EXAMEN DE MAYO DE 2010 DE PROGRAMACIÓN II

1. **Test (3.2 puntos. Cada respuesta correcta suma 0.2 puntos. Cada respuesta incorrecta resta 0.1 puntos)**

1. Un Tipo Abstracto de Dato (TAD) permite

a. definir la sintaxis, semántica e implementación de un nuevo de tipo de datos

b. hacer explícitos los detalles de programación asociados a un nuevo de tipo de datos

c. abstraerse de las operaciones que se harán sobre el tipo (esto es, no se concretan)

d. ninguna de las anteriores

2. Un Tipo Abstracto de Dato (TAD) se gestiona de manera opaca para

a. ocultar los detalles de implementación de las operaciones

b. ocultar la interfaz del tipo

c. ocultar la interfaz y detalles de implementación del tipo

d. ninguna de las anteriores

3. Un TAD pila se implementa de dos maneras alternativas (A y B). La alternativa A es una lista enlazada de elementos con un puntero a la cima de la pila. La alternativa B es mediante un bloque de memoria contigua. ¿cual de las siguientes afirmaciones es cierta?

a. un problema con la alternativa A es que tengo que decidir cuantos elementos reservo a priori en la lista enlazada

b. un problema con la alternativa B es que necesito un puntero adicional para cada valor almacenado en la pila

c. un problema con la alternativa B es para decidir cuantos elementos reservo memoria en el bloque

d. ninguna de las anteriores

4. Un TAD lista se implementa de dos maneras alternativas (A y B). La alternativa A es una lista enlazada de elementos con un puntero al primer elemento de la lista. La alternativa B es mediante una lista doblemente enlazada con un puntero al primer elemento de la lista y doble enlace entre cada elemento de la lista (hacia delante y hacia atrás). ¿cual de las siguientes afirmaciones es cierta?

a. la alternativa A me facilita la gestión de los accesos, tanto para inserciones como para eliminaciones de elementos de la lista

b. la alternativa A es más eficaz pues los elementos están dispuestos de manera contigua en memoria

c. un problema con la alternativa B es que utiliza más memoria que la A.

d. ninguna de las anteriores

5. La complejidad espacial es una estimación de

a. el tiempo que tardará en ejecutarse un programa

b. la memoria que utilizará el programa cuando se ejecuta

c. el tiempo y memoria que utilizará un programa al ejecutarse

d. ninguna de las anteriores

6. El análisis de algoritmos se suele realizar de manera asintótica, esto quiere decir

a. que sólo importan los lazos del programa. Lo demás se descarta.

b. que nos fijamos en las funciones del programa que hagan cálculos matemáticos relacionados con límites.

c. que nos fijamos en el comportamiento del algoritmo para tallas grandes del problema

d. ninguna de las anteriores

7. Se dispone de un algoritmo logarítmico y un algoritmo lineal que ambos resuelven el mismo problema…

a. el logarítmico es siempre mejor

b. para tallas pequeñas el lineal pudiera ser mejor

c. para tallas grandes el lineal será mejor

d. ninguna de las anteriores

8. Un algoritmo toma 2N+100 pasos para resolver un problema y otro toma $\sqrt{N}$ + 27

a. el segundo algoritmo es sublineal

b. en general, para resolver distintos tipos de problemas con distintas tallas, escogeremos el primer algoritmo

c. el segundo algoritmo es superlineal

d. ninguna de las anteriores

9. El orden de complejidad nos da una idea de

a. el comportamiento del algoritmo en el mejor de los casos

b. el comportamiento del algoritmo en el caso promedio

c. el comportamiento del algoritmo en el caso de tallas pequeñas

d. ninguna de las anteriores

10. El algoritmo de búsqueda lineal es un ejemplo clásico de

a. estrategia divide y vencerás

b. estrategia fuerza bruta

c. encadenamiento hacia atrás

d. ninguna de las anteriores

11. El algoritmo de búsqueda binaria es un algoritmo

a. cuadrático

b. lineal

c. superlineal

d. ninguna de las anteriores

12. Una estrategia divide y vencerás

a. es siempre mejor que una estrategia iterativa (por ejemplo, búsqueda binaria es mejor que búsqueda lineal)

b. puede dar lugar a mayor coste que una estrategia iterativa (depende de los costes de descomposición, combinación, etc.)

