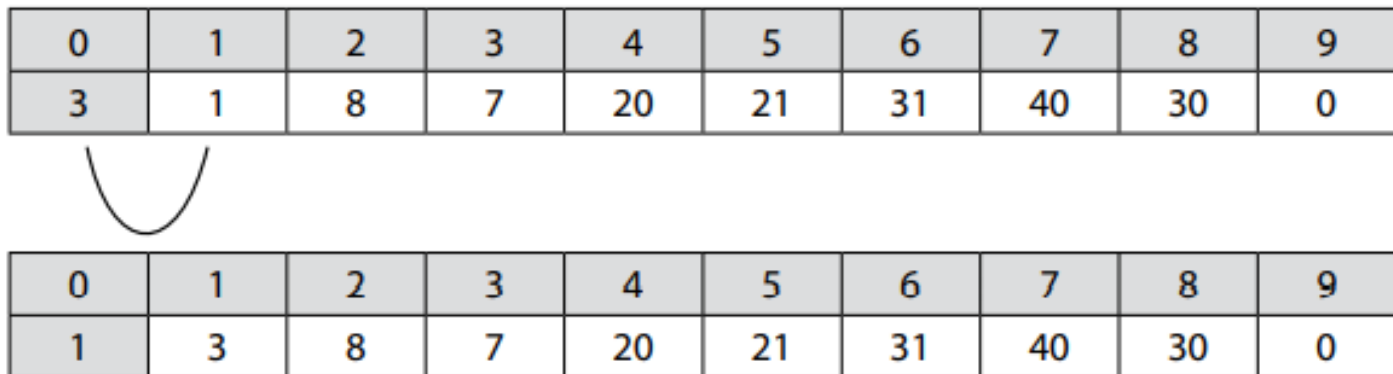
Exemplo de simulação do algoritmo com base no vetor `vec[]` desordenado de testes apresentado aqui:

0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0

Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

Vamos fixar na primeira posição $\text{vec}[0]=3$ e compará-lo com o próximo $\text{vec}[1]=1$. Como $3 > 1$, os valores são trocados no vetor.



The diagram illustrates the first step of the bubble sort algorithm. It shows two states of a 10-element array. In the initial state, the first element is 3 and the second is 1. A curved arrow indicates the swap of these two elements. In the resulting state, the first element is 1 and the second is 3. The rest of the array remains unchanged.

0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0


0	1	2	3	4	5	6	7	8	9
1	3	8	7	20	21	31	40	30	0

Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

Continuamos fixo no **vec[0]**, que agora possui valor 1, e comparamos com a próxima posição **vec[2]=8**. Como $1 < 8$, nada acontece.

O programa continua comparando **vec[0]** com todas as demais posições do vetor de dados até encontrar **vec[9]=0**, onde haverá nova troca.




0	1	2	3	4	5	6	7	8	9
1	3	8	7	20	21	31	40	30	0

0	1	2	3	4	5	6	7	8	9
0	3	8	7	20	21	31	40	30	1

Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

Depois fixamos em **vec[1]** e novamente comparamos com todos os demais. Como **vec[1]=3**, acontecerá troca apenas em **vec[9]=1**.



0	1	2	3	4	5	6	7	8	9
0	3	8	7	20	21	31	40	30	1

0	1	2	3	4	5	6	7	8	9
0	1	8	7	20	21	31	40	30	3

Unidade 6

Técnicas de Ordenação - BUBBLESORT (MÉTODO DA BOLHA)

Depois fixamos **vec[2]** e, fazemos comparação com o restante do vetor e assim por diante, até que encontremos a última posição e o arquivos estejam ordenados. Ao mesmo tempo em que os valores maiores são empurrados para a direita, os menores são puxados para a esquerda.

No final de cada iteração do laço externo do algoritmo, a parte inicial da tabela fica mais e mais ordenada..

Unidade 6

Técnicas de Ordenação - SELECTIONSORT

A técnica também é de simples implementação e de alto consumo computacional.

