A)

INPUT:

* n treinen
* Het repareren van trein i kost rti tijd

Def. OPT(i, RT) = subset van treinen 1,…,i voor depot 1 het dichtst bij een totale repareertijd TRT

* Case 1: OPT plaatst rti in depot 2
* OPT selecteert beste set uit {1, 2, …, i-1} met RT
* Case 2: OPT plaatst rti in depot 1
* Nieuwe totale repareer tijd = TRT – rti
* OPT selecteert beste set uit {1, 2, …, i-1} met de nieuwe totale repareer tijd

De treinen worden over depot 1 en 2 verdeeld. Omdat je het verschil van totale repareertijden in depot 1 en 2 zo klein mogelijk wil houden, betekent dat, dat de totale repareertijd van depot 1 zo dicht mogelijk bij moet zitten. Deze waarde gebruiken we aan het begin voor TRT.

In het algoritme wordt gekeken naar de maximale sommatie van rt waardes die niet groter is dan TRT. Er is dus een keuze om rt toe te voegen aan de tijd van depot 1 of niet (dan komt hij bij depot 2). Deze 2 keuzes worden vergeleken op maximale waarde, waarbij recursief wordt gezocht naar welke rt wel en welke niet wordt toegevoegd.

Als een rt groter is dan TRT, dan wordt deze aan depot 2 toegevoegd. Zo blijft de totale tijd in depot 1 ≤ TRT. Als i = 0 dan stopt het algoritme, want alle rt zijn aan een depot toegevoegd. De totale tijd van depot 1 wordt hierbij teruggegeven, logischer wijs is de tijd van depot 2 = - tijd depot 1. Het verschil in tijd tussen de 2 depots kan dus worden gevonden door .

B)

**Input**: n, rt1,…,rtn

TRT = 0

**For** j = 1 **to** n

TRT = TRT + rtj

**End**

TRT = TRT / 2

**For** i = 1 **to** n

**For** trt = 0 **to** TRT

**If** i = 1 **then**

M[i, trt] = 0

**End**

**Else If** rti > trt **then**

M[i, trt] = M[i-1, trt]

**End**

**Else**

M[i, trt] = max(rti + M[i-1, trt – rti], M[i-1, trt])

**End**

**End**

**End**

**Return** 2 \* (TRT – M[n, TRT] )

De looptijd van de eerste for-loop in het algoritme is O(n) en de tweede is O(n \* TRT) vanwege de for-loop die erin staat. De looptijd voor het algoritme volgt dus uit

T(n, TRT) = O(n) + O(TRT) + O(n \* TRT) = O(n \* TRT).

C)

Def. OPT(i, k, w) = subset van uren i,…,T die gebruikt worden om k treinen (k є {1,…,n}) zo goedkoop mogelijk te repareren, waarbij de wachttijd w niet groter dan q mag worden.

* Case 1: OPT gebruikt uur i niet om de volgende trein te repareren.
* Nieuwe wachtijd = w + 1
* OPT selecteert beste set uit {i+1,…,T} voor k treinen met nieuwe wachttijd.
* Case 2: OPT gebruikt uur i om de volgende trein te repareren.
* Nieuwe wachttijd = 0
* Nieuw aantal treinen = k – 1
* Nieuw aantal uren = i + rt(n+1)-k (het zijn namelijk de k laatste treinen uit de rij van n treinen, k = 1 betekent de laatste trein, dus trein n)
* OPT selecteert beste set uit {i + rt(n+1)-k,…, T} voor k-1 treinen en nieuwe wachttijd

D)

**Input:** n, q, T, rt1,…,rtn, c1,…,cn

A = { rtn }

**For** j = n - 1 **to** 1

A = A U {rti + A[last element] }

**End**

A = A U {T}

**For** m = T **to** T – A[last element - 1]

OPT[m, 0, 0] = 0

**End**

**For** k = 1 **to** n

**For** i = T – A[k] **to** T – A[k+1]

c = 0

**For** m = 0 **to** rt(n+1)-k

c = c + ci+m

**End**

**For** w = 0 **to** q

**If** w = q || i + A[k] = T **then**

OPT[i, k, w] = c + OPT[i+ rt(n+1)-k, k-1, 0]

**End**

**Else**

OPT[i, k , w] = min(OPT[i+1,k, w+1], c + OPT[i+ rt(n+1)-k, k-1, 0])

**End**

**End**

**End**

**End**

**Return** OPT[0, n, 0]