**UNIVERSIDADE DE SANTIAGO DE COMPOSTELA**

**E.T.S.E**

**PROGRAMACIÓN II**

**Examen**

**Profesor: David E. Losada** **Julio 2011**

Fecha: 4/7/2011.
Nombre y apellidos:_____

PARTE I (SIN MATERIAL)

1. Test (2 puntos. Cada respuesta correcta suma 0.2 puntos. Cada respuesta incorrecta resta 0.1 puntos)

1. La ventaja fundamental de un Tipo Abstracto de Dato (TAD) reside
   a. en unir sintaxis, semántica e implementación en un mismo módulo o fichero
   b. que la misma implementación pueda corresponderse con distintas especificaciones del TAD
   c. que los programas se abstraen de la implementación del TAD y si la implementación cambia no es necesario cambiar los programas que usan el TAD
   d. ninguna de las anteriores

2. Un TAD pila
   a. es tan flexible como un TAD lista (permite las operaciones de una lista) pero, además, incorpora funcionalidad para meter y sacar elementos por la cima de la pila
   b. puede implementarse internamente mediante un TAD lista, restringiendo el uso de la lista a las operaciones que soporta la pila
   c. permite acceso a cualquier elemento de la pila, independientemente de su posición
   d. ninguna de las anteriores

3. La búsqueda binaria
   a. es más efectiva que la búsqueda lineal con centinela
   b. es menos efectiva que la búsqueda lineal sin centinela
   c. tiene orden de complejidad equivalente a la búsqueda lineal con centinela
   d. ninguna de las anteriores

4. Para un mismo problema disponemos de un algoritmo de orden $n \log n$, otro de orden $n!$, otro de orden $2^n$ y otro de orden $n^{10}$. ¿Cuál elegimos?
   a. el polinómico ($n^{10}$)
   b. el de orden $n^n$
   c. el de orden $n!$
   d. ninguna de las anteriores

5. El orden exacto de complejidad ($\Theta$) nos da
   a. exclusivamente una cota inferior a la complejidad del problema
   b. exclusivamente una cota superior a la complejidad del problema
   c. una cota superior y una cota inferior a la complejidad del problema
   d. ninguna de las anteriores

6. ¿por qué se suele realizar el análisis de algoritmos centrándonos en el comportamiento en el peor de los casos?
   a. el análisis de algoritmos no se realiza de ese modo
   b. porque así tenemos acotado el tiempo máximo que tardará en algoritmo cualquiera que sean los datos de entrada
   c. porque el comportamiento en el peor y en el mejor de los casos suele ser equivalente
   d. ninguna de las anteriores

7. El algoritmo quicksort de ordenación
   a. para tallas pequeñas puede ser menos eficiente temporalmente que el algoritmo de la burbuja
   b. es un ejemplo de algoritmo sublineal
   c. es un ejemplo de algoritmo cuadrático
   d. ninguna de las anteriores

8. Los algoritmos de fuerza bruta
   a. suelen implicar bajo coste computacional
   b. suelen encontrar siempre la solución al problema
   c. realizan una poda del espacio de búsqueda de soluciones
   d. ninguna de las anteriores

9. Un algoritmo voraz
   a. es un caso especial de divide y vencerás
   b. suele ser más eficiente temporalmente que uno de fuerza bruta
   c. es un caso especial de programación dinámica
   d. ninguna de las anteriores

10. La programación dinámica
    a. suele consistir en almacenar datos tabularmente para reutilizar resultados en los cómputos
    b. consiste en sacrificar tiempo para conseguir mejorar el uso de memoria
    c. consiste en decidir dinámicamente (en tiempo de ejecución) entre ahorrar memoria o ahorrar tiempo
    d. ninguna de las anteriores

2. (1 pto) Explica brevemente la diferencia entre complejidad espacial y complejidad temporal

3. (1 pto) Explica ventajas y desventajas de utilizar un bloque de memoria contigua para almacenar una lista de elementos. ¿que operaciones sobre el TAD lista son más problemáticas para un bloque contiguo? ¿qué operaciones son más sencillas? ¿por qué?

