UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Segundo parcial – 20/11/2015

Nro. de orden: 72
LU: 79/15
Apellidos: Borgna
Nombres: Agostín Pablo

1	2	3	4	TOTAL
20	35	15	20	90

A

Aclaraciones: El parcial NO es a libro abierto. Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos. **Entregar cada ejercicio en hoja separada.**

Ejercicio 1. [25 puntos]

Dado el siguiente programa, y asumiendo que se cuenta con el siguiente `requiere`: $|a| == n \land n > 0$

```
void sumarPiramidal(int[] a. int n) {
        int i = 0;
        int j = n - 1;
        while (i < j) {
                a[i] += i;
                a[j] += i;
                i++;
                j--;
        }
}
```

a) Dar la especificación del ciclo (P_c, Q_c, I, f_v y cota)

b) Demostrar que $(I \land f_v \leq c) \Rightarrow \neg B$ y que la función variante f_v es estrictamente decreciente.

Ejercicio 2. [35 puntos]

Dado el siguiente código, que contiene un único ciclo cuya precondición es $P_c : i == 0 \land a == pre(a)$ y su postcondición es $Q_c : |a| == |pre(a)| \land (\forall j \leftarrow [0..|a|))a_j == \sum[pre(a)_k \,|\, k \leftarrow [0..j]]$. Asumir que se cuenta con el siguiente `requiere`: $|a| == n \land n > 0$

```
void sumasParciales(int[] a. int n) {
        int i = 0;
        while (i < n-1) {
                i++;
                a[i] += a[i-1];
        }
}
```

a) Determinar si el siguiente invariante es correcto para el ciclo dado
$I : 0 \leq i < n \land |a| == |pre(a)| \land (\forall j \leftarrow [0..i])a_j == \sum[pre(a)_k \,|\, k \leftarrow [0..j]]$.
Si no lo es, presentar una versión corregida de dicho invariante (y usar esa versión corregida para las siguientes demostraciones).

b) Demostrar $P_c \Rightarrow I$

c) Demostrar $(I \land \neg B) \Rightarrow Q_c$

d) Demostrar que el cuerpo del ciclo preserva el invariante

Ejercicio 3. [15 puntos] Implementar un programa en *C++* que dado un vector de tuplas en donde cada posición contiene el nombre de una persona y su puntaje en una determinada competencia, devuelva los k primeros puestos. En particular, se debe cumplir con la siguiente especificación y signatura(vector<String> kPrimeros(vector<pair<String, **int**> > a, **int** k)):

```
problema kPrimeros (a: [(String,ℤ)], k: ℤ) = res : [String] {
        requiere 1 ≤ k ∧ k ≤ |a|;
        requiere (∀i, j ← [0..|a|), i ≠ j)fst(a[i]) ≠ fst(a[j]);
        asegura |res| == k;
        asegura (∀i ← res)(∃j ← a)i == fst(j);
        asegura (∀i, j ← [0..|res|), i ≠ j)res[i] ≠ res[j];
        asegura (∀i ← [0..k − 1))puntaje(res[i], a) ≥ puntaje(res[i + 1], a);
        asegura (∀i ← a, fst(a) ∉ res)puntaje(fst(i), a) ≤ puntaje(res[k − 1], a);
        aux puntaje (s:String, a: [(String,ℤ)]) : ℤ = cab([snd(x)|x ← a, s == fst(x)]);
}
```

Ejercicio 4. [25 puntos]

Dado dos arreglos a y b ordenados de números enteros mayores o iguales a 0 y un entero k. Se quiere decidir si existe un número en cada arreglo tal que sumados dan k. Formalmente, se quiere resolver el problema que cumple la siguiente especificación.

```
problema dosQueSumanK (a: [ℤ], b: [ℤ], n: ℤ, k: ℤ) = res : Bool {
        requiere |a| == n;
        requiere |b| == n;
        requiere losValoresEstanAcotados : (∀i ← [0..n))0 ≤ a_i ∧ a_i ≤ 13 ∗ n;
        requiere losValoresEstanAcotados2 : (∀i ← [0..n))0 ≤ b_i ∧ b_i ≤ 13 ∗ n;
        requiere arregloOrdenado : (∀i ← [0..n − 1))a_i ≤ a_{i+1};
        requiere arregloOrdenado2 : (∀i ← [0..n − 1))b_i ≤ b_{i+1};
        asegura res == (∃i, j ← [0..n))a[i] + b[j] == k;
}
```

Se proponen los 3 siguientes programas en lenguaje *C++*. Cada código tiene comentado los pasos utilizados para resolverlo.

```
bool dosQueSumanK1(int a[], int b[], int n, int k){
  int i = 0;
  int j = 0;
  bool res = false;
  while(i<n&&!res){
    j = 0;
    while(j<n&&!res){
      if(a[i]+b[j]==k){
        res = true;
      }
      j++;
    }
    i++;
  }
  return res;
}

bool dosQueSumanK2(int a[], int b[], int n, int k){
  int i = 0;
  int j = 0;
  bool res = false;
  while(i<n){//Solo se busca si con el mas chico del arreglo b suman algo acotado por arriba por k
    if(a[i]+b[0]<=k){
    //Solo se busca si con el mas grande del arreglo b suman algo acotado por abajo por k
      if(a[i]+b[n-1]>=k){
        //En este caso, hay que buscar en el segundo arreglo si alguno suma k con a[i].
        //El siguiente while es una busqueda binaria entre la primera y la ultima
        //posicion del arreglo que realiza correctamente el computo deseado.
        while(...){
          ...
        }
      }
    }
    i++;
  }
  return res;
}

bool dosQueSumanK3(int a[], int b[], int n, int k){
  int i = n-1;
  int j = 0;
  bool res = false;
  while(i>=0&&j<n){
    if(a[i]+b[j]==k){ //Si suman k, ya encontramos la solucion
      res = true; break;
    }else{   //Si suman menos, entonces solo tiene sentido tratar de hacer crecer la suma
      if(a[i]+b[j]<k){
        j++;
      }else{   //Si suman mas, solo tiene sentido tratar de hacer decrecer la suma
        i--;
      }
    }
  }
  return res;
}
```

a) Indicar el orden de complejidad temporal en peor caso de los 3 algoritmos presentados y explicar como se llega a dichos órdenes.

b) Si se elimina la restricción de que el arreglo esté ordenado, hay que modificar los algoritmos para que funcionen. Si se cuenta con la función *sort*[1] que sirve para ordenar un arreglo de n enteros con un algoritmo que tiene un orden de complejidad perteneciente a $\mathcal{O}(nlog(n))$. ¿Cómo se modifican los 3 algoritmos propuestos para que sigan funcionando?¿Que complejidad temporal pasan a tener cada uno de los algoritmos?

c) Si además de eliminar la restricción de que el arreglo esté ordenado, se tiene en cuenta el requiere del problema que indica que los números del arreglo se encuentran acotados entre 0 y la longitud del arreglo. ¿Podría usarse esta restricción para modificar la complejidad de las soluciones propuestas en el inciso anterior? En caso afirmativo, indique como cambiaría el orden de complejidad. En caso negativo, explique brevemente el porqué.

[1]La función sort para un arreglo a de tamaño n se utiliza escribiendo sort(a,$a+n$)