UNIVERSIDADE FEDERAL DE GOIÁS – UFG
INSTITUTO DE INFORMÁTICA
SEMESTRE SELETIVO 2018/1

CURSO:	Engenharia da Computação
DISCIPLINA:	ESTRUTURA DE DADOS II
PROFESSORA:	Ma. Renata Dutra Braga (renata@inf.ufg.br ou professorarenatabraga@gmail.com)
TEMA DA AULA:	Algoritmos de ordenação
AULAS	09 e 11/abril/2018

ALGORITMOS DE ORDENAÇÃO

Algoritmos: considerações gerais

- Servem para ordenar/organizar uma lista de números ou palavras de acordo com a sua necessidade.
- Os mais populares algoritmos de ordenação são: Insertion sort, Selection sort, Bubble sort, Quick sort, Merge sort, Heap sort e Shell sort, estando assim estruturados:
    - Ordenação por inserção
        - *Insertion Sort (direta)*
        - **Shell sort**
    - Ordenação por troca
        - *Bubble sort*
        - **Quick sort**
    - Ordenação por seleção
        - *Selection sort*
        - Heap sort
    - Ordenação por intercalação
        - **Merge sort**
    - Ordenação por distribuição

Existem vários aplicativos na Play Store e Aplle Store que demonstram, de forma didática, o funcionamento de cada algoritmo (ex: Sort Simulation, Sorting algorithms e Algorithm View).

Algoritmos de ordenação: considerações gerais

- Algoritmo de ordenação é um algoritmo que coloca os elementos de uma dada sequência em uma certa ordem.
- Isto é, rearranja os itens de um vetor ou lista de modo que suas chaves estejam ordenadas de acordo com
- alguma regra (podendo a ordenação ser completa ou parcial).
- O objetivo da ordenação é facilitar a recuperação dos dados de uma lista.
- Exemplo de aplicação: A realizar a consulta por produtos, ordenados pela data de cadastro, em um banco de dados.

Alguns conceitos importantes:

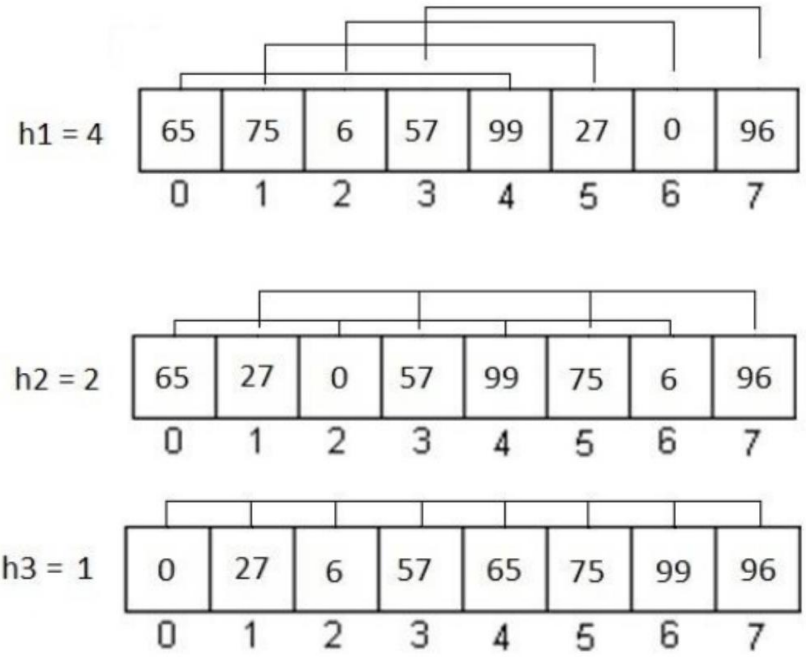
- **Ordenação interna**: todas os dados estão em memória principal (RAM). Portanto, todos os elementos a serem ordenados cabem na memória principal e qualquer registro pode ser imediatamente acessado.
- **Ordenação externa**: memória principal não cabe todos os dados. Portanto, devem estar armazenados em memória secundária (disco) e os registros são acessados sequencialmente ou em grandes blocos.
- **Estabilidade**: um algoritmo é estável se a ordem relativa dos registros com a mesma chave não se altera após a ordenação (mantém a ordem de inserção).
- **Adaptabilidade**: um algoritmo é adaptável quando a sequência de operações realizadas depende da entrada. Inversamente, quando um algoritmo que sempre realiza as mesmas operações, independente da entrada, é não adaptável.
- **In-place**: é quando necessita de uma estrutura auxiliar para realizar a ordenação (comparações e movimentos).
- A quantidade de comparação e a troca realizadas por um algoritmo são os fatores que determinam a eficiência (ou ineficiência) do mesmo.

