* **Pergunta 1**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Na disciplina Fundamentos de Estruturas de Dados estudamos as características de algoritmos iterativos e recursivos. No exemplo a seguir a apresentamos um método que você deverá analisar detalhadamente.       void metodoEnigma(int start, int lim) {          if (start <= lim && start%2==0) {              metodoEnigma(start + 2, lim);              System.out.print("  " + start);          }      }    Qual das alternativas a seguir explica melhor as características deste método? |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 2**

0,2 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Um algoritmo frequente com vetores, conhecido como busca binária, é utilizado para procurar um determinado item dentro de um vetor ordenado. Este algoritmo retornará -1 se o item não foi encontrado, e deverá retornar a posição do vetor onde se encontra o item buscado caso o mesmo for encontrado.    Supondo uma busca binária em um vetor de valores reais, ordenado de menor a maior (em ordem crescente), qual dos métodos a seguir resolve corretamente a busca binária em um vetor?        // lambda      public int buscaBinaria(float item, int ini, int fim, float v[]) {          if (ini > fim) return -1;          else {              int m = (ini + fim) / 2;              if (m == item) return m;              if (v[m] < item) return buscaBinaria(item, m + 1, fim, v);              else return buscaBinaria(item, ini, m - 1, v);          }      }        // alfa      public int buscaBinaria(float item, int ini, int fim, float v[]) {          if (ini > fim) return -1;          else {              int m = (ini + fim) / 2;              if (v[m] == item) return m;              if (v[m] < item) return buscaBinaria(item, m + 1, fim, v);              else return buscaBinaria(item, ini, m - 1, v);          }      }        // beta      public int buscaBinaria(float item, int ini, int fim, float v[]) {          if (ini > fim) return -1;          else {              int m = fim / 2;              if (v[m] = item) return m;              if (v[m] < item) return buscaBinaria(item, m + 1, fim, v);              else return buscaBinaria(item, ini, m - 1, v);          }      }        // pi      public int buscaBinaria(float item, int ini, int fim, float v[]) {          if (ini > fim) return -1;          else {              int m = (ini + fim) / 2;              if (v[m] == item) return v[m];              if (v[m] < item) return buscaBinaria(item, m + 1, fim, v);              else return buscaBinaria(item, ini, m - 1, v);          }      }        // sigma      public int buscaBinaria(float item, int ini, int fim, float v[]) {          if (ini < fim) return -1;          else {              int m = (ini + fim) / 2;              if (v[m] == item) return m;              if (v[m] > item) return buscaBinaria(item, m + 1, fim, v);              else return buscaBinaria(item, ini, m - 1, v);          }      } |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 3**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Colocar valores aleatórios dentro de um vetor foi um assunto estudado nesta disciplina.  Temos a tarefa de construir um método, que recebe um vetor de valores reais, e deve gerar e colocar valores aleatórios apenas nas posições pares deste vetor, retornando o vetor alterado.  Em sua opinião, qual dos métodos a seguir implementa corretamente esta tarefa? |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 4**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Seja um pilha **pp**, que guarda modelos de automóveis, inicialmente no estado:  **pp: [ a4, fiesta ]**  utilizando a convenção: pilha: [ a, b, c, d, **topo** ], como ficará esta pilha depois de executar a sequência de comandos a seguir?         Object a = pp.top();         Object b = pp.pop();         pp.push("outlander");         pp.push("camaro");         pp.push("passat");         pp.push(b);         pp.push("captiva");         pp.push(pp.pop());         pp.push("gol"); |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 5**

0 em 0,4 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Um método com o cabeçalho: public void **processa**(Pilha p1, Pilha p2, Pilha p3); recebe duas pilhas p1 e p2 e deverá eliminar todos os elementos destas pilhas e adicionar apenas aqueles que são maiores que um (1) na pilha p3. Considere que a pilha p3 está vazia no momento de efetuar uma chamada a **processa**.  Em sua opinião, qual é a implementação correta para este método? |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 6**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Um algoritmo frequente com vetores, conhecido como busca sequencial, é utilizado para procurar um determinado item dentro de um vetor. Este algoritmo retorna -1 se o item não foi encontrado, e deverá retornar a posição do vetor onde se encontra o item buscado caso for encontrado.  Qual dos métodos a seguir resolve corretamente a busca sequencial em um vetor?      // alfa      int buscaSequencial(double v[], float item) {          for (int i = 0; i < v.length; i++) {              if (v[i] == item) {                  return item;              }          }          return -1;      }        // beta      int buscaSequencial(double v[], float item) {          for (int i = 0; i < v.length; i++) {              if (v[i] < item) {                  i++;              }          }          return v[i-1];      }        // pi      int buscaSequencial(double b[], float item) {          for (int i = 0; i < b.length; i++) if (b[i] == item) return b[i];          return -1;      }      // lambda      int buscaSequencial(double a[], float item) {          for (int i = 0; i < a.length; i++) if (a[i] == item) return i;          return -1;      }      // sigma      int buscaSequencial(double a[], float item) {          for (int j = 0; j < a.length; j++) {              if (a[j] == item) return -1;          }          return j-1;      } |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 7**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Utilizando seus conhecimentos quanto a métodos de ordenação e estruturas de dados, analize o método a seguir e selecione a alternativa correta.                public Object **enigma**(Object newvalue) {                              if( top == MAXIMO-1 ) {       return null;                              }                              else {                                             top = top + 1;                                             valores[top] = newvalue;                                             return newvalue;                              }                } |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 8**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | O método com o cabeçalho mostrado a seguir deverá implementar a lógica para retirar (remover, eliminar) um elemento de uma pilha dinâmica encadeada (ligada).  public Object **removerDaPilha** ( );    Para programar corretamente este método:     * + considere, obrigatoriamente, que existe um atributo na classe Pilha declarado como:  Node topoPilha;  que guarda o endereço do topo da pilha;   + considere que a classe Node fornece os métodos públicos getProximo() e getElemento (), além dos métodos set correspondentes;   + considere que os elementos da pilha se encontram armazenados na memória heap, em forma encadeada, enlaçados;   + o método removerDaPilha deverá retornar null se a eliminação não foi possível ou retornar o objeto retirado se a remoção foi satisfatória;   + não poderá considerar que existem outros atributos na classe Pilha, nem chamar outros métodos auxiliares, ou seja, implemente a lógica completa sem depender de métodos como isEmpty(), isFull() ou qualquer outro, mas verifique as condições necessárias.     Qual é, em sua opinião, a implementação correta? |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 9**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Analise cuidadosamente o método **enigma** mostrado a seguir.       private int **enigma**(int arr [], int low, int high) {        int comp = arr[low];        int i = low + 1, f = high;        while (i <= f) {           if (arr[i] <= comp) i++;           else if (comp < arr[f]) f--;           else {              int aux = arr[i];              arr[i] = arr[f];              arr[f] = aux;              i++; f--;           }        }        arr[low] = arr[f];        arr[f] = comp;        return f;     } |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 10**

