1. Machete Parcial

1.1. Especificacion lista de Partes

Dada una secuencia l con todos sus elementos distintos, devolver la secuencia de partes; es decir, la secuencia de todas las secuencias incluidas en l, cada una con sus elementos en el mismo orden en que aparecen en l. Características de la entrada:

- por simpleza, supongamos una secuencia de enteros "l"
- todos sus elementos son distintos (ojo: no necesariamente cumplen un orden)

Caracerísticas de la salida

- secuencia de secuencias de enteros res"
- $|res| = 2^{|l|}$ (Recordando algebra 1)
- no hay secuencias repetidas en res
- todos los elementos de res son "sublistas"de l (importante: "...cada una con sus elementos en el mismo orden en que aparecen en l.")

¿Qué sería una "sublista"de l? Supongamos que se llama r, entonces:

- todos los elementos de r están en l
- r no tiene elementos repetidos
- los elementos de r se encuentran en el mismo oreden en que aparecen en l

Ahora sí, especifiquemoslo.

`proc partes` (in l:$seq\langle\mathbb{Z}\rangle$, out res:`Bool`) {
`Pre` $\{\neg hayRepetidos(l)\}$
`Post` $\{|res| = 2^{|l|} \land \neg hayRepetidos(res) \land todasSublistas(res, l)\}$
`pred hayRepetidos` (l:$seq\langle\mathbb{Z}\rangle$) {
$(\exists i, j : \mathbb{Z})(0 \leq i, j < |l| \land i \neq j \land todasSublistas(res, l))\}$
`pred todasSublistas` (res:$seq\langle seq\langle\mathbb{Z}\rangle\rangle$, l:$seq\langle\mathbb{Z}\rangle$) {
$(\forall r : seq\langle\mathbb{Z}\rangle)(r \in res \longrightarrow esSublista(r, l))\}$
`pred esSublista` (r:$seq\langle\mathbb{Z}\rangle$, l$seq\langle\mathbb{Z}\rangle$) {
$estaContenida(r, l) \land \neq hayRepetidos(r) \land respetaOrden(r, l)\}$
`pred estaContenida` (r:$seq\langle\mathbb{Z}\rangle$, l:$seq\langle\mathbb{Z}\rangle$) {
$(\forall x : \mathbb{Z})(x \in r \longrightarrow x \in l)\}$
`pred respetaOrden` (r:$seq\langle\mathbb{Z}\rangle$, l:$seq\langle\mathbb{Z}\rangle$) {
$(\forall i : \mathbb{Z})(0 \leq i < (|r| - 1) \longrightarrow_L estaAntes(r[i], r[i+1], l))\}$
`pred estaAntes` (a,b:$\mathbb{Z}$, l:$seq\langle\mathbb{Z}\rangle$) {
$(\exists i, j : \mathbb{Z})(0 \leq i, j < |l| \land_L s[i] = a \land s[j] = b \land i < j)\}$
}

2. Tiempo de ejecucion en peor caso

2.1. Ejercicio 2a

```
// Si contamos division y resta entonces
void f(vector<int> &v){                 // n = |v|
    int i = v.size();                   // 2
    while(i >= 0){                      // 1, n/2 ciclos
        v[v.size()/2 - i] = i;          // 3
        v[v.size()/2 + i] = i;          // 3
        i--;                            // 1
    }                                   // t(n) = 1 + 4n
}
```

$$t(n) = 3 + 4n = O(n) \tag{1}$$

2.2. Ejercicio 2b

```
// Pre: e pertenece a v1
int f(vector<int> &v1, int e){  // n = |v|
    int i = 0;                  // 1
    while(v1[i] != e){          // 1 (Distinccion), itera mientras no coincida,
                                // busco el peor caso donde recorro todo el vector
                                // n iteraciones
        i++                     // 1 (Incremento)
    }                           // t(n) = 1 + 2n
    return i;
}
```

$$t(n) = 2 + 2n = O(n) \tag{2}$$

2.3. Ejercicio 2c

```
push_back tiene costo 0(1);

void f(vector<int> &v1, vector<int> &v2){   // n = |v1| m = |v2|
    vector<int> res();                      // 1
    for(int i = 0; i<v1.size(); i++){       // 2 (init, guard), n iteraciones
        res.push_back(v1[i]);               // 1
    }                                       // t(n) = 2 + 2n

    for(int i = 0; i<v2.size(); i++){       // 2, n iteraciones
        res.push_back(v2[i]);               // 1
    }                                       // t(m) = 2 + 2n;
    return res;
}
```

$$t(n, m) = 5 + 2n + 2m = O(n + m) \tag{3}$$

2.4. Ejercicio 3

Escribir un programa que sea correcto respecto de la especificacion y cuyo tiempo de ejecucion de peor caso pertenezca a O($|s|$) donde s es la secuencia pasada como parametro

`proc restarLosPares` (in s:$seq\langle\mathbb{Z}\rangle$, in x:$\mathbb{Z}$, out res:$seq\langle\mathbb{Z}\rangle$) {

`Pre` {true}
`Post` $\{|res| = |s| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L res[i] = x - \sum_{j=0}^{i} \text{if } esPar(s[j]) \text{ then } s[j] \text{ else } 0 \text{ fi})\}$
}

```
vector<int> restarLosPares(vector<int> s, int x){    // O(|s|)
    vector<int> res;                                 // 1
    int i = 0;                                       // 1
    int suma = 0;                                    // 1
    while(i < s.size()){                             // 1, |s| iteraciones
        if(s[i] %2 == 0){                            // 2
            suma += S[i];                            // 1
        }
        res.push_back(x - suma);                     // 2
        i++;
    }                                                // t(n) = 1 + 6n
    return res;
}
```

$$t(n) = 4 + 6n = O(n) -> n = |s| -> O(|s|) \tag{4}$$

2.5. Ejercicio 4

Una matriz cuadrada se dice triangular si todos los elementos por debajo de las diagonal son iguales a 0

