Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Programa de Pós-Graduação em Ciência da Computação
Projeto e Análise de Algoritmos

Exercícios de Ordenação

1. De acordo com o procedimento TESTE, apresentado em anexo, responda os itens abaixo.

a) O que acontece ao chamarmos TESTE(A, n, i) quando $i > \frac{n}{2}$?

b) Tendo como entrada o vetor $A = [10\ 13\ 12\ 17]$, ilustre e/ou explique o funcionamento do algoritmo HeapSort para ordená-lo de forma crescente.

2. Dado o vetor de caracteres $A = [S\ O\ R\ T]$, com quatro elementos, e o algoritmo de ordenação QUICK-SORT, apresentado em anexo, responda os itens abaixo.

a) Ilustre e/ou explique o funcionamento do algoritmo QUICK-SORT para ordenar de forma crescente o vetor A.

b) Comente sobre o tempo de execução esperado para o procedimento de ordenação realizado no item (a). Justifique sua resposta.

c) Caso a escolha do elemento pivô fosse feita de forma aleatória, o tempo de execução esperado para o procedimento de ordenação realizado no item (a) poderia ser reduzido? Por quê?

3. [POSCOMP 2006] Quais algoritmos de ordenação têm complexidade no tempo $O(n\ log\ n)$ para o melhor caso, onde n é o número de elementos a ordenar?

(A) InsertionSort e QuickSort.
(B) QuickSort e HeapSort.
(C) BubbleSort e InsertionSort.
(D) HeapSort e InsertionSort.
(E) QuickSort e BubbleSort.

4. Um grupo de pesquisa resolveu avaliar a eficiência no tempo de três algoritmos de ordenação: MergeSort, QuickSort e HeapSort. O resultado (em segundos) dos experimentos para ordenar 10^6 números inteiros de forma crescente, considerando quatro situações iniciais de ordem, encontra-se na tabela abaixo. Na ordem identificada como "repetida", todos os elementos do vetor são iguais.

Algoritmo	**Aleatória**	**Crescente**	**Decrescente**	**Repetida**
A	0,90	0,80	0,80	0,85
B	0,80	0,80	0,70	0,20
C	0,60	8,95	8,00	9,00

Agora, baseado nos dados da tabela acima, trabalhe os itens a seguir.

a) Identifique os algoritmos A, B e C. Justifique sua resposta.

b) Na sua opinião, qual foi a versão do algoritmo QuickSort, com relação a escolha do pivô, usada nos experimentos? Por quê?

c) Você concorda com a afirmativa de que o tempo de execução esperado para o algoritmo HeapSort não é influenciado pela ordenação inicial do vetor de entrada? Justifique.

5. Dado o vetor $A = [4\ 0\ 2\ 0\ 1]$, com cinco números inteiros, responda os itens abaixo.

a) Ilustre e/ou explique o funcionamento do algoritmo CountingSort para ordenar de forma crescente o vetor A.

b) Comente sobre o tempo de execução esperado para o procedimento de ordenação realizado no item (a). Justifique sua resposta.

c) A ordenação realizada no item (a) foi estável? Explique.

6. [POSCOMP 2005] Um algoritmo de ordenação é estável se a ordem relativa dos itens com chaves iguais mantém-se inalterada após a ordenação. Quais dos seguintes algoritmos de ordenação são estáveis?

I. BubbleSort (ordenação por bolha).
II. InsertionSort (ordenação por inserção).
III. HeapSort.
IV. QuickSort.

(A) Somente II.
(B) Somente I e II.
(C) Somente I, II e III.
(D) Somente II, III e IV.
(E) Somente I, III e IV.

7. [POSCOMP 2006] Seja P o problema de ordenar, usando comparação, $n \geq 1$ elementos e C a classe dos algoritmos que resolvem P. O limitante inferior de C é

(A) $\Omega(1)$.
(B) $\Omega(log\ n)$.
(C) $\Omega(n)$.
(D) $\Omega(n\ log\ n)$.
(E) $\Omega(n^2)$.