A partir da primeira posição, procura-se o menor valor em todo o vetor. Chegando no final da estrutura, trocamos o menor valor encontrado com a primeira posição. Em seguida, ele parte para a segunda posição e passa a procurar o segundo menor valor do vetor até o final da tabela, fazendo a troca de posição dos valores.

[Dança do Selection sort](#)

Unidade 6

Técnicas de Ordenação - SELECTIONSORT

O algoritmo repete até que a lógica seja aplicada a cada uma das posições da tabela.

```
//Aplica o modo selectionSort
int selectionSort(int vec[], int tam){
    int i, j, min, qtd=0;
    for (i = 0; i < (tam-1); i++)
    {
        min = i;
        for (j = (i+1); j < tam; j++) {
            if(vec[j] < vec[min]) {
                min = j;
            }
            qtd++;
        }
        if (i != min) {
            troca(&vec[i], &vec[min]);
        }
    }
    return(qtd);
}
```

Unidade 6

Técnicas de Ordenação - SELECTIONSORT


Exemplo de simulação do algoritmo com base no vetor `vec[]` desordenado de testes apresentado aqui:

0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0

Unidade 6

Técnicas de Ordenação - SELECTIONSORT

A partir da primeira posição, o algoritmo vai percorrer o vetor até o seu final armazenando numa variável temporária o menor valor encontrado, que no nosso caso é $\text{vec}[9]=0$. Mais uma vez usaremos a **função troca**. Ela irá trocar os valores da primeira posição $\text{vec}[0]=3$ com $\text{vec}[9]=0$.




0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0

0	1	2	3	4	5	6	7	8	9
0	1	8	7	20	21	31	40	30	3

Unidade 6

Técnicas de Ordenação - SELECTIONSORT

Agora, a partir da segunda posição, o algoritmo percorrerá todo o vetor buscando o segundo menor valor, que é ele mesmo $\text{vec}[1]=1$. Nada acontece. O próximo passo é procurar a partir da terceira posição o terceiro menor valor, que é $\text{vec}[9]=3$. O valor é trocado com o de $\text{vec}[2]=8$



0	1	2	3	4	5	6	7	8	9
0	1	8	7	20	21	31	40	30	3

0	1	2	3	4	5	6	7	8	9
0	1	3	7	20	21	31	40	30	8

Unidade 6

Técnicas de Ordenação - SELECTIONSORT

O algoritmo continua até que o processo seja repetido para cada uma das posições da tabela.

Enquanto o **Bubblesort** faz uma troca sempre que a posição atual fixa é maior que a posição visitada, o **Selectionsort** faz a troca apenas quando tem certeza que o menor valor foi encontrado para a atual posição

Unidade 6

Técnicas de Ordenação - INSERTIONSORT

A ordenação **Insertionsort** também é conhecida como ordenação por **inserção**. É de implementação simples e traz bons resultados.

A técnica consiste em remover o primeiro elemento da lista, e procurar sua posição ideal no vetor e reinseri-lo na tabela. O processo é repetido para todos os elementos

[Dança do Insertion sort](#)

Unidade 6

Técnicas de Ordenação - INSERTIONSORT

O Programa apresenta dois laços de repetição, o primeiro é executado inteiro e o segundo um número aleatório, dependendo da distância que o elemento está da sua posição ideal. Esse algoritmo também utiliza a função **troca**.

```
//Aplicando o insertionSort
int insertionSort(int vec[], int tam)
{
    int i, j, qtd=0;
    for(i = 1; i < tam; i++){
        j = i;
        while((vec[j] < vec[j - 1]) && (j!=0)){
            troca(&vec[j], &vec[j-1]);
            j--;
            qtd++;
        }
    }
    return(qtd);
}
```

Unidade 6

Técnicas de Ordenação - INSERTIONSORT

A princípio o **Insertionsort** é muito parecido com o **Bubblesort** e o **Selectionsort**, já que todos os três trazem dois laços de repetição aninhados.

