4. Tomás quiere viajar de Buenos Aires a Mar del Plata en su flamante Renault 12. Como está preocupado por la autonomía de su vehículo, se tomó el tiempo de anotar las distintas estaciones de servicio que se encuentran en el camino. Modeló el mismo como un segmento de 0 a $M$, donde Buenos aires está en el kilómetro 0, Mar del Plata en el $M$, y las distintas estaciones de servicio están ubicadas en los kilómetros $0 = x_1 \leq x_2 \leq \dots x_n \leq M$.

   Razonablemente, Tomás quiere minimizar la cantidad de paradas para cargar nafta. Él sabe que su auto es capaz de hacer hasta $C$ kilómetros con el tanque lleno, y que al comenzar el viaje este está vacío.

---

*a*) Proponer un algoritmo *greedy* que indique cuál es la cantidad mínima de paradas para cargar nafta que debe hacer Tomás, y que aparte devuelva el conjunto de estaciones en las que hay que detenerse. Probar su correctitud.

**Ante todo voy a asumir que existe camino válido, y no adjuntamos M al resultado final ya que no es una estación (o al menos eso asumo)**

$S = \{0, x_2, ..., x_n, M\}$

$S_a$ es la última parada en la que cargué nafta

La idea es ver $\forall j :: S_j - S_a > C \underset{L}{\rightarrow} a \coloneqq j-1 \land R \cup S_a$

## Algoritmo

```
RutaEficiente(S,C):
  n = |S|

  a = 1
  R = [0]

  for j: 2..n+1:
    if S[j]-S[a] > C:
      a = j-1
      R.append(S[a])

  return R
```

## Correctitud

$S = \{0, x_2, ..., x_n, M\}$

$S_a$ es la última parada en la que cargué nafta

Sea $O$ una solución óptima, queremos ver que nuestra solución greedy $G$ es tan buena como $O$.

**Por inducción fuerte en la cantidad $R$ estaciones a detenerse:**

**Caso base |R|=1:**

Siempre empezamos con 0 nafta, en el kilómetro 0 y sabemos que $S_1 = 0$ por lo que $R = \{0\}$, que es exactamente lo que calcula $G$ en el primer paso. Trivialmente, $G$ es igual de óptima que $O$.

**Paso inductivo:**

**HI:** $\forall j < k$, $G$ es igual de óptima que $O$, o sea, ambas computaron $|R| = j$ en el $j$-ésimo paso.

Por HI sabemos que tanto $G$ como $O$ tienen exactamente $k-1$ estaciones acumuladas en $R$.

En la $k$-ésima estación válida en $G$:

- Tomamos $S_k - S_{k-1} > C \underset{L}{\rightarrow} R = R \cup S_{k-1}$

Si $O$ toma $S_{k-1}$ entonces $G$ es igual de óptima.

Si $O$ no toma $S_{k-1}$ entonces tenemos 2 opciones:

**Caso 1:**

La anterior estación en la que se cargó nafta en la solución $O$ en el paso $k$ es $S_a$:

Un $S_j \leq S_{k-1}$, $S_j$ es una estación válida, pero dijimos que no toma $S_{k-1}$, por lo que agarra un $S_j$ óptimo, y $|R \cup S_j| = |R \cup S_{k-1}|$ por lo que son igual de óptimas.

**Caso 2:** La anterior estación en la que se cargó nafta en la solución $O$ en el paso $k$ es $S_j \neq S_{k-1}$:

Opciones:

**Caso $j > k-1$**

Absurdo! $G$ supone que $S_{k-1}$ es la última estación posible tal que $S_k - S_{k-1} \leq C$, por lo que $j$ no puede ser mayor a $k-1$

**Caso $j < k-1$**

Si $j < k-1$ entonces toma $S_j$, y $|R \cup S_{k-1}| = |R \cup S_j|$ por lo que $G$ es igual de óptimo que $O$

Por lo que el $k$-ésimo paso es óptimo, entonces por inducción, $G$ es tan óptima como $O$.

---

*b)* Dar una implementación de complejidad temporal $O(n)$ del algoritmo del inciso *a)*.

```java
public static List<Integer> rutaEficiente(List<Integer> S, int C) {
    int n = S.size();
    List<Integer> R = new ArrayList<>();

    int a = 0;
    R.add(S.get(0));

    for (int j = 1; j<n ;j++) { //O(n)
        if (S.get(j)-S.get(a) > C) {
            a = j-1;
            R.add(S.get(a));
        }
    }

    return R;
}
```