Prova algoritmos e complexidade

Questão 1 (Projeto de algoritmos, 1/3)
Dado uma sequencia de números inteiros $a_1, a_2, \ldots, a_n$ queremos encontrar uma tripla $(i, j, k) \in [n]^3$ com $i \neq j$, $i \neq k$, $j \neq k$, tal que $a_i + a_j = a_k$ ou decidir que tal tripla não existe. Projete um algoritmo que resolve o problema em tempo $o(n^3)$. Prove a corretude do algoritmo e analisa a sua complexidade.

(Observe que: a) a questão pede "oh-minúsculo" e b) os números podem ser negativos.)

Questão 2 (Complexidade da algoritmos, 1/3)
Atribui a cada algoritmo na esquerda a sua menor complexidade na direita (conforme vista em aula; caso foram discutidos diferentes complexidades para o mesmo algoritmo justifica brevemente a atribuição explicando o algoritmo escolhido). Para algoritmos em grafos supõe que o grafo possui $m = \Theta(n^2)$ arestas. Uma dada complexidade pode ser usada mais que uma vez, ou nunca. (Observe que classificações erradas recebem pontos negativos.)

Emparelhamento estável	$O(\log n)$
Algoritmo de Kruskal	$O(n)$
Algoritmo de Strassen	$O(n \log n)$
Transformada rápida de Fourier	$O(n^{1.5})$
Multiplicação de números binários	$O(n^2)$
Algoritmo de Edmonds-Karp	$O(n^2 \log n)$
Algoritmo Húngaro	$O(n^{2.5})$
Algoritmo de Prim	$O(n^3)$
	$O(n^3 \log n)$
	$O(n^4)$
	$O(n^4 \log n)$
	$O(n^5)$

Questão 3 (Analise de algoritmos, 1/3)
Dado um vetor $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_n$ queremos obter a permutação $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$. Como o vetor pode ser grande, a reordenação tem que ser feito com memória extra $O(1)$. O seguinte algoritmo de divisão-e-conquista resolve o problema.

```
R(v_1, v_2, ..., v_2n) :=
  if n = 1
    return
  else
    m := ⌊n/2⌋
    for i ∈ [m] do
      troca v_{m+i} e v_i
    if 2m ≠ n then
      for i ∈ [n − m] do
        troca v_{2m+i} e v_{2m+i+1}
    R(v_1, ..., v_2m)
    R(v_{2m+1}, ..., v_2n)
```

Qual a complexidade do algoritmo? Determina a equação de recorrência para o tempo de R e resolve-la usando algum método vista em aula.

Questão 4 (Projeto de algoritmos, 1/3)
Vankin's mile é um jogo num tabuleiro $n \times n$ para um único jogador. O jogador começa escolher algum campo do tabuleiro. Depois ele pode repetidamente mover uma posição para a direita ou uma posição para baixo. O jogo termina quando a próxima posição cai fora do tabuleiro. Cada campo do tabuleiro possui um número inteiro e o valor do jogo é igual a soma dos campos visitados. O objetivo é encontrar o jogo de maior valor. Por exemplo, no tabuleiro

3	1	-4	1
5	-1	9	2
6	5	-3	-5
-8	-9	-7	-9

começando com 9 indo para baixo, e depois para direita, direita vale $9 - 3 - 5 = 1$ e a melhor solução começa com 3 indo para baixo, direita, direita, direita, direita vale $3 + 5 - 1 + 9 + 2 = 18$.
Projete um algoritmo eficiente (i.e. polinomial) para determinar o jogo de maior valor. Prove a corretude do algoritmo e analisa a sua complexidade.