Ordenação por inserção

Shell sort

- É o mais eficiente algoritmo de classificação dentre os de complexidade quadrática.
- Extensão (ou refinamento) do algoritmo de ordenação por inserção direta (Insertion sort).
- O algoritmo de ordenação de Shell (nome de seu inventor Donald Shell) foi um dos primeiros a baixar a complexidade de $O(N^2)$.
- Ordenação por inserção só troca itens adjacentes para determinar o ponto de inserção. O método de Shell contorna este problema, permitindo trocas de registros distantes.
    - Se o menor item estiver na posição mais a direita no vetor então o número de comparações e movimentações é igual a n-1 para encontrar o seu ponto de inserção.
- Portanto, O Shell sort difere do método de inserção direta pelo fato de no lugar de considerar o array a ser ordenado como um único segmento, ele considera vários segmentos sendo aplicado o método de inserção direta em cada um deles.
    - O algoritmo passa várias vezes pela lista dividindo o grupo maior em menores.
    - Nos grupos menores é aplicado o método da ordenação por inserção.
- Funcionamento:
    - O algoritmo usa uma sequência de incrementos e ordena os elementos cuja distância é igual a este incremento.
        - Cada incremento não deve ser múltiplo do anterior.
    - A cada etapa o incremento vai diminuindo até chegar a 1, ao final de uma etapa com incremento h, dizemos que o conjunto está h-ordenado.
    - Uma importante propriedade deste algoritmo é que para h1 < h2 um conjunto h1-ordenado, que é submetido a uma ordenação com incremento h2, permanece h1-ordenado. A figura ilustra a ideia do algoritmo.

n = 8
h = 4



    - Método proposto por D. L. Shell que propõe a ordenação dos elementos distantes **h** posições. Este parâmetro h é uma sequência que sempre termina em 1. Ele não apresenta regra fixa para ser definida, sendo que em várias situações o resultado esperado pode não ocorrer.
    - A ideia do método é rearranjar a lista de tal forma que fiquem ordenados os elementos que ocupam posições que diferem entre si por um passo h constante. Na verdade, a lista passa a consistir em sub listas ordenadas com elementos espaçados de h posições. Essa tal lista se diz h-ordenação.

```
Função ShellSort(A, n)
    aux, i, j, h = n/2;
    Enquanto h > 0
        i = h;
        Enquanto i < n
            aux = A[i]
            j = i;
        Enquanto j >= h && aux < A[j - h]
                A[j] = A[j - h];
                j = j - h;
            A[j] = aux;
            i = i +1;
    h = h/2;
```

Características do Shell sort

- Bom para ordenar um número moderado de elementos
- Quando encontra um arquivo parcialmente ordenado, trabalha menos

Shell sort	
classe	Algoritmo de ordenação
estrutura de dados	Array, Listas ligadas
complexidade pior caso	depende da sequência do gap. Melhor conhecida: $O(n \log_2 n)$
complexidade caso médio	depende da sequência do gap
complexidade melhor caso	$O(n)$
complexidade de espaços pior caso	$O(n)$

Vantagens do Shell sort

- Shellsort é uma ótima opção para arquivos de tamanho moderado
- Sua implementação é simples e requer uma quantidade de código pequena

Desvantagens do Shell sort

- O tempo de execução do algoritmo é sensível à ordem inicial do arquivo
- O método não é estável, pois ele nem sempre deixa registros com chaves iguais na mesma posição relativa.

Vídeos sobre o Shell sort:

- https://www.youtube.com/watch?v=c1tRyovhb3I

Vamos praticar?