8. [POSCOMP 2010] Considere o problema de ordenação onde os vetores a serem ordenados, de tamanho $n > 0$, possuem $\lfloor n/2 \rfloor$ valores iguais a um número inteiro x e $\lceil n/2 \rceil$ valores iguais a um outro número inteiro y. Considere que os números x e y são conhecidos e fixos, porém, estão distribuídos aleatoriamente no vetor a ser ordenado. Neste caso, é correto afirmar que

(A) podemos ordenar estes vetores a um custo $O(n)$.
(B) no caso médio, o Quicksort será o algoritmo mais eficiente para este problema, com um custo de $O(n\ log\ n)$.
(C) o algoritmo de ordenação por inserção sempre opera no melhor caso com um custo $O(n)$.
(D) O limite inferior para esta classe de problema é $\Omega(n^2)$.
(E) O limite inferior para esta classe de problema é $\Omega(n\ log\ n)$.

9. Algoritmos de ordenação podem ser ou não *in-place*. Um algoritmo de ordenação é *in-place* se a memória adicional requerida é independente do tamanho do vetor que está sendo ordenado. Dito isso, assinale a alternativa que possui apenas algoritmos de ordenação que NÃO são *in-place*.

(A) MergeSort e BubbleSort.
(B) BubbleSort e CountingSort.
(C) CountingSort e MergeSort.
(D) HeapSort e CountingSort.
(E) HeapSort e MergeSort.

10. [POSCOMP 2008] Assinale a afirmativa INCORRETA.

(A) Seja $A[1, n]$ um vetor não ordenado de inteiros com um número constante k de valores distintos. Então existe algoritmo de ordenação por contagem que ordena A em tempo linear.

(B) Seja $A[1, n]$ um vetor não ordenado de inteiros com um número constante k de valores distintos, então o limite inferior para um algoritmo de ordenação por comparações para ordenar A é de $O(nlgn)$.

(C) Seja $A[1, n]$ um vetor não ordenado de inteiros, cada inteiro com no máximo d dígitos, onde cada dígito assume um valor entre um número constante k de valores distintos. Então o problema de ordenar A tem limite inferior $O(n)$.

(D) Seja $A[1, n]$ um vetor não ordenado de inteiros, cada inteiro com no máximo d dígitos, onde cada dígito assume um valor entre $O(n)$ valores distintos. Então o problema de ordenar A tem limite inferior $O(nlgn)$.

(E) Seja $A[1, n]$ um vetor não ordenado de inteiros com um número constante k de valores distintos, então um algoritmo de ordenação por comparações ótimo para ordenar A tem complexidade $O(nlgn)$.

11. O vetor $A = [8\ 7\ 6\ 5\ 4\ 3\ 2\ 1]$ foi submetido a um algoritmo de ordenação. Em algum ponto da ordenação, o vetor se encontra da seguinte forma $A = [6\ 5\ 3\ 1\ 4\ 2\ 7\ 8]$. Dentre os listados abaixo, qual foi o algoritmo de ordenação utilizado para ordenar o vetor A?

(A) Seleção.
(B) Inserção.
(C) BubbleSort.
(D) HeapSort.
(E) Nenhum deles.

12. Sobre o algoritmo HeapSort, é correto afirmar que

(A) sua ideia básica se baseia na ordenação por árvore de decisão, ao invés de ordenação por comparação.

(B) a estrutura de dados que utiliza, chamada *heap*, pode ser interpretada como uma árvore binária de pesquisa.

(C) seu desempenho de pior caso é pior do que o do algoritmo QuickSort.

(D) conhecido também como método da seleção em árvore, seu desempenho de pior caso é o mesmo da ordenação por intercalação.

(E) seu desempenho de pior caso é melhor do que o da ordenação por contagem quando o elemento de maior valor no vetor de entrada é um inteiro $O(n)$, onde n é o número de elementos do vetor de entrada.

13. [POSCOMP 2003] Em um *heap* com n vértices existem

(A) exatamente $\lfloor n/5 \rfloor$ folhas.

(B) aproximadamente $log(n)$ folhas.

(C) não mais que $\lfloor n/5 \rfloor$ folhas.

(D) exatamente $\lceil n/2 \rceil$ folhas.

(E) não menos que $2n/3$ folhas.

14. [POSCOMP 2012] Seja V um vetor de n inteiros não negativos, tal que o maior valor encontrado em V é $m > 0$. Com relação à ordenação de V, analise as afirmativas a seguir.

