Lista 15

2-) a) Obs: Assumindo postos ordenados por distâncias.

```
Procedimento acharParadas(postos[1..n], L){
    paradas = [postos[1]]
    ultimaParada = postos[1].distancia;
    i=1
    Enquanto (i <= n){
        Se (postos(i+2).distancia - ultimaParada > L){
            paradas.addFinal(postos[i]);
            ultimaParada = postos[i];
        }
        i++;
    }
    retorna paradas;
}
```

b) O algoritmo sempre pergunta se o carro consegue ir da última parada i até a parada i+2.
Se a resposta for sim, então ele não para na parada i+1, caso contrário, ele para na parada i+1. Dessa forma, ele escolhe um posto apenas se for realmente necessário abastecer. No final, teremos uma lista de paradas ótima.

3- a) Procedimento alocaSalas(salas[1..n], atividades[1..k]){

atividadesOrdenadasInicio = mergeSort(atividades) // ordem início

Para (i de 1 até k){

Para j de (1 até n){

Se (salas[j].atividadeAtual.fim ≤ atividadesOrdenadasInicio[i].começo){

salas[j].atividadeAtual = atividadesOrdenadasInicio[i];

salas[j].atividadesAlocadas.addFinal(atividadesOrdenadas Inicio[i]);

break;

}

}

}

retorna salas;

}

b) Como ele sempre aloca a próxima atividade para a próxima sala disponível iterando a partir da sala 1, é fácil ver que o menor número de salas necessárias estará alocado em qualquer momento de execução do algoritmo.