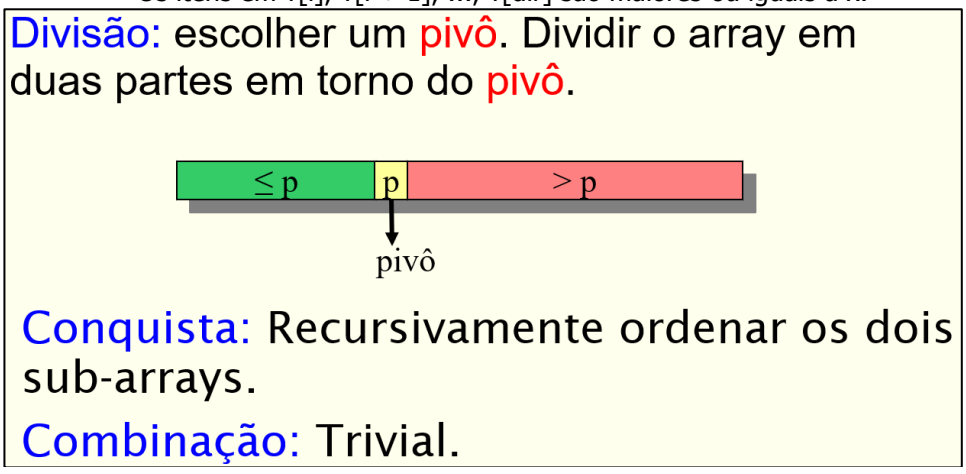
QUESTÃO 01

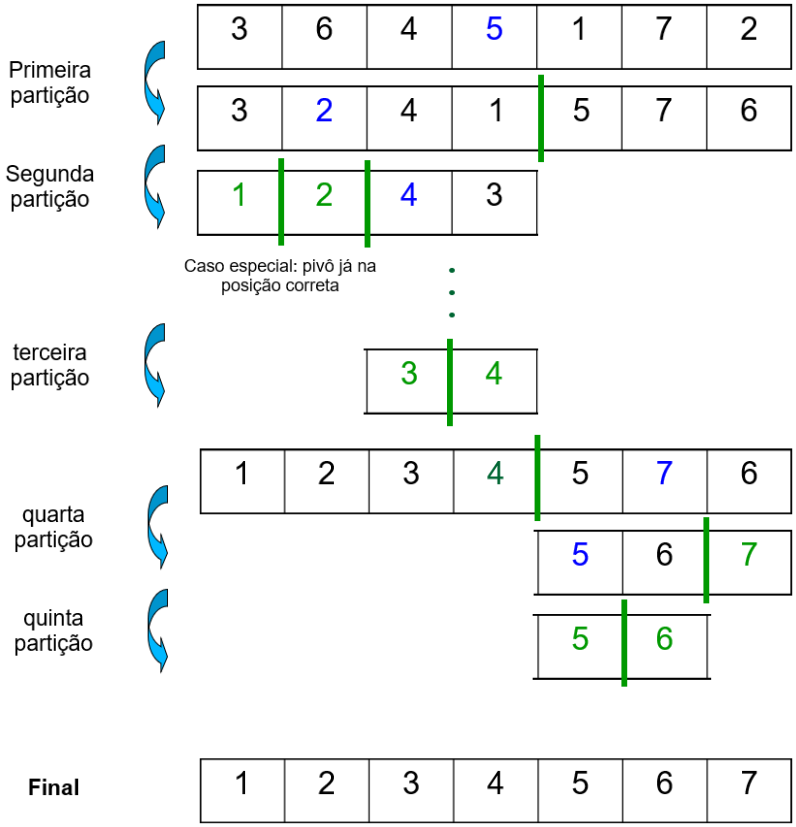
Implemente na linguagem C o algoritmo de ordenação Shell sort. Utilize uma função auxiliar para implementar a ordenação.

Ordenação por troca

Quick sort

- É o algoritmo mais eficiente que existe para uma grande variedade de situações. Provavelmente é o mais utilizado.
- O algoritmo é recursivo, o que demanda uma pequena quantidade de memória adicional
- Pior caso realiza $O(n^2)$ operações.
- Basicamente a operação do algoritmo pode ser resumida na seguinte estratégia:
    - divide 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.
- O principal cuidado a ser tomado é com relação à escolha do pivô
    - A escolha do elemento do meio do arranjo melhora o desempenho quando o arquivo está total ou parcialmente ordenado
    - O pior caso tem uma probabilidade muito pequena de ocorrer quando os elementos forem aleatórios
    - Eficiência: mediana de três: *Geralmente se usa a mediana de uma amostra de três elementos para evitar o pior caso*
- Usar inserção em partições pequenas melhora o desempenho significativamente.
    - Funcionamento:
    - A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois sub-problemas menores.
    - Os problemas menores são ordenados independentemente.
    - Os resultados são combinados para produzir a solução final.
    - A parte mais delicada do método é o processo de partição.
    - O vetor v[esq..dir] é rearranjado por meio da escolha arbitrária de um pivô x.
    - O vetor v é particionado em duas partes:
        - Parte esquerda: chaves ≤ x.
        - Parte direita: chaves ≥ x.
    - Ideia: Dividir e Conquistar
    - Algoritmo para o particionamento:
        - 1. Escolha arbitrariamente um pivô x.
        - 2. Percorra o vetor a partir da esquerda até que v[i] ≥ x.
        - 3. Percorra o vetor a partir da direita até que v[j] ≤ x.
        - 4. Troque v[i] com v[j].
        - 5. Continue este processo até os apontadores i e j se cruzarem.
    - Ao final, do algoritmo de partição:
        - Vetor v[esq..dir] está particionado de tal forma que:
            - Os itens em v[esq], v[esq + 1], ..., v[j] são menores ou iguais a x;
            - Os itens em v[i], v[i + 1], ..., v[dir] são maiores ou iguais a x.





