Algoritmos e Estrutura de Dados II 2/2018 Prof. Luis Cuevas

1. Assinale a alternativa que corresponde a um algoritmo de ordenação de vetores que adota a estratégia de divisão e conquista.
a)Ordenação por seleção; b)Ordenação bolha (Bubble Sort);, c)Ordenação por inserção; d)Ordenação QuickSort.

2. O algoritmo de ordenação baseado em vários percursos sobre o array, realizando, quando necessárias, trocas entre pares de elementos consecutivos denomina-se método:
a) das trocas (exchange sort);b) da inserção (insertion sort);c) da bolha (bubble sort);d) da seleção (selection sort);e) da permuta (permutation sort).

3. Um algoritmo de ordenação é executado através dos seguintes passos: (I) escolha de um elemento da lista, denominado pivô; (II) rearranjo da lista, de forma que todos os elementos anteriores ao pivô sejam menores do que ele e que todos os elementos posteriores ao pivô sejam maiores do que ele; e, também, de modo que o pivô, ao fim do processo, esteja em sua posição final, havendo duas sublistas não ordenadas; (III) ordenação recursiva das sublistas dos elementos menores e dos elementos maiores. Que algoritmo é esse?
A Quick Sort; B Merge Sort; C Bubble Sort; D Insertion Sort; E Selection Sort

4. Algoritmo de ordenação é um algoritmo que coloca os elementos de uma dada sequência em uma certa ordem. Assinale a alternativa que NÃO é considerada um algoritmo de ordenação.
A Bubble Sort; B Merge Sort; C Column Sort; D Selection Sort; E Quick Sort.

5. Em seu pior caso, o tempo de ordenação do algoritmo Quicksort sobre um arranjo de n números é igual a
A $O(n^2)$; B O(n).; C O(n+1); D O(n logn); E O(n/2).

6. O seguinte algoritmo de ordenação é estável? Exemplifique.

```
void countSort(int vetor[], int INTERV, int tamanho)
{
    char output[tamanho];
    int contar[INTERV + 1], i;
    for (i = 0; i <= INTERV; ++i)
        contar[i] = 0;
    for(i = 0; i < tamanho; i++)
        contar[vetor[i]]++;
    for (i = 1; i <= INTERV; ++i)
        contar[i] += contar[i-1];
    for (i = 0; i < tamanho; ++i)
    {
        output[contar[vetor[i]]-1] = vetor[i];
        contar[vetor[i]]--;
    }
    for (i = 0; vetor[i]; ++i)
        vetor[i] = output[i];
}
```

7. Para ordenar o vetor (4, 3, 1, 5, 2) usando o algoritmo Quick_sort serão feitas:
a) 7 comparações e 5 trocas; b) 8 comparações e 4 trocas;
c) 6 comparações e 4 trocas; d) 7 comparações e 4 trocas;
e) 6 comparações e 5 trocas

8. O algoritmo Quick sort NÃO é estável. Demostre isso com um exemplo.

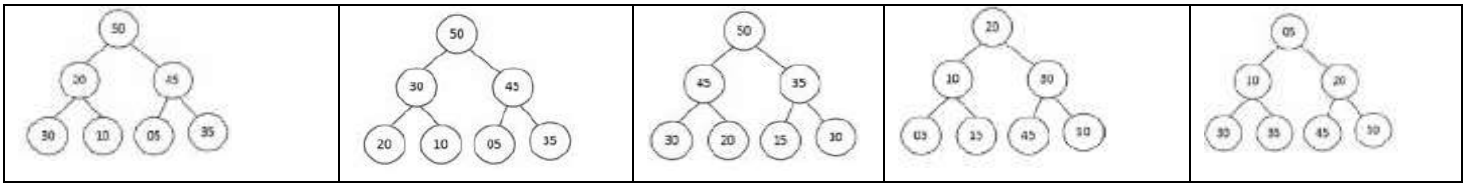
9. Ao usar os algoritmos Selection sort e Insertion sort para ordenar o vetor (4, 3, 1, 5, 2) é possível afirmar que:
a) Os dois precisam fazer a mesma quantidade de comparações; b) O Selection sort faz mais comparações que Insertion Sort; c) O Insertion Sort faz mais comparações que Selection sort