Porém os **Bubblesort** e o **Selectionsort** percorrem sempre os dois laços por inteiro. Esse é o motivo do **Insertionsort** ser mais rápido do que os outros dois, veja o exemplo:

Unidade 6

Técnicas de Ordenação - INSERTIONSORT


Exemplo de simulação do algoritmo com base no vetor `vec[]` desordenado de testes apresentado aqui:

0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0

Unidade 6

Técnicas de Ordenação - INSERTIONSORT

O primeiro elemento **vec[0]=3**, vamos removê-lo do vetor e inseri-lo novamente do lado do primeiro número que encontrarmos que for menor do que ele.




0	1	2	3	4	5	6	7	8	9
3	1	8	7	20	21	31	40	30	0

0	1	2	3	4	5	6	7	8	9
1	3	8	7	20	21	31	40	30	0

Unidade 6

Técnicas de Ordenação - INSERTIONSORT

Os valores de $\text{vec}[0]=1$ e $\text{vec}[1]=3$ se encontram corretamente posicionados no vetor. O próximo a ser analisado será $\text{vec}[2]=8$. Ele será removido e então reinserido assim que for encontrado um valor menor do que ele, que no caso será $\text{vec}[3]=7$.



The diagram illustrates the insertion sort process. An arrow points from the element 8 at index 2 to the position before index 3, indicating its movement to be inserted before the element 7.

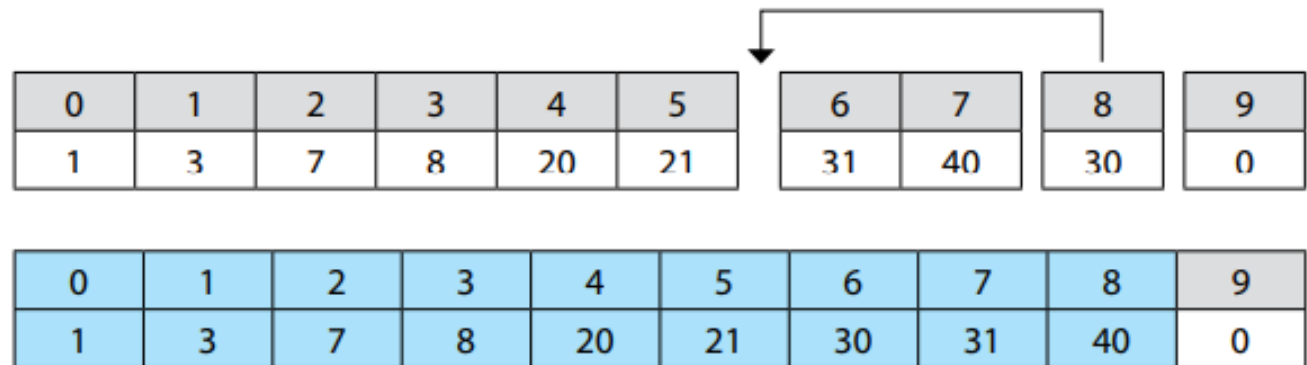
0	1	2	3	4	5	6	7	8	9
1	3	8	7	20	21	31	40	30	0

0	1	2	3	4	5	6	7	8	9
1	3	7	8	20	21	31	40	30	0

Unidade 6

Técnicas de Ordenação - INSERTIONSORT

O valor de **vec[4]** já se encontra ao lado de um número maior que ele, assim como **vec[5]** e **vec[6]**, não havendo nenhuma alteração. O próximo valor que será removido e reinserido no vetor é **vec[8]=30**.



Agora falta fazer o mesmo com o último elemento **vec[9]** e o vetor estará ordenado.

Unidade 6

Técnicas de Ordenação - SHELLSORT

O algoritmo **Shellshort** de ordenação não tem nada a ver com uma concha (shell, em inglês).

Ele tem esse nome em homenagem ao seu criador Donald Shell, publicado pela Universidade de Cincinnati em 1959.

Segundo Wirth (1989, p. 61 a 63), ele é o mais eficiente dentro dos algoritmos classificados como de complexidade quadrática. Ele é uma **técnica refinada do método de ordenação por inserção**.

Unidade 6

Técnicas de Ordenação - SHELLSORT

Ao invés de tratar o arquivo como um todo, ele divide a tabela em segmentos menores e em cada um deles **é aplicado o Insertionsort**.