I. O tempo de execução dos algoritmos Quicksort e Mergesort para ordenar V é $\Omega(nlgn)$ para qualquer valor de m.

II. Quando $m = O(n)$, é possível ordenar V em tempo de execução $O(n)$ no pior caso.

III. O tempo de execução de pior caso do Quicksort para ordenar V é $O(nlgn)$ quando $m = O(n)$.

IV. Para instâncias onde $n = O(m)$, o algoritmo Countingsort é mais eficiente que o Mergesort, em função de n.

A análise permite concluir que somente as afirmativas

(A) I e II são corretas.

(B) I e IV são corretas.

(C) III e IV são corretas.

(D) I, II e III são corretas.

(E) II, III e IV são corretas.

15. Seja $V =< v_1, ..., v_n >$ uma lista qualquer de inteiros distintos não negativos que se deseja ordenar em ordem crescente. Analise as seguintes afirmativas.

I. Considere o algoritmo Quicksort. Suponha uma execução do algoritmo sobre V tal que a cada sorteio do pivot, a mediana do (sub)problema em questão é escolhida. Então, a complexidade dessa execução é $O(n\ lg\ n)$.

II. Considere o algoritmo de ordenação Quicksort. Suponha uma execução do algoritmo sobre V tal que a cada sorteio do pivot, os dois subproblemas gerados têm tamanho $\frac{1}{10}$ e $\frac{9}{10}$ respectivamente do tamanho do (sub)problema em questão. Então, a complexidade dessa execução é $O(n^2)$.

III. Considere o algoritmo Mergesort. A complexidade do pior caso do algoritmo é $O(n\ lg\ n)$ e a complexidade do melhor caso (vetor já está ordenado) é $O(n)$.

IV. Considere o algoritmo Heapsort. A complexidade do pior caso do algoritmo é $O(n\ lg\ n)$ e a complexidade do melhor caso (vetor já está ordenado) é $O(n)$.

V. Se para todo i, v_i é $O(n)$, então a complexidade do algoritmo Countingsort é $O(n)$.

A partir dos dados acima, pode-se concluir que estão corretas

(A) apenas as afirmativas I e II.
(B) apenas as afirmativas I, II e III.
(C) apenas as afirmativas I, III e V.
(D) apenas as afirmativas III, IV e V.
(E) apenas as afirmativas I e V.

16. [POSCOMP 2002] Quais dos algoritmos de ordenação listados abaixo possuem tempo no pior caso e tempo médio de execução proporcional a $O(n\ log\ n)$?

(A) Bubblesort e Quicksort.
(B) Quicksort e Mergesort.
(C) Mergesort e Bubblesort.
(D) Heapsort e ordenação por seleção.
(E) Mergesort e Heapsort.

17. [POSCOMP 2002] Um estudante universitário propôs o seguinte algoritmo de ordenação, chamado SuperMerge, similar ao Mergesort: divida o vetor em 4 partes do mesmo tamanho; ordene recursivamente cada uma das partes; e depois intercale-as por um procedimento semelhante ao procedimento de intercalação do Mergesort. Qual das alternativas abaixo é verdadeira?

(A) SuperMerge não está correto. Não é possível ordenar quebrando o vetor em 4 partes.
(B) SuperMerge está correto, mas consome tempo $O(Mergesort)$.
(C) SuperMerge está correto, mas consome tempo maior que $O(Mergesort)$.
(D) SuperMerge está correto, mas consome tempo menor que $O(Mergesort)$.
(E) Nenhuma das afirmativas acima está correta.

18. [POSCOMP 2013] Sobre a escolha adequada para um algoritmo de ordenação, considere as afirmativas a seguir.

I. Quando os cenários de pior caso for a preocupação, o algoritmo ideal é o HeapSort.

II. Quando o vetor apresenta a maioria dos elementos ordenados, o algoritmo ideal é o InsertionSort.

III. Quando o interesse for um bom resultado para o médio caso, o algoritmo ideal é o QuickSort.

IV. Quando o interesse é o melhor caso e o pior caso de mesma complexidade, o algoritmo ideal é o BubbleSort.

Assinale a alternativa correta.

(A) Somente as afirmativas I e II são corretas.
(B) Somente as afirmativas I e IV são corretas.
(C) Somente as afirmativas III e IV são corretas.
(D) Somente as afirmativas I, II e III são corretas.
(E) Somente as afirmativas II, III e IV são corretas.

