Prova Substitutiva – SIN5013 – Análise de Algoritmos e Estrutura de Dados

Nome:_____ **Número USP:** _____

OBSERVAÇÃO: Escreva seu nome e número USP também na folha de rascunho que deverá ser entregue juntamente com a prova. Para as questões de completar ou assinalar as alternativas Verdadeiras/Falsas, cada alternativa assinalada incorretamente anulará uma assinalada corretamente dentro da mesma questão. Isto se aplica às questões 1c, 2, 4 e 5 (recomenda-se deixar a alternativa em branco, caso não saiba a resposta).

1 (1,2). Dado o seguinte algoritmo que recebe um arranjo de inteiros como entrada e teoricamente retorna o arranjo ordenado.

```
public static int[] ordenar(int[] entrada){
      int n = entrada.length;
      for (int x=0;x<n-1;x++){
            for (int y=1;y<n;y++){
                  if (entrada[y-1]>=entrada[y]){
                        int temp = entrada[y];
                        entrada[y] = entrada[y-1];
                        entrada[y-1] = temp;
                  }
            }
      }
      return entrada;
}
```

1a(0,4) Escreva a função **T(n)** (número exato de vezes que a comparação que está na linha destacada do ***if*** será executada, em relação à n, isto é, o tamanho do arranjo de entrada), considerando o **melhor caso**.

1b)(0,2) Escreva a complexidade assintótica θ dessa função: T(n) ϵ θ( )

1c) (0,6) Assinale V para verdadeiro e F para falso para as seguintes afirmações sobre o algoritmo:

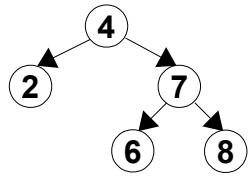
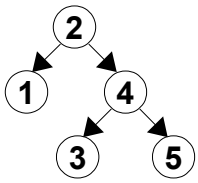
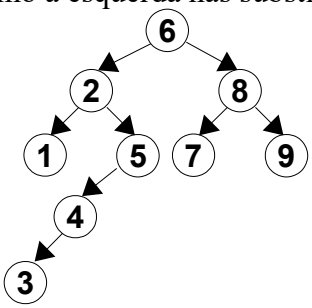
[ ] Esse método de ordenação é *in-place*.
[ ] Esse método de ordenação é estável.
[ ] Esse método ordena corretamente (e de maneira crescente) o arranjo de entrada.

Questão 2 (0,8) Dado o seguinte arranjo:

1	2	3	4	5	6	7	8

Preencha o arranjo abaixo com o resultado da execução do método ConstroiHeapMaximo, executado sobre o arranjo acima (ou seja, o resultado abaixo deve conter o Heap Máximo resultante da execução do método ConstroiHeapMaximo, conforme visto em sala).

Questão 3 (0,9)
Dadas as árvores abaixo, desenhe a árvore **resultante** de cada operação (não é necessário desenhar figuras intermediárias, apenas a árvore resultante final):

a) Insira o nó com chave=5 (inserção em árvores AVL) 	
b) Insira o nó com chave=6 (inserção em árvores AVL): 	
c) Remova o nó com chave=6 (remoção em árvore de busca binária, utilizando maior/máximo a esquerda nas substituições): 	

Questão 4 (1,6) Assinale cada sentença como verdadeira (V) ou falsa (F). Lembre-se das diferenças entre árvore binária, árvore de busca binária [ou árvore binária de busca], e árvore AVL. Cada alternativa assinalada incorretamente anulará uma alternativa assinalada corretamente.

[ ]	Em um **heap máximo** qualquer elemento do heap sempre terá seu valor maior ou igual ao valor de seus filhos.
[ ]	O nó com o terceiro maior valor de um **heap máximo** (que tenha ao menos três elementos) sempre será um dos filhos da raiz do heap.
[ ]	Em uma **árvore binária** sempre é possível fazer busca binária (isto é, se o elemento procurado existir ele sempre será encontrado pela busca binária, a qual, dependendo do valor do elemento atual, irá continuar a busca apenas para a subárvore à direita ou à esquerda).
[ ]	Em uma **árvore AVL** sempre é possível fazer busca binária (ou seja, se o elemento procurado existir ele sempre será encontrado) e há garantia de que o algoritmo será eficiente.
[ ]	Toda **árvore AVL** é uma **árvore binária de busca**.
[ ]	Em qualquer nó de uma **árvore AVL** a diferença de balanceamento (ou seja, a diferença de altura) entre a altura da subárvore a esquerda e a altura da subárvore a direita é exatamente 1.
[ ]	Em uma **árvore binária de busca** a busca binária, no pior caso, tem complexidade assintótica $\Theta(n)$ onde n é o número de nós da árvore.
[ ]	Dada uma **árvore binária de busca**, o nó de menor valor desta árvore sempre será uma folha.

