Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Faculdade de Computação
Análise de Algoritmos

Algoritmos de Ordenação

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

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

b) Ilustre e/ou explique o funcionamento do método HeapSort para ordenar de forma crescente o vetor $A = [10\ 13\ 12\ 17]$. O procedimento HEAPIFY deve ser usado para manter a condição de *heap* máximo.

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 do 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 do 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 seg) 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. O algoritmo C apresentou complexidade quadrática em três situações identificadas com "–".

Algoritmo	**Aleatória**	**Crescente**	**Decrescente**	**Repetida**
A	0,90	0,80	0,80	0,80
B	1,60	0,75	0,75	0,10
C	0,65	–	–	–

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? Justifique sua resposta.

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 do procedimento de ordenação realizado no item (a). Justifique sua resposta.

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

6. Verifique se a sequência S abaixo é um *heap*:

33 32 28 31 26 29 25 30 27

Sabe-se que todo vetor ordenado é um *heap*. Caso S não seja um *heap*, transforme-a em um *heap* por meio de algum método de ordenação.

7. [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.

8. [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) $\Omega(n^2)$.

9. [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 $\Omega(n\ log\ n)$.

10. 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.

11. [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 $O(n\ lg\ n)$.

(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 $O(n\ lg\ n)$.

(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 $O(n\ lg\ n)$.

12. 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.

13. 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.

14. Invente um vetor-exemplo de entrada para provar que a ordenação por seleção é um método instável. Mostre os passos de execução do algoritmo até que a estabilidade seja violada. Note que quanto menor for o vetor que você inventar, mais rápido você vai resolver a questão.

15. [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, considere as afirmativas a seguir.

I. O tempo de execução dos algoritmos Quicksort e Mergesort para ordenar V é $\Omega(n\ lg\ n)$ 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 $O(n\ lg\ n)$ quando $m = O(n)$.

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

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.

16. [POSCOMP 2007 - EDITADA] Seja $V =< v_1, ..., v_n >$ uma lista qualquer de inteiros distintos que se deseja ordenar em ordem *não decrescente*. 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 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.

17. [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.

18. [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.

19. [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.

20. [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.

21. 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 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.

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

vetor inicial	O R D E N A
após 1a iteração	A R D E N O
após 2a iteração	A D R E N O
após 3a iteração	A D E R N O
após 4a iteração	A D E N R O
após 5a iteração	A D E N O R

Qual foi o algoritmo de ordenação usado para ordenar o vetor inicial?

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

ANEXO

```
HEAPIFY (A,n,i)
// A : vetor de entrada A[1..n]
// n : número de elementos do vetor A
// i : posição no vetor A
1. r = 2i + 1
2. l = 2i
3. maior = i
4. se (l <= n) e (A[l] > A[maior])
5.    maior = l
6. se (r <= n) e (A[r] > A[maior])
7.    maior = r
8. se (maior != i)
9.    troca A[i] com A[maior]
10.   HEAPIFY (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
1. se (p < r) então
2.    q = PARTITION (A, p, r)
3.    QUICK-SORT (A, p, q - 1)
4.    QUICK-SORT (A, q + 1, r)
```

```
PARTITION (A,p,r)
1. x = A[r]
2. i = p - 1
3. para (j = p) até (r - 1) faça
4.    se (A[j] <= x) então
5.       i = i + 1
6.       troca A[i] com A[j]
7. troca A[i + 1] com A[r]
8. retorne (i + 1)
```