19. [POSCOMP 2011] Com relação aos métodos de ordenação, relacione a coluna da esquerda com a coluna da direita.

(I) Inserção	(A) Encontra o menor elemento e o troca com a primeira posição, depois o segundo menor com a segunda posição e assim sucessivamente ($n-1$ vezes).
(II) Seleção	(B) A partir do segundo elemento, este deve ser colocado na sua posição correspondente (entre os elementos já analisados, como ao se organizarem as cartas de baralho na mão do jogador). Repete-se o procedimento até o último elemento.
(III) QuickSort	(C) Escolhe-se um ponto de referência (pivô) e separam-se os elementos em 2 partes: à esquerda, ficam os elementos menores que o pivô, e à direita, os maiores. Repete-se este processo para os grupos de elementos formados (esquerda e direita) até que todos os elementos estejam ordenados.
(IV) MergeSort	(D) Divide-se o grupo de elementos ao meio, repete-se a divisão para cada um dos subgrupos, até que cada subgrupo tenha apenas 1 elemento. Nesse ponto, faz-se o reagrupamento dos subgrupos comparando os elementos e trocando, se necessário, para que eles fiquem ordenados. Repete-se este procedimento até restar um só grupo de elementos.

Assinale a alternativa que contém a associação correta.

(A) I-A, II-B, III-D, IV-C.
(B) I-B, II-A, III-D, IV-C.
(C) I-C, II-B, III-D, IV-A.
(D) I-B, II-A, III-C, IV-D.
(E) I-A, II-B, III-C, IV-D.

20. A tabela abaixo apresenta um quadro comparativo do tempo total real para ordenar de forma crescente arranjos com 500, 5.000, 10.000 e 30.000 registros que se encontram inicialmente na ordem crescente. O método que levou menos tempo real para executar recebeu o valor 1 e os outros receberam valores relativos a ele.

Método	**500**	**5.000**	**10.000**	**30.000**
A	1	1	1	1
B	128	1.524	3.066	–
C	4,1	6,3	6,8	7,1
D	12,2	20,8	22,4	24,6

Com base nos dados mostrados na tabela acima, assinale a alternativa que contém a associação correta entre os métodos A, B, C e D, e conhecidos métodos de ordenação por comparação de chaves.

(A) A-Heapsort, B-Inserção, C-Quicksort, D-Seleção.
(B) A-Inserção, B-Quicksort, C-Heapsort, D-Seleção.
(C) A-Inserção, B-Seleção, C-Quicksort, D-Heapsort.
(D) A-Quicksort, B-Seleção, C-Inserção, D-Heapsort.
(E) A-Seleção, B-Quicksort, C-Heapsort, D-Inserção.

21. A tabela abaixo mostra um exemplo de aplicação de um algoritmo de ordenação, sendo quatro iterações suficientes para a ordenação desejada.

vetor inicial	40 37 95 42 39 51 60
após 1a iteração	37 40 42 39 51 60 95
após 2a iteração	37 40 39 42 51 60 95
após 3a iteração	37 39 40 42 51 60 95
após 4a iteração	37 39 40 42 51 60 95

Qual foi o algoritmo de ordenação utilizado para ordenar o vetor?

(A) Seleção.
(B) Inserção.
(C) BubbleSort.
(D) HeapSort.
(E) Nenhum deles.

ANEXO

```
TESTE (A,n,i)
// A : vetor de entrada A[1..n]
// n : número de elementos do vetor A
// i : posição no vetor A
r = 2i + 1
l = 2i
maior = i
se (l <= n) e (A[l] > A[maior])
   maior = l
se (r <= n) e (A[r] > A[maior])
   maior = r
se (maior != i)
   troca A[i] com A[maior]
   TESTE (A, n, maior)
```

```
QUICK-SORT (A,p,r)
// A : vetor de entrada A[1..n]
// p : primeira posição do vetor A
// r : última posição do vetor A
se (p < r) então
   q = PARTITION (A, p, r)
   QUICK-SORT (A, p, q - 1)
   QUICK-SORT (A, q + 1, r)
```

```
PARTITION (A,p,r)
x = A[r]
i = p - 1
para (j = p) até (r - 1) faça
   se (A[j] <= x) então
      i = i + 1
      troca A[i] com A[j]
troca A[i + 1] com A[r]
retorne (i + 1)
```