Características do Quick sort

- É um algoritmo de comparação que emprega a estratégia de "divisão e conquista".
- A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores.
- Os problemas menores são ordenados independentemente e os resultados são combinados para produzir a solução final.

Quicksort	
classe	Algoritmo de ordenação
estrutura de dados	Array, Listas ligadas
complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n \log n)$
complexidade melhor caso	$O(n \log n)$
complexidade de espaços pior caso	$O(n)$
otimo	Não
estabilidade	não-estável

Vantagens do Quick sort

- É extremamente eficiente para ordenar arquivos de dados.
- Necessita de apenas uma pequena pilha como memória auxiliar.
- Requer O(n log n) comparações em média (caso médio) para ordenar n itens.

Desvantagens do Quick sort

- Tem um pior caso $O(n^2)$ comparações.
- Sua implementação é delicada e difícil:
- Um pequeno engano pode levar a efeitos inesperados para algumas entradas de dados.
- O método não é estável.

Vídeos sobre o Quick sort:

- https://www.youtube.com/watch?v=GqVRiqNl1UA
- https://www.youtube.com/watch?v=Ge5MVrOkN8U

Vamos praticar?

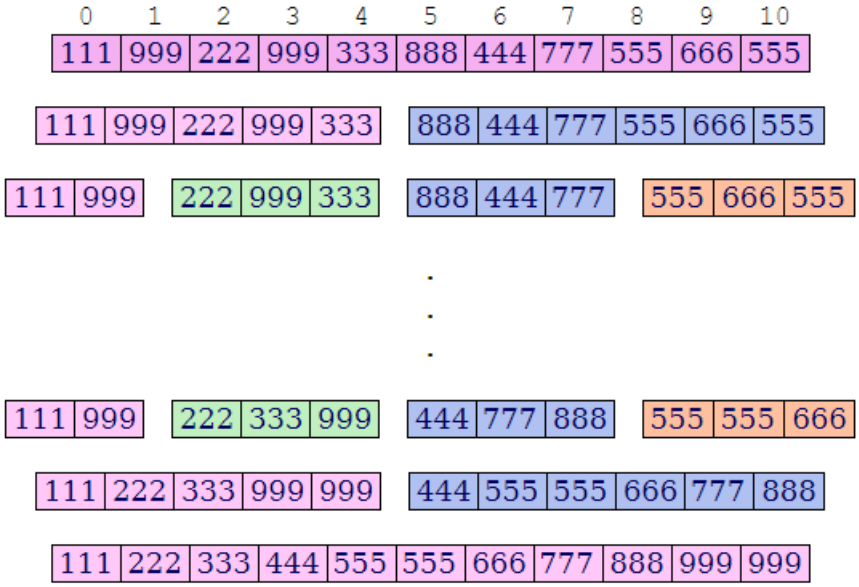
QUESTÃO 02

Implemente na linguagem C o algoritmo de ordenação Quick sort. Utilize uma função auxiliar para implementar a ordenação.

Ordenação por intercalação

Merge sort

- Criado por C.A.R. Hoare (em 1960), é um método de ordenação muito rápido e eficiente.
- Princípio:
  - Parte o arranjo em dois.
  - Intercala dois arranjos independentes.
- Ou seja,
- Método que consiste em Dividir a entrada em conjuntos menores
  - Resolve cada instância menor de maneira recursiva
  - Reuni as soluções parciais para compor a solução do problema original.



