Apuntes Final Algo1

Teorema del Invariante y Corrección del Ciclo

Sean:
1. I: Invariante
2. Pc: Precondicion
3. Qc: Postcondicion
4. B: Guarda
5. S: Cuerpo del ciclo
6. fv: Función variante

Teorema del Invariante
Si se cumple lo siguiente:
1. $Pc => I$
2. $\{I \land B\}S\{I\}$
3. $I \land \neg B => Qc$

y **el ciclo termina** entonces la siguiente tripla de Hoare es valida:
$\{Pc\}$while B do S endwhile$\{Qc\}$

Teorema de corrección de un ciclo
Si además de los trés predicados anteriores, se cumple lo suiguiente:
1. $\{I \land B \land fv = v0\}S\{fv < v0\}$
2. $I \land fv \leq 0 => \neg B$

Entonces la siguiente tripla de Hoare es valida:
$\{Pc\}$while B do S endwhile$\{Qc\}$
Es decir, también nos garantiza que el ciclo termina.

Testing

Algoritmos de Ordenamiento

Insertion Sort
Idea: Recorrer la lista e ir manteniendo la parte izquierda siempre ordenada, luego por cada elemento siguiente irlo insertando segun corresponda en la parte ya ordenada.
Complejidad: $O(n^2)$ ya que tenemos dos loops anidados.

```
insertar(lista, i) {
  while (i > 0 && lista[i] < lista[i-1]) {
    swap(lista, i, i-1)
  }
}


insertionSort(lista) {
```

```
  for (i = 0; i < |lista|; i++) {
    insertar(lista, i);
  }
  return lista;
}
```

Selection Sort
Idea: Ir seleccionando el minimo de la lista e irlo insertando al principio hasta llegar al final.
Complejidad: $O(n^2)$ ya que minimo realizamos n pasadas y en cada una hacemos n operaciones.

```
emplazarMinimo(lista, desde) {
  posMinimo = desde;
  for (i = desde; i < |lista|; i++) {
    if (lista[i] < lista[posMinimo]) {
      posMinimo = i;
    }
  }
  swap(lista, desde, posMinimo);
}

selectionSort(lista) {
  for (i = 0; i < |lista|; i++) {
    emplazarMinimo(lista, i);
  }
  return lista;
}
```

Bubble Sort
Idea: Comparamos cada elemento de la lista con el siguiente, intercambiandolos si tienen el orden inverso.
Necesitamos revisar la lista hasta que no se necesiten mas intercambios.
Complejidad: $O(n^2)$ ya que burbujeamos n veces y por cada burbujeo realizamos n operaciones.

```
burbujeo(lista, desde) {
  for (i = |lista|-1; i > desde; i--) {
    if (lista[i] < lista[i-1]) {
      swap(lista, i, i-1)
    }
  }
}

bubbleSort(lista) {
  for (i = 0; i < |lista|; i++) {
    burbujeo(lista, i);
  }
```

```
}
```

Counting Sort
Idea: Dado una cantidad finita de valores a ordenar podemos ordenar en $O(n)$ contando cuantos tenemos de cada uno en una pasada y luego rellenando un nuevo array con los elementos correspondientes en el orden que corresponda.

Resolucion de Finales

Final 1

1. **¿Cual es la diferencia entre especificación e implementación?**
   La especificación es una manera de describir el problema a resolver mediante un lenguaje formal; nos informa sobre las condiciones que se tienen que cumplir para resolver el problema y cómo debe verse la solución, pero no nos dice COMO se resuelve el problema; eso depende de la implementación, la cual, respetando la especificación, nos dice paso a paso como resolver el problema.
2. **¿Que es un tipo de dato?**
   A la hora de ejecutar nuestro programa, en la memoria, todos los datos son finalmente guardados como un valor en bytes; si bien podríamos operar de esta manera, nos da muy poca información sobre que es cada dato.
   Si una variable tiene un cierto tipo de dato, esto nos da un contexto sobre que va a representar la variable, y nos acota la cantidad de posibles valores que pueda tomar (por ejemplo, no le daríamos el valor de “casa” a una variable de tipo numerico); además podemos definir operaciones en base a estos tipos con el fin de que nuestro programa sea mas correcto, y no poder operar en tipos incompatibles, como “casa” + 5 por ejemplo.
3. **¿Que significa que {P}S{Q} sea una Tripla de Hoare válida?**
   Quiere decir que dados los predicados P (Precondición) y Q (Postcondicion) y un programa S: Siempre que iniciemos el programa S cumpliendose P, al terminar su ejecución, en el estado final se cumple Q.
4. **¿Que diferencia hay entre un algoritmo y un programa?**
   Un programa es código que podemos compilar/interpretar/ensamblar y ejecutar en una PC, y que va a ejecutar y dar un resultado tangible; un Algoritmo es una descripción paso a paso de como vamos a resolver el problema en cuestión, pero no viene dado en un lenguaje de programación y por lo tanto no podremos ejecutarlo.
   Dado un algoritmo podemos programarlos de diversas maneras y con distintos lenguajes de programación, efectivamente obteniendo programas distintos pero en base al mismo algoritmo que diseñamos.
5. **Explicar el criterio de adecuación de cubrimiento de arcos y el cubrimiento de branches/decisiones. ¿Alguno incluye a el otro?**
   ???
6. **¿Que es un invariante de ciclo?**
   El invariante de ciclo es un predicado que refleja una condición que se cumple al inicio de un ciclo y se mantiene igual (invariante) al final de este, si bien es posible que no se cumpla a la mitad de la ejecución del mismo.
   Este es de vital importancia para ser utilizado en el Teorema del Invariante, en el cual mediante el Invariante que hayamos propuesto, y otros predicados y operaciones lógicas podemos llegar a demostrar que nuestro ciclo termina, y/o si es correcto.
7. **Enunciar las hipotesis del Teorema de corrección de un Ciclo**
   Dada la precondición P_C, la postcondición Q_C, la guarda B y el invariante de ciclo I, el teorema de corrección de ciclo indica que si se cumple:
   i. $P_C => I$
   ii. $\{I \land B\} S \{I\}$
   iii. $I \land \neg B => Q_C$