Ele faz isso diversas vezes, dividindo grupos maiores em menores até que todo o vetor esteja ordenado.

Dança do Shell sort

Unidade 6

Técnicas de Ordenação - SHELLSORT

O Programa apresenta a implementação do algoritmo de **Shellsort**. Ele possui uma variável chamada **gap**. O **gap** determina a distância entre os elementos que serão removidos do vetor original. Ao sub vetor aplica-se o algoritmo de **Insertionsort**, e o sub vetor é novamente inserido no vetor original. O processo se repete até atingir todos os elementos.

```
//Aplica o shellSort
int shellSort(int vec[], int tam) {
    int i , j , valor, qtd=0;
    int gap = 1;
    do {
        gap = 3*gap+1;
    } while(gap < tam);
    do {
        gap /= 3;
        for(i = gap; i < tam; i++) {
            valor = vec[i];
            j = i - gap;
            while (j >= 0 && valor < vec[j]) {
                vec[j + gap] = vec[j];
                j -= gap;
            }
            qtd++;
            vec[j + gap] = valor;
        }
    } while ( gap > 1);
    return (qtd);
}
```

Unidade 6

Técnicas de Ordenação - SHELLSORT

O valor de **gap** sofre um decremento e uma nova quantidade de grupos é criada no vetor parcialmente ordenado. Aplica-se o processo de ordenação por **Insertionsort** em cada um dos sub vetores. O processo se repete até que **gap** seja igual a 1, então uma nova sequência de **insertionsort** é realizada e o vetor termina por estar ordenado.

```
//Aplica o shellSort
int shellSort(int vec[], int tam) {
    int i , j , valor, qtd=0;
    int gap = 1;
    do {
        gap = 3*gap+1;
    } while(gap < tam);
    do {
        gap /= 3;
        for(i = gap; i < tam; i++) {
            valor = vec[i];
            j = i - gap;
            while (j >= 0 && valor < vec[j]) {
                vec[j + gap] = vec[j];
                j -= gap;
            }
            qtd++;
            vec[j + gap] = valor;
        }
    } while ( gap > 1);
    return (qtd);
}
```

Unidade 6

Técnicas de Ordenação - SHELLSORT

Imagine um vetor **vec** de 12 posições e a variável **gap** com valor 3. O primeiro subgrupo terá os valores:

vec[0], vec[3], vec[6], vec[9], vec[12].

Nesse sub vetor de quatro elementos é aplicado o **Insertionsort** e seus dados serão inseridos de volta no vetor original. Depois é a vez dos valores:

vec[1], vec[4], vec[7], vec[10].

Aplica-se o **Insertionsort** e devolve os valores ao vetor. O processo continua agora para os valores:

vec[2], vec[5], vec[8], vec[11]

Unidade 6

Técnicas de Ordenação - SHELLSORT

O novo sub vetor é ordenado por **Insertionsort**, seu resultado é inserido de volta ao vetor original. Nesse momento, finaliza a primeira passagem e gap sofre um decremento.

Vamos supor para o nosso caso que o decremento é de 1, então gap agora será 2. Com o novo valor de gap, um novo subvetor é criado com os seguintes valores:

vec[0], vec[2], vec[4], vec[6], vec[8], vec[10], vec[12]

Unidade 6

Técnicas de Ordenação - SHELLSORT

Esse sub vetor é ordenado por **Insertionsort**. O processo continua até que todos os valores tenham sido escolhidos e o gap sofre um novo incremento. Quando o valor de gap for 1, será aplicado o algoritmo de **Insertionsort no vetor original** parcialmente ordenado e o processo é finalizado.

Unidade 6



Referências:

SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estruturas de dados e seus algoritmos. 2ed. Rio de Janeiro: LTC, 1994. 320p.

TENENBAUM, Aaron M.; LANGSAM, Yedidiah; AUGENSTEIN, Moshé J.. Estruturas de dados usando C. São Paulo: Makron Books, 1995. 884p.

VELOSO, Paulo et al.. Estruturas de dados. Rio de Janeiro: Campus, 2001. 228p

Atividades - Unidade 6