**UNIVERSIDADE DE SANTIAGO DE COMPOSTELA** **E.T.S.E**

**PROGRAMACIÓN II**

**Examen**

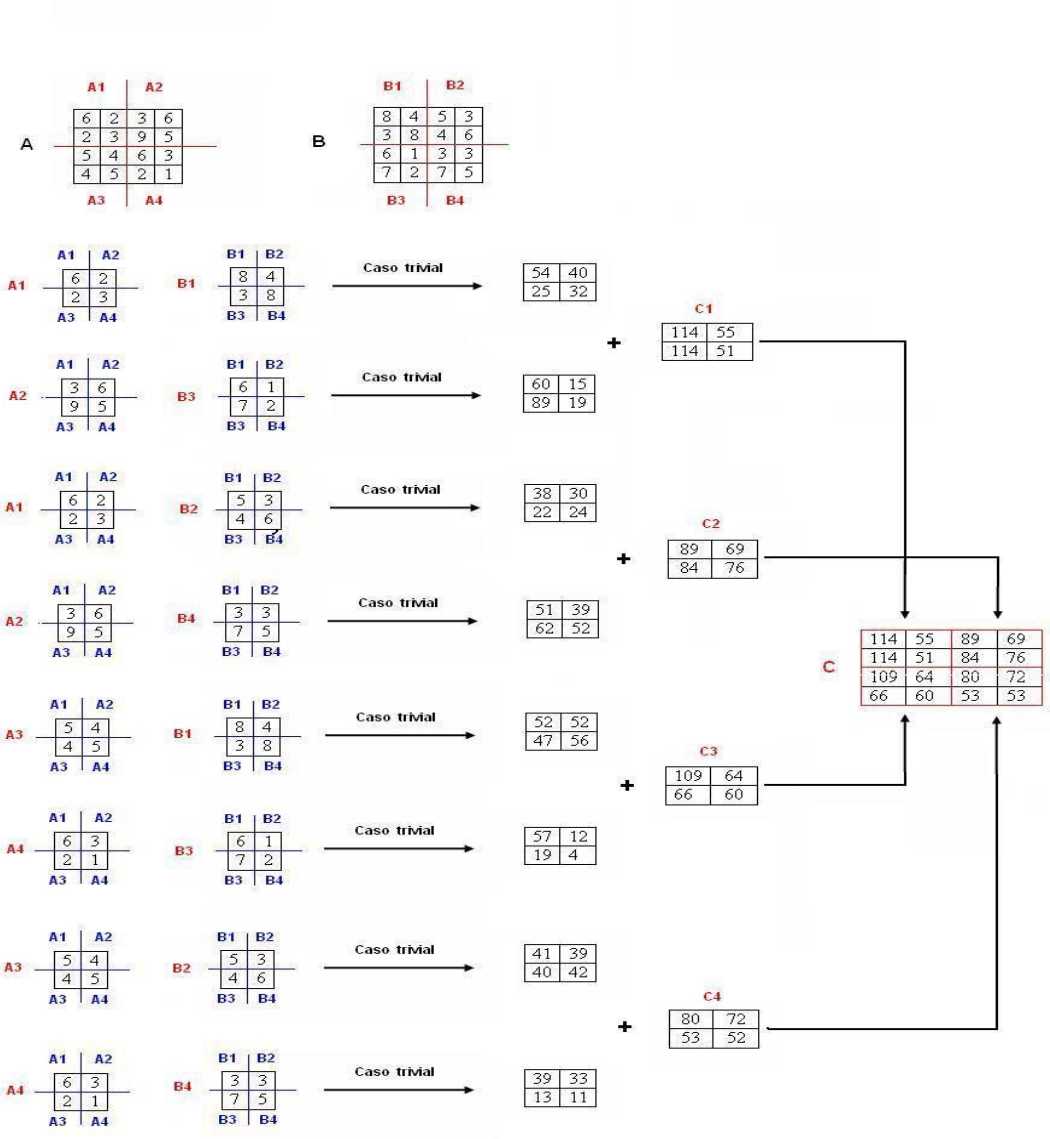
**Profesor: David E. Losada** **Julio 2011**

Fecha: 4/7/2011.

Nombre y apellidos:_____

PARTE II (CON MATERIAL)

4. (1 punto) La múltiplicación de matrices cuadradas, en su implementación directa iterativa (que no es divide y vencerás), ¿qué orden de complejidad tiene respecto al tamaño de la matriz en número de filas/columnas? ¿por qué?
   En cambio, la multiplicación divide y vencerás realizada en las prácticas, basada en el esquema de división que figura a continuación, ¿qué orden de complejidad tiene? ¿por qué?