   y además el ciclo termina, entonces nuestro ciclo es correcto, es decir, vale:
   $\{P_C\}$while B do S endwhile$\{Q_C\}$
8. **¿Que diferencia hay entre subespecificar y sobreespecificar?**

Subespecificación: Da una precondición mas restrictiva o una postcondicion mas debil que la que se podría dar. Aumenta el numero de posibles soluciones al problema.
Sobreespecificación: Da una postcondición mas restrictiva que la necesaria o una precondición mas laxa. Reduce el numero de posibles soluciones al problema.

9. **¿Que algoritmos de búsqueda en arreglos conoce? ¿Que complejidad computacional tienen estos algoritmos?**
(Ver sección de Sorting)
10. **¿Por que el teorema del invariante solicita que la función variante sea estrictamente decreciente? ¿Que ocurre si no lo es?**
La función variante debe ser estrictamente decreciente ya que esta codifica el valor que garantiza la terminación del ciclo al llegar a cero; si esta no fuese estrictamente decreciente, podríamos ciclar infinitamente sin que nunca llegue al valor de corte y por lo tanto el ciclo no terminaría.
11. **¿Que es un tipo compuesto (en especificación? ¿Como se implementaría en C++?**
Un tipo compuesto es aquel que podemos definir nosotros en base a varios tipos ya existentes, y que modela una entidad mas compleja. Por ejemplo podemos definir un tipo compuesto "Jugador" que tenga un campo numérico de "puntos" y una cadena de texto "nombre", que en C++ sería implementado mediante la directiva Struct de la siguiente manera:

```
struct Jugador {
  int    puntos;
  string nombre;
};
```

12. **Sean f y g especificaciones. ¿Podemos preferir alguna de las dos? ¿Por que?**

```
problema f(x: T1) = res T2 {
    requiere P^R
    asegura Q
}

problema g(x: T1) = res T2 {
    requiere P
    asegura Q
}
```

En principio no puedo preferir ninguna de las dos porque no tengo ningún otro detalle sobre las implementaciones, no tengo idea si quizá f es mejor en complejidad que g o viceversa, etc. Sin embargo, si solamente me importase cual cubre mas casos, en ese caso elegiría la g ya que al tener una precondicion mas laxa me garantiza que voy a encontrar soluciones para un conjunto mas amplio de input, y por lo tanto el programa va a ser mas versátil.

13. **Escribir un programa en pseudocodigo que devuelva dos listas ordenadas de distinto tamaño y devuelva una tercera que resulte de la unión de ambas y no contenga números repetidos.**

```
def merge(la, lb):
    lm = []
    la_i, lb_i = 0, 0
    while la_i < len(la) and lb_i < len(lb):
        if len(lm) > 0:
            prev = lm[len(lm)-1]
        if la[la_i] <= lb[lb_i]:
```

```
            if len(lm) < 1 or prev != la[la_i]:
                lm.append(la[la_i])
            la_i += 1
        else:
            if len(lm) < 1 or prev != lb[lb_i]:
                lm.append(lb[lb_i])
            lb_i += 1

    while la_i < len(la):
        lm.append(la[la_i])
        la_i += 1
    while lb_i < len(lb):
        lm.append(lb[lb_i])
        lb_i += 1
    return lm
```

14. **Dado un arreglo de enteros ordenado, encontrar el numero primo que se repite la menor cantidad de veces. Especificar, dar un algoritmo O(n) para solucionarlo y dar el invariante o función variante de los ciclos del programa.**
    Despues
1.
2. **Especificar el problema de ordenar una secuencia de enteros, de mayor a menor, segun la cantidad de divisores primos de cada numero.**
    proc ordenarPorDiv(in s : $seq < Z >$){
    $Pre = \{s = s_0\}$
    $Post = \{(\forall i : N)(0 \leq i < |s| - 1) \rightarrow_L (\#divPrimos(s[i]) \geq \#divPrimos(s[i+1])) \wedge mismos(s, s_0)\}$
    aux #divPrimos($n : Z$) : N = $\sum_{i=1}^{n}$ if n mod i = 0 ∧ esPrimo(i) then 1 else 0 fi;
    pred esPrimo($n : Z$){$True \leftrightarrow (\forall d : N)(1 < d < n \rightarrow_L$ n mod d ≠ 0)
    pred mismos($s, t : seq < Z >$)$(\forall e : Z)\#apariciones(s, e) = \#apariciones(t, e)$
    aux #apariciones($s : seq, e : Z$) : Z = $\sum_{i=0}^{\lceil s \rceil - 1}$ if s[i] = e then 1 else 0 fi;
    }