Questão 1 (a)

```
//método insere nó no ínico da lista passada por parâmetro
void insereNoInicio(ListaLigada L, No novo){
    novo.prox = L.cabeca.prox;
     L.cabeca = novo;
}

//método cria nova lista que é a inversa da lista passada por parâmetro
ListaLigada inverte(ListaLigada L){
   No itr = L.cabeca.prox;
   No aux;
   ListaLigada nova = new ListaLigada();
   while(itr != null){ //enquanto não chegar no fim da lista
        aux = new No();
        aux.valor = itr.valor;
        insereNoInicio(nova, aux);
        itr = itr.prox;
   }
   return nova;
}
```

Questão 1 (b)

```
//método retorna nova lista sem os primeiros i nós da lista ligada passada por parâmetro
ListaLigada copia(ListaLigada L, int i){
   ListaLigada nova = new ListaLigada();
   No ultimo = nova.cabeca; //ultimo sempre aponta para último nó da nova lista
   No aux, itr;
   itr = L.cabeca.prox; //itr aponta para primeiro nó da lista

   int j=0;
   while(itr!=null && j<i){ //desconsidera os i primeiros nós
        itr = itr.prox;
        j++;
   }
   while(itr != null){ //copia o restanto dos nós
         aux = new No();
         aux.valor = itr.valor;  //cria novo nó com mesmo valor da lista passada por parâmetro
         ultimo.prox = aux; //insere no fim da nova lista
         ultimo = ultimo.prox;
```

```
        itr = itr.prox;
    }
  return nova;
}
```

Questâo 2

Definição Recursiva. Seja v um vetor com n elementos.
Se n=2 então retorne um vetor de duas posições; na primeira posição ponha o máximo e na segunda o mínimo..
Se n>2, ache de forma recursiva o máximo e mínimo do vetor v considerando os primeiros n-1 elementos.
Cheque se v[n-1] é maior do que o máximo ou menor do que o mínimo, atualize a resposta e retorne.

```
int[] MinMax(int[] v, int n){
  if(n == 2){
       int[] resposta = new int[2];
      if(v[0]>v[1]) { resposta[0] = v[0]; resposta[1] = v[1]; }
      else{resposta[0] = v[1]; resposta[1] = v[0]; }
      return resposta;
   }
  else{
      int[] resposta = MinMax(v,n-1);
      if(v[n-1] > resposta[0])
          resposta[0] = v[n-1];
      if(v[n-1] < resposta[1])
          resposta[1] = v[n-1];
      return resposta;
  }
}
```

Questão 3

```
//A parte utilizada do vetor vai de 0 até n. Temos 0<= i <= j<=n.
void remove(int[] v, int i, int j){
    j = j+1;
    while(j<=n){
        v[i] = v[j];
        i++;
        j++;
    }
    n=i-1; //indica a nova parte útil do vetor.
}
```

No pior caso i=0 e j=0 e o laço é executadao O(n) vezes sendo esta a complexidade do pior caso.
No melhor caso temos i=j=n e portanto a complexidade é O(1) pois não há execução do laço.

Questão 4

```
boolean organizado(int[] v){
    int i;
  for(i=0;i<v.length-1;i++)
       if(v[i+1] != (v[i]+2) )
            return false;
    return true;
}
```

O laço é executado n-1 vezes no pior caso e portanto a complexidade do algoritmo é O(n).