Características do Merge sort

- Dividir: Calcula o ponto médio do vetor e divide.
- Conquistar: Recursivamente, resolve dois sub-problemas, cada um de tamanho n/2.
- Combinar: Unir os sub-vetores em um único conjunto ordenado.

Merge sort	
classe	Algoritmo de ordenação
estrutura de dados	Array, Listas ligadas
complexidade pior caso	$\Theta(n \log n)$
complexidade caso médio	$\Theta(n \log n)$
complexidade melhor caso	$\Theta(n \log n)$ típico, $\Theta(n)$ variante natural
complexidade de espaços pior caso	$\Theta(n \log n)$

Vantagens do Merge sort

- Pior caso: O (n log n)
- Só precisa acesso sequencial aos dados.
- Boa opção quando dados estão em lista encadeada.
- Fácil implementação.
- Indicado para aplicações que exigem restrição de tempo (executa sempre em um determinado tempo para um dado n)

Desvantagens do Merge sort

- Por utilizar funções recursivas, o algoritmo necessita de memória extra (auxiliar).
- O algoritmo cria uma cópia do vetor para cada nível da chamada recursiva, totalizando um uso adicional de memória igual a (n log n).

Vídeos sobre o Merge sort:

- https://www.youtube.com/watch?v=-UsrwCcTrzY
- https://www.youtube.com/watch?v=PKCMMSXQyJE

Vamos praticar?

QUESTÃO 03

Implemente na linguagem C o algoritmo de ordenação Merge sort. Utilize uma função auxiliar para implementar a ordenação.

REFERÊNCIAS SUPLEMENTAR

Juliano Schimiguel. **Algoritmos de ordenação: análise e comparação**. Disponível em: https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261

W. Celes e J.L. Rangel. **Ordenação**. Disponível em: http://www.ic.unicamp.br/~ra069320/PED/MC102/1s2008/Apostilas/Cap15.pdf

EXERCÍCIOS

QUESTÃO 04

O trecho de algoritmo a seguir corresponde ao método de ordenação do tipo:

```
1   declare X[5], n, i, aux numérico
2   para n ← 1 até 5 faça
3       início
4           para i ← 0 até 3 faça
5               início
6                   se (X [i] > X[i+1] )
7                       então início
8                           aux X[i]
9                           X [i] ← X[i + 1]
10                          X[i+1] ← aux
11              fim
12      fim
13  fim
```

a) Quick sort.
b) Merge sort.
c) Bubble sort.
d) Insertion sort.
e) Selection sort.

QUESTÃO 05

O seguinte algoritmo, chamado ordena, implementa um conhecido método de ordenação para listas seqüenciais:

```
ordena (int vet[], int n) {
        int i, j, pos, aux;
        para ( i = 0; i < n - 1; i++ ){
                pos = i;
                para ( j = i + 1; j < n; j++ )
                        se ( vet [pos] > vet [j] )
                                pos = j;
                se ( pos <> i ) {
                        aux = vet[i];
                        vet[i] = vet[pos];
                        vet[pos] = aux;
                }
        }
}
```

Se o algoritmo for executado recebendo como parâmetros {5, 3, 1, 2, 4} e 5, quantas trocas são efetuadas e em que sentido é feita a ordenação (crescente ou decrescente)?
a) 5, crescente.
b) 6, crescente.
c) 9, crescente.
d) 4, decrescente.
e) 7, decrescente.

QUESTÃO 06

Considere o seguinte algoritmo, responsável por realizar a ordenação de um array de dados.

```
public int[] mySortingAlgorithm (int[] data){
 int size = data.length;
 int tmp = 0;
 for(int i = 0;i<size;i++){
  for(int j = (size-1);j>=(i+1);j--){
   if(data[j]<data[j-1]){
    tmp = data[j];
    data[j]=data[j-1];
    data[j-1]=tmp;
   }
  }
 }
 return data;
}
```

Podemos afirmar que o método de ordenação utilizado pelo algoritmo é o:
a) quickSort;
b) insertionSort;
c) mergeSort;
d) shellSort;
e) bubbleSort.