Questão 5 (1,0). Você deseja ordenar um conjunto de ***n*** **números**, cujos **valores variam de 1 a n^3**. Qual a complexidade assintótica θ **(no pior caso)** em relação a n dos seguintes algoritmos para se ordenar este conjunto? Considere que cada valor do conjunto a ser ordenado tem uma quantidade de no máximo **n dígitos**. Cada resposta preenchida incorretamente anulará uma resposta correta.

Algoritmo	complexidade assintótica θ (**no pior caso**)
CountingSort	
RadixSort, utilizando CountingSort como algoritmo auxiliar	
RadixSort, utilizando BubbleSort como algoritmo auxiliar	
HeapSort	
QuickSort	

6 (1,5) Resolva a seguinte equação de recorrência:

$T(1) = 0$ para $n=1$

$T(n) = T(n-1) + n - 1$ para $n > 1$

Utilize a folha de almaço para implementar os métodos solicitados nas questões 7 e 8 (e/ou métodos auxiliares)

Questão 7 (2,0) Implemente um método que retorne o número de vértices do maior componente conexo de um grafo (não dirigido/direcionado, conforme estudado neste semestre). Escolha a representação de grafo que irá utilizar dentre as duas providas abaixo (matriz de adjacências ou arranjo de nós no qual cada nó possui um arranjo de vizinhos). O seu método deverá retornar o número de vértices do maior componente conexo (isto é, do componente que possui o maior número de vértices) do grafo. Você pode assumir que, antes do início da execução de seu método, cada valor do arranjo visitados e o atributo visitado de cada nó estará com valor ***false***. O grafo possuirá um ou mais nós (o atributo *numNos* conterá o número de nós do respectivo grafo). Se necessário, implemente métodos auxiliares.

```
public class Grafo {
     static int numNos;
     static boolean[][] adjacencias;
     static boolean[] visitados;
     static No nos[];
}
```

```
class No{
     int numVizinhos;
     No[] vizinhos;
     boolean visitado;
}
```

Questão 8 (1,0) Dada uma nova estrutura de dados que chamaremos de FILAX e que se comporta da seguinte maneira: permite inserções no final de sua estrutura e remoções em seu início (assim como uma fila convencional), porém cada um de seus registros possui 2 ponteiros: um para o elemento posterior (prox) e um para o posterior do posterior (superProx). Além disso, essa estrutura possui **um nó cabeça** do tipo REGISTRO (que fica sempre em seu início). Esta estrutura **é circular**. Vale lembrar que após ser criado na função inicializarFilaX o nó cabeça nunca será excluído. Implemente a seguinte função:
`bool excluirInicio(FILAX* f)`, que deve retornar false caso não haja nenhum elemento válido na fila; caso contrário deverá excluir o primeiro elemento da fila (acertar ponteiros, liberar memória, etc) e retornar *true*;
As seguintes definições já foram dadas:

```
#include <stdio.h>
#include <malloc.h>
# define true 1
# define false 0

typedef int bool;

typedef struct tempR{
  int valor;
  struct tempR* prox;
  struct tempR* superProx;
} REGISTRO;

typedef REGISTRO* PONT;

typedef struct {
  PONT cabeca;
} FILAX;
```

```
void inicializarFilaX(FILAX* f){
  f->cabeca=(PONT) malloc(sizeof(REGISTRO));
  f->cabeca->prox = f->cabeca;
  f->cabeca->valor = -1;
  f->cabeca->superProx = f->cabeca;
}

PONT ultimoFila(FILAX* f, PONT* penultimo){
  PONT ult = f->cabeca;
  *penultimo = f->cabeca;
  while (ult->prox != f->cabeca) {
    *penultimo = ult;
    ult = ult->prox;
  }
  return ult;
}
```

Dica: pelo fato da estrutura ser circular é necessário ter bastante atenção com os ponteiros.