5. (1.5 puntos) Dado los siguientes programas, determina el orden superior (O) e inferior (Ω) de complejidad de los mismos siendo la talla del problema el número N

Programa A:

```
....

int main()
{
....

if (N%55 == 0)
        k = f1(N);
else
        k=f2(N);


}
```

```
float f1(int num)
{
int aux, aux2, aux3;
float rdo=0;

for (aux=0;aux<10000;aux++)
  for (aux2=0;aux2<num;aux2++)
    for (aux3=0;aux3<num;aux3++)
       rdo += aux*aux2*aux3;

return rdo;
}
```

```
float f2(int num)
{
int aux,aux2;
float rdo=0;

for (aux=0;aux<num;aux++)
   for (aux2=0;aux2<num;aux2++)
      rdo += aux+aux2;

return rdo;
}
```

Programa B:

```
....

int main()
{
....

k = f1(N);

}
```

```
float f1(int num)
{
float rdo;

rdo= 2*f1(num-1)*f1(num-2);

return(rdo);
}
```

Programa C:

```
....

int main()
{
....

k = f2(N);

if (k>1000)
    k+=f1(N);

}
```

```
float f1(int num)
{
int aux, aux2;
float rdo=0;

for (aux=0;aux<num;aux++)
  for (aux2=0;aux2<num;aux2++)
       rdo += aux*aux2;

return (rdo);
}
```

```
float f2(int num)
{
int aux;
float rdo=0;

for (aux=0;aux<num;aux++)
  rdo += aux;

return rdo;
}
```

6. (1.5 punto) Partiendo de la especificación formal del TAD Natural vista en clase (ver a continuación) completarla meditante la incorporación de las operaciones producto y potencia.

**Especificación formal:**
**Tipo: Natural**
**Sintaxis:**

***Cero → Natural**
***Sucesor(Natural)→ Natural**
**EsCero(Natural) → Boolean**
**Igual(Natural,Natural)→ Boolean**
**Suma(Natural, Natural) → Natural**
**Antecesor(Natural)→Natural**
**Diferencia(Natural,Natural)→Natural**

**Semántica: ∀m,n∈Natural**

**EsCero(Cero) ⇒ True**
**EsCero(Sucesor(n))⇒ False**
**Igual(Cero,n) ⇒ EsCero(n)**
**Igual(Sucesor(n),Cero) ⇒ False**
**Igual(Sucesor(n),Sucesor(m))⇒ Igual(m,n)**
**Suma(Cero,n) ⇒ n**
**Suma(Sucesor(m),n)⇒ Sucesor(Suma(m,n))**
**Antecesor(Cero)⇒ error**
**Antecesor(Sucesor(n)) ⇒ n**
**Diferencia(n,0)⇒ n**
**Diferencia(Cero,Sucesor(n))⇒error**
**Diferencia(Sucesor(m),Sucesor(n))⇒**
**Diferencia(m,n)**

7. (2 puntos) Un Sistema Operativo mantiene la lista de procesos a la espera de CPU como una cola con tres niveles de prioridad (1, 2, 3; siendo 3 la máxima prioridad). Internamente, guarda todos los procesos en una lista estándar implementada como una estructura con simple enlace donde el tipoelemento es una estructura con el identificador del proceso y su prioridad:

| | | | | |
|---|---|---|---|---|
| proceso | 1234 | 345 | 1245 | .... |
| prioridad | 2 | 1 | 3 | .... |

```
typedef struct{
        long proceso;
        int prioridad;
        } TELEMENTO;

typedef struct celda {TELEMENTO elemento;
          struct celda * sig;} TCELDA;

typedef TCELDA * POSICION;
typedef struct lista {POSICION inicio;
           int longitud;
           POSICION fin;} * TLISTA;

#tydepef  TLISTA  TCOLAPRIO
```

Escribe una función que dada una de estas colas, calcule el primer proceso en la cola por prioridad (esto es, el primer elemento en la lista que tiene más prioridad, en el ejemplo sería el 1245). La función tendría la siguiente cabecera:

Sólo es necesario utilizar las operaciones del TAD lista (no hay que acudir para nada a detalles de implementación del TAD lista)

```
long primeroColaPrio(TCOLAPRIO   procesosencolados  )
{
// rellena las instrucciones necesarias

.....

}
```