QUESTÃO 07
Observe o código abaixo, que busca o maior elemento de um vetor v[0..n -1].

```
int max(int n, int v[])
{
  int j, x = v[0];
  for (j = 1; j < n; j += 1)
    if (x < v[j]) x = v[j];
  return x;
}
```

A complexidade de tempo desse algoritmo é:
a) O(logn)
b) O(n)
c) (nlogn)
d) O(1)
e) (n2)

QUESTÃO 08
Qual é o tipo de algoritmo de ordenação que tem como princípio percorrer o vetor diversas vezes, a cada passagem fazendo o maior elemento se mover para o final da estrutura?
a) Double sort
b) Heap sort
c) Merge sort
d) Bubble sort
e) Insertion sort

QUESTÃO 09
Correlacione os algoritmos internos de ordenação de listas da coluna à esquerda com sua descrição, na coluna à direita.
1) Bubblesort.
2) Ordenação por Seleção
3) Ordenação por Inserção
4) Shellsort
5) Quicksort

( ) Escolhe-se um pivot e particiona-se a lista em duas sublistas: uma com os elementos menores que ele e outra com os maiores, que, ao serem ordenadas e combinadas com o pivot, geram uma lista ordenada. O processo é aplicado às partições para ordená-las. Embora tenha uma complexidade de pior caso de O(n2), no caso médio é de O(n log n).

( ) Encontra-se o menor item do vetor. Troca-se com o item da primeira posição do vetor. Repetem-se essas duas operações com os n − 1 itens restantes, depois com os n − 2 itens, até que reste apenas um elemento.

( ) Método preferido dos jogadores de cartas. A cada momento existem duas partes na lista: uma ordenada (destino) e outra não ordenada (fonte). Inicialmente a lista destino tem apenas o primeiro elemento, e a fonte os demais elementos. Em cada passo a partir de i=2, seleciona-se o i-ésimo item da lista fonte. Deve-se colocá-lo no lugar apropriado na lista destino, de acordo com o critério de ordenação.

( ) É uma extensão de um outro algoritmo de ordenação conhecido e permite trocas de elementos distantes um do outro, não necessariamente adjacentes. Os itens separados de h posições são rearranjados. Todo h-ésimo item leva a uma lista ordenada. Tal lista é dita estar h-ordenada.

( ) Varre-se a lista trocando-se de posição os elementos adjacentes fora de ordem. Varre-se a lista até que não haja mais trocas e, neste caso, a lista está ordenada.

A sequência correta, de cima para baixo, é:

a) 1, 2, 3, 4, 5.
b) 5, 2, 3, 4, 1.
c) 1, 4, 5, 3, 2.
d) 5, 4, 2, 3, 1.
e) 1, 3, 2, 4, 5.

QUESTÃO 10
Quantas comparações e trocas de posição ocorrerão se utilizarmos o algoritmo Bubble Sort para ordenar do menor para o maior valor o vetor [60,32,45,5,6,2], respectivamente:
a) 10 e 4.
b) 15 e 13.
c) 25 e 15.
d) 22 e 12.
e) 12 e 9.

QUESTÃO 11
O algoritmo de ordenação denominado quicksort é baseado na partição do arquivo em duas partes, a partir de um elemento arbitrariamente escolhido que termina localizado na sua posição final. Cada uma das partes é então ordenada independentemente, aplicando-se o algoritmo recursivamente, até que todo o arquivo esteja ordenado.

Analise as mudanças na disposição dos elementos de um vetor com 10 elementos que é submetido ao processo de partição.

1	2	3	4	5	6	7	8	9	10
A	S	O	R	T	I	N	B	C	E
A	C	O	R	T	I	N	B	S	E
A	C	B	R	T	I	N	O	S	E
A	C	B	E	T	I	N	O	S	R

O elemento arbitrariamente escolhido foi aquele que estava na posição:
a) 1
b) 5
c) 8
d) 9
e) 10

QUESTÃO 12

Utilizando o algoritmo Shell sort, obtenha o número de comparações e movimentações em cada passo (h e i) para os seguintes vetores

- 45,56,12,43,95,19,8,67
- 8,12,19,43,45,56,67,95
- 95,67,56,45,43,19,12,8
- 19,12,8,45,43,56,67,95

QUESTÃO 13

Escreva uma função que receba vetores disjuntos x[0..m-1] e y[0..n-1], ambos em ordem crescente, e produza um vetor z[0..m+n-1] que contenha o resultado da intercalação dos dois vetores dados. Escreva duas versões da função: uma iterativa e uma recursiva.

Any fool can write code that a computer can understand.
Good programmers write code
that humans can understand.
— Martin Fowler,
Refactoring: Improving the Design of Existing Code