1. Escribir un programa que calcule el determinante de una matriz triangular. Recordar que el determinante de una matriz triangular es el producto de los elementos de su diagonal
2. Escribir un programa que determine si una matriz de N + M es o no triangular
3. Calcular el tiempo de ejecucion de peor caso de los programas
   a) en funcion de la cantidad de filas de la matriz
   b) en funcion de la cantidad de elementos de la matriz

```
//1. detTriangular
// Pre: M es matriz cuadrada y es triangular
// Post res = M[0][0] * M[1][1] * .... * M[n-1][n-1]
int detTriangular(vector<vector<int>> M){
                            // 3.1 n = |M|          //3.2 n = |M|*|M|
    int i = 0;              // 1                    // 1
    int res = 1;            // 1                    // 1
    while(i < M.size()){    // 1, n iteraciones     // 1, sqrt(n) iteraciones
        res = res*M[i][i]   // 2                    // 2
        i++;                //
    }                       // t(n) = 1 + 3n        // t(n) = 1 + 3*sqrt(n)
    return res;             // t(n) = 3 + 3n        // t(n) = 3 + 3*sqrt(n)
}                           // O(n)                 // O(sqrt(n))
// 2. esTriangular
```

$$Pre : (\forall i : \mathbb{Z})(0 \leq i < |M| \longrightarrow_L |M[i]| = |M[0]|) \tag{5}$$

$$Post : res = true \leftrightarrow (\forall i, j : \mathbb{Z})(0 \leq i < |M| \wedge 0 \leq j < i \longrightarrow_L M[i][j] = 0) \tag{6}$$

```
bool esTriangular(vector<vector<int>> M){
                                        // 3.1 n = |M|          //3.2 n = |M|*|M|
    bool res = true;                    // 1                    // 1
    for(int i = 0; i<M.size(); i++){    // 2, n iteraciones     // 2, sqrt(n) iteraciones
        for(int j = 0; j<i; j++){       // 2, i iteraciones     //
            res = res && M[i][j] == 0;  // 3                    //
        }                               // t(i) = 2 + 5i        //
        // 3.1 t(n) = 2+sum(i=0,n-1)(2+5i)
    }
        // 3.1 t(n) = 4+(2n-2)+5*sum(i=1,n-1)(i)
        // 3.1 t(n) = 4+(2n-2)+5*(n*(n-1) / 2)
        // 3.1 t(n) = 5 + (2n-2)+5*(n*(n-1) / 2)
        // 3.1 t(n) = 3+2n+5(n^2-n)/2
```

```
        // 3.1 O(1) + O(n) + O(n^2) = O(n^2)

        // 3.2 Donde dice n tenemos raiz cuadrada de n, entonces
        // lo unico que necesitamos hacer es llegar a que en la cuenta final
        // analizamos reemplazando n^2 por sqrt(n) entonces tenemos que
        // 3.2 es O(sqrt^2) -> O(n)
    return res;
}
```

2.6. Ejercicio 5

proc `multiplicar` (in m1:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$, in m2:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$, out res:$seq\langle seq\langle \mathbb{Z}\rangle\rangle$) {
`Pre` {...}
`Post` $\{|res| = |m1| \wedge_L (\forall i : \mathbb{Z})(0 \leq i < |m1| \longrightarrow_L (|res[i]| = |m2[0]| \wedge_L$
$(\forall j : \mathbb{Z})(0 \leq j < |m2[0]| \longrightarrow_L res[i][j] = \sum_{k=0}^{|m2|-1} m1[i][k] * m2[k][j]))))\}$
}

1. Completar la precondicion del problema
2. Escribir un programa que retorne AB
3. Determinar el tiempo de ejecucion de peor caso de este programa en fuuncion de
   a) la cantidad de filas y columnas de cada matriz
   b) Suponiendo que N = filas(m1) = filas(m2) = columnas(m1) = columnas(m2)

```
// 1 . Pre = {esMatriz(m1) && esMatriz(m2) &&L |m1[0]| = |m2|}
vector<vector<int>> multiplicar(vector<vector<int>> m1, vector<vector<int>> m2){
                        // n = |m1|, m = |m1[0]| = |m2|, r = |m2|
    vector<vector<int>> res;                            // 1
    for(int i = 0; i < m1.size(); i++){                 // 2, n iteraciones
        // fila i de res
        temp = vector<int>();                           // 1
        for(int j = 0; j<m2[0].size(); j++){            // 2, m iteraciones
            // columna j de res
            int v = 0;                                  // 1
            for(int k = 0; k < m2.size(); k++)          // 2, r iteraciones
                v += m1[i][k] * m2[k][j];               // 2
            temp.push_back(v);                          // 1
        }
        res.push_back(temp);                            // 1
    }
    return res;
}

// t(r) = 2 + 3r
// t(m, r) = 4 + m*(2+3r)
// t(n, m, r) = 2 + n*(4+m*(2+3r))
// t(n, m, r) = 3 + n*(4+m*(2+3r)) -> 3 + n*(4+2m+3mr) = 3 + 4n + 2mn + 3mnr
// El termino que domina es mnr -> O(mnr)

// Si asumo que m = n = r entonces tengo O(n*n*n) = O(n^3)
```