10 Para ordenar de forma crescente uma lista ordenada em ordem crescente usando o algoritmo Bubble Sort você tem:
a) Fazer cero comparações e cero movimentações; b) Fazer a mesma quantidade de comparações que movimentações;
c) Fazer cero movimentações e varias comparações; d) Fazer cero comparações e varias movimentações.

11. Na primeira fase do método de ordenação usando uma árvore binária (heapsort), deve ser montada uma heap a partir do vetor com os dados que se deseja ordenar, conforme os mostrados na tabela a seguir.

	20	10	05	30	50	45	35
indice	1	2	3	4	5	6	7

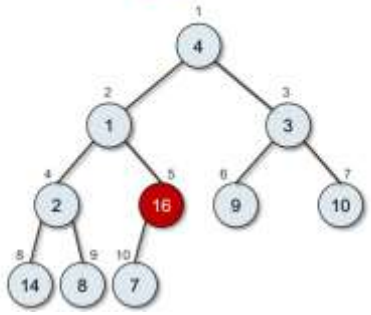
Considerando essas informações, assinale a opção que apresenta a heap max (toda a árvore ordenada como uma heap) formada ao final dessa fase.



```
HEAPFY(A,i)
 1: l ← LEFT(i);     // LEFT(i)  = 2i
 2: r ← RIGHT(i);    // RIGHT(i) = 2i + 1
 3: m ← i;
 4: if l ≤ tam-heap(A) & A[l]>A[i] then
 5:   m ← l;
 6: end if
 7: if r ≤ tam-heap(A) & A[r]>A[m] then
 8:   m ← r;
 9: end if
10: if m ≠ i then
11:   trocar(A[i],A[m]);
12:   HEAPFY(A,m);
13: end if
```

```
BUILD-HEAP(A,n)
 1: tam-heap(A) ← n;
 2: for i ← ⌊n/2⌋ downto 1 do
 3:   HEAPFY(A,i);
 4: end for
```

	1	2	3	4	5	6	7	8	9	10
A	4	1	3	2	16	9	10	14	8	7




```
HEAPSORT(A,n)
 1: BUILD-HEAP(A,n);
 2: for i ← n downto 2 do
 3:   troca(A[1],A[i]);
 4:   tam-heap(A)← tam-heap(A) - 1;
 5:   HEAPFY(A,1);
 6: end for
```

1. Um arranjo que está em seqüência ordenada é um heap mínimo.
2. A seqüência (23, 17, 14, 6, 13, 10, 1, 5, 7, 12) é um heap máximo.
3. Ilustre a operação HEAPIFY(A,3) sobre o arranjo A=(27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4,8,9,0)
4. Ilustre a operação BUILDHEAP sobre o arranjo A=(5, 3, 17, 10,84, 19,6,22,9)
5. Ilustre a operação HEAPSORT sobre o arranjo A=(5, 13, 2,25,7,17,20,8,4)

1. A função *particiona* retorna o valor do pivot para ordenar um vetor usando o algoritmo Quicksort. No seguinte código foi programada essa função, mas não se tem os resultados esperado.

```
int particiona_CLRS (int v[], int p, int r) {
    int c = v[r], i = p, j, t;
```

```
    for (j = p; j < r; j++) {
       if (v[j] <= c) {
           t = v[i], v[i] = v[j], v[j] = t;
           i++; }
     }
     return i;
}
```

a) Identifique porque essa função não dá o resultado esperado.
b) Propor uma função para obter os resultados esperados.