0,2 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Utilizando seus conhecimentos quanto a métodos de ordenação e estruturas de dados, analize o método a seguir e selecione a alternativa correta.        public void **enigma**(float valores []) {          for (int i = 0; i < valores.length - 1; i++) {              for (int j = 0; j < valores.length - 1 - i; j++) {                if (valores[j] > valores[j+1]) {                      float tmp = valores[j];                      valores[j] = valores[j+1];                      valores[j+1] = tmp;                 }              }          }      } |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 11**

0,2 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | Considerando seus conhecimentos quanto aos métodos de ordenação estudados e quanto ao  funcionamento das estruturas de dados estudadas nesta disciplina e suas características específicas, analise os métodos a seguir e selecione a alternativa com as afirmações que considere corretas.    public Node **enigmaA** (Object vl) {              try {                      if(vl == null) return null;                      Node novo = new Node();                      novo.setValue(vl);                      novo.setNext(p1);                      p1 = novo;                      return novo;              } catch(Exception ex) {                      return null;              }  }     public Object **enigmaB**() {              if (topo == null) return null;              Object valor = topo.getValue();              Node temp = topo;              topo = topo.getNext();              temp = null;              return valor;  } |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 12**

0 em 0,2 pontos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | | |
|  | O método com o cabeçalho mostrado a seguir deverá implementar a lógica para inserir (adicionar) um elemento em uma pilha dinâmica encadeada (ligada).  public Object **inserirNaPilha** ( Object elemento );    Para programar corretamente este método:     * + considere, obrigatoriamente, que existe um atributo na classe Pilha declarado como:  Node topoPilha;  que guarda o endereço do topo da pilha;   + considere que a classe Node fornece os métodos públicos getProximo() e getElemento (), além dos métodos set correspondentes;   + considere que os elementos da pilha se encontram armazenados na memória heap, em forma encadeada, enlaçados;   + o método inserirNaPilha deverá retornar o nodo que foi inserido na pilha;   + não poderá considerar que existem outros atributos na classe Pilha, nem chamar outros métodos auxiliares, ou seja, implemente a lógica completa sem depender de métodos como isEmpty(), isFull() ou qualquer outro, mas verifique as condições necessárias.     Qual é, em sua opinião, uma implementação simplificada correta deste método? |  |  |  |
| |  | | --- | |  | |  |  |  |

* **Pergunta 13**

0 em 0,4 pontos

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | | | | |
|  | Um programa a ser utilizado em uma eleição no Senado definiu a classe **Candidato**, que descreve as características de um candidato à presidência de uma comissão interna para analisar problemas relacionados ao meio ambiente.    Um candidato (classe Candidato) possui quatro dados: nome do candidato, sexo, partido político ao qual ele pertence e a quantidade de votos obtidos na votação para este cargo. Os atributos foram chamados: **nome**, **sexo**, **partido** e **votos,** sendo desenvolvidos todos os métodos na convenção padrão, tais como: get/set, construtores e o método toString.    Devemos construir um método chamado **candidatosHomensDeUmPartidoPolitico**, que retorne todos os dados dos candidatos homens de um determinado partido político especificado no parâmetro **opartido**, que obtiveram pelo menos um voto. Os dados dos candidatos se encontram guardados em uma pilha **p**, também recebida como parâmetro neste método. Na lógica deste método, todos os candidatos serão retirados da pilha p para poder efetuar esta análise. O cabeçalho do método deverá ser, obrigatoriamente:         public String **candidatosHomensDeUmPartidoPolitico**(Pilha p, String opartido)    Em sua opinião, qual é a lógica correta para o método solicitado? Obs: apenas uma das alternativas mostradas é completamente correta. | |  |  |  |
|  |  |  |  |  |