Questão 1 (a)

```
//Método insere nó novo no fim de L
void insereNoFim(ListaLigada L, No novo){
    novo.prox = L.cauda;
    novo.ant = L.cauda.ant;
    L.cauda.ant.prox = novo;
    L.cauda.ant = novo;
}

//método recebe duas listas e retorna concatenação destas
ListaLigada concatena(ListaLigada L1, ListaLigada L2){
    ListaLigada nova_lista = new ListaLigada();
    No itr;
    No aux;


    //copia elementos de L1
    itr = L1.cabeca.prox; //itr comeca com primeiro no
    while(itr != L1.cauda){//enquanto nao chegar no fim
      aux = new No();
      aux.valor = itr.valor; //cria novo no e insere no fim da
      insereNofim(aux,nova_lista);//nova lista
      itr = itr.prox;
    }

    //copia elementos de L2
    itr = L2.cabeca.prox;
    while(itr != L2.cauda){
      aux = new No();
      aux.valor = itr.valor;
      insereNofim(aux,nova_lista);
      itr = itr.prox;
    }

    return nova_lista;
}


//Questao 1 (b)

void remove_valor(ListaLigada L, int valor_remover){
    No itr = L.cabeca.prox;//itr comeca com o primeiro no
    while(itr != L.cauda){//enquanto nao chegar no fim

      if(itr.valor == valor_remover){//se nó tiver valor a ser removido
          itr.ant.prox = itr.prox; //remove no apontado por itr
          itr.prox.ant = itr.ant;  //da lista
      }
      itr = itr.prox;
    }

}
```

Questão 2

```
/*Definição recursiva para média de vetor v:
Se número de elementos do vetor for 1, então devolve elemento.
Se número de elementos do vetor for n>1, então calcule MEDIA
como sendo a média dos n-1 primeiros elementos do vetor.
Retorne ((MEDIA*(n-1))+v[n-1])/n
*/

double calculaMedia(double[] v){
    return calculaRecursivo(v,v.length);
}

//calcula a media dos n elementos de v
double calculaRecursivo(double[] v, int n){
    if(n==1)
      return v[0];
    else{
      double m = calculaRecursivo(v,n-1);
      m = m*(n-1);
      m = (m+v[n-1])/n;
      return m;
    }

}
```

Questão 3

Pode-se assumir que n é variável da classe
ou poderia receber n por parâmetro.
Há espaço no vetor e índice i é válido

```
void insere(int[] v, int valor, int i){
    int j=n; //j recebe a ultima posicao com dados

    while(j>=i){ //desloca dados de i ate n para frente
      v[j+1] = v[j];
      j--;
    }
    v[i] = valor; //põe valor na posição i
}
```

No pior caso i=0, e temos que deslocar elementos
da posição n até 0 para frente. Portanto o laço
executa O(n) vezes que é a complexidade de tempo
neste pior caso.

No melhor caso inserimos o elemento na última posição
e portanto a complexidade de tempo é O(1).

Questão 4

O algoritmo lê caracteres e empilha estes até encontrar um 'X'.
Depois o algoritmos continua lendo caracteres da string, e para
cada caractere lido desempilha um caractere e checa se são iguais.
Todos os caracteres checados devem ser iguais e ao final a pilha
deve estar vazia.

Pseudo-Código:

```
    String s; //string a ser testada da forma aXb
    Pilha = vazia;
    Caracter c1,c2;

    //empilha string a
    c1 = proximo_char_de_s
    while(c1!='X'){
      empilha c1;
      c1 = proximo_char_de_s;
      if(c1 == vazio)//nao ha mais caracteres
        return false;
    }

   //testa se string empilhada é inversa da string restante
    while(existe_char_em_s){
      c1 = proximo_char_de_s;

      if(Pilha == vazia)
         return false;

      c2 = desempilha;
      if(c1 != c2)
          return false;
    }

    if(Pilha == vazia)
      return true;
    else
      return false;
```

Seja n o número de caracteres em s.

O primeiro laço é executado até encontrarmos
o caractere 'X', ou até lermos toda string
(neste caso retornamos false).

O segundo laço é executado até terminarmos
de ler a string, ou descobrirmos que a string
não é da forma aXb.

De qualquer forma, no pior caso, os laços serão
executados até lermos toda a string
de tamanho n.

Portanto a complexidade de tempo do algoritmo
é O(n).