2. Na ordenação por contagem (counting sort) da sequencia { 11, 9, 7, 5, 3, 1 } o algoritmo para ordenar decrescentemente tem que fazer
a. Zero comparações e zero troca.
b. Seis comparações e cinco trocas.
c. Cinco comparações e seis trocas.

3. O algoritmo Heapsort, quando usado para ordenar uma coleção n elementos distintos, possui, respectivamente, complexidade de melhor caso e de pior caso iguais a
a) O(1) e O(n log n)
b) $O(n^2)$ e $O(n^4)$
c) O(n) e $O(n^2)$
d) O(n log n) e O(n log n)
e) O(n log n) e $O(n \log n^4)$

4. Analise as proposições abaixo sobre algoritmos e estrutura de dados:
I. Os métodos de ordenação por inserção e bolha possuem complexidade $O(n^2)$ em relação ao número de comparações.
II. Embora $O(n^2)$, o método de ordenação por inserção possui complexidade Ω(n) em relação ao número de comparações.
III. O método de ordenação por inserção, assim como o Quicksort, é estável.
IV. O método de ordenação Quicksort tem complexidade $O(n^2)$ em seu pior caso.
Assinale a alternativa CORRETA:
a) Somente as proposições I e III estão corretas.
b) Somente as proposições I e IV estão corretas.
c) Somente as proposições II e III estão corretas.
d) Somente as proposições I, II e IV estão corretas.

5. Dada a sequência numérica (15,11,16,18,23,5,10,22,21,12) para ordenar pelo algoritmo Selection Sort, qual é a sequência parcialmente ordenada depois de completada a quinta passagem do algoritmo?
a) [15, 11, 16, 18, 12, 5, 10, 21, 22, 23]
b) [15, 11, 5, 10, 12, 16, 18, 21, 22, 23]
c) [15, 11, 16, 10, 12, 5, 18, 21, 22, 23]
d) [10, 11, 5, 12, 15, 16, 18, 21, 22, 23]
e) [12, 11, 5, 10, 15, 16, 18, 21, 22, 23]

1. O algoritmo Bubble Sort é popular, mesmo que ineficiente. Usando-se esse algoritmo para ordenar uma tabela, alocada seqüencialmente, em ordem crescente contendo os números [5, 4, 1, 3, 2] serão feitas:

a) 10 comparações e 8 trocas
b) 10 comparações e 9 trocas
c) 10 comparações e 10 trocas
d) 16 comparações e 9 trocas
e) 16 comparações e 10 trocas

1. Há situações em que é necessário ordenar os dados. Para esse procedimento existem algoritmos de ordenação. Um deles consiste na ordenação onde são efetuadas comparações entre os dados armazenados em um vetor de tamanho n, e cada elemento de posição i é comparado com o elemento de posição i+1, sendo que quando a ordenação procurada é encontrada, uma troca de posições entre os elementos é feita. Qual o nome deste tipo de algoritmo de ordenação?

a) Algoritmo de ordenação rápida (quick sort).
b) Algoritmo de ordenação por intercalação (merge sort).
c) Algoritmo de ordenação por troca (bubble sort).
d) Algoritmo de ordenação por inserção (insertion sort).
e) Algoritmo de ordenação por seleção (selection sort).

2. Dado o vetor A={5,1,6,2,3}, mostre cada um dos passos (vetores resultantes) quando chama-se a função constHeap(A,5)

```
void constHeap(int arr[], int n)
{
    tamHeap=n;
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr,i);
}

void heapify(int arr[], int i)
{
    int l = 2*i + 1;
    int r = 2*i + 2;
    int m = i;
    int temp;
    if (l < tamHeap && arr[l] > arr[m])
        m = l;
    if (r < tamHeap && arr[r] > arr[m])
        m = r;
    if (m != i)
    {
        temp=arr[i];
        arr[i]=arr[m];
        arr[m]=temp;
        heapify(arr, m);
    }
}
```