c. se puede aplicar siempre (sea como sea el problema)

d. ninguna de las anteriores

13. Un algoritmo voraz

a. nunca reconsidera las decisiones que toma

b. puede reconsiderar las decisiones que toma en caso de que llegue a un camino sin salida

c. explora siempre todas las posibilidades antes de tomar cualquier decisión

d. ninguna de las anteriores

14. Un algoritmo voraz

a. está garantizado que si hay solución óptima llega a ella

b. nunca encuentra la solución óptima

c. suele ser bastante eficiente

d. ninguna de las anteriores

15. La programación dinámica

a. consigue mejor complejidad temporal a costa de sacrificar complejidad espacial

b. consigue mejor complejidad espacial a costa de sacrificar complejidad temporal

c. escoge dinámicamente entre optimizar la complejidad espacial o la temporal

d. ninguna de las anteriores

16. La verificación de programas

a. permite asegurar que no hay errores en el software

b. debe asegurar cubrir los menos casos posibles con el mayor número de ejemplos de prueba

c. puede permitir descubrir algunos errores del software, no necesariamente todos

d. ninguna de las anteriores

2. (1.5 puntos) Construye la especificación formal del TAD Complejo con las operaciones Crea, Conjugado, Suma, Producto, ParteReal y ParteImaginaria. El cuerpo de los números complejos responde a las siguientes definiciones:

a) Se define número complejo a todo par ordenado (a,b) donde a y b son números reales (se suele representar como a+bi)

b) Dado el número complejo (a,b), se dice que a es su parte real y b su parte imaginaria

c) Llamaremos conjugado del número complejo (a,b) al número complejo (a,-b)

d) Se define la suma de complejos como (a1,b1)+(a2,b2)=(a1+a2,b1+b2)

e) Se define el producto de complejos como (a1,b1)*(a2,b2) = = (a1*a2-b1*b2 , a1*b2+b1*a2)

ESPECIFICACIÓN FORMAL DEL TAD COMPLEJO

Tipo: Complejo

Sintaxis:

*Crea	(Real, Real)	→	Complejo
Conjugado	(Complejo)	→	Complejo
Suma	(Complejo, Complejo)	→	Complejo
Producto	(Complejo, Complejo)	→	Complejo
ParteReal	(Complejo)	→	Real
Parte Imaginaria	(Complejo)	→	Real

Semántica:

$\forall\ a_1, b_1, a_2, b_2, \in$ Real, $\forall\ c_1, c_2 \in$ Complejo/ $c_1 = (a_1, b_1) = a_1 + b_1 * i$ y $c_2 = (a_2, b_2) = a_2 + b_2 * i$

Conjugado$((a_1,b_1))$	⇒	$(a_1, -b_1)$
Suma$((a_1,b_1),(a_2,b_2))$	⇒	$(a_1 + a_2,\ b_1 + b_2)$
Producto$((a_1,b_1),(a_2,b_2))$	⇒	$(a_1 * a_2 - b_1 * b_2,\ a_1 * b_2 + b_1 * a_2)$
ParteReal$((a_1,b_1))$	⇒	a_1
ParteImaginaria$((a_1,b_1))$	⇒	b_1

3. (2 puntos) Dado un TAD Pila, que cuenta con las conocidas operaciones de PilaVacia, EsVacia, Cima, Push y Pop, y dada una Pila que contiene números enteros escribe una función VaciaHasta3 que lo que hace es eliminar elementos de la pila hasta que la cima tenga el valor 3 (este valor de 3 se dejaría en la pila) o la pila quede totalmente vacía.

// rellena los argumentos y su tipo que recibe la función

void VaciaHasta3 ( )

{

// rellena las instrucciones necesarias

……

}

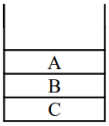
```
void VaciaHasta3 (TPILA *p) {
        TELEMENTO e;
        while (!EsVacia(*p)){
                Cima(*p,&e);
                if (e==3)
                        break;
                Pop(p);
        }
}
```

4. (1.8 puntos) Se dispone de una pila y se quiere extraer sus elementos y meterlos en una cola de modo que el elemento en la base de la pila sea el primero de la cola, y el elemento en la cima de la pila sea el último de la cola (y los elementos intermedios están en sus posiciones respectivas según indica la figura). La cola hay que crearla dentro de la función (no viene creada de fuera). Al finalizar la función la cola debe contener todos los elementos originalmente en la pila y la pila debe quedar vacía.

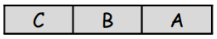
Realizar una función que utilizando las operaciones conocidas en los TADs Pila (PilaVacia, EsVacia, Cima, Push y Pop) y Cola (ColaVacia, EsColaVacia, primeroCola, añadirCola, eliminarCola) realice esta operación.

No se puede utilizar ningún otro TAD que no sea Pila/Cola. Sugerencia: Utilizar una pila auxiliar. El tipo de los elementos denominadlo TElemento.

PILA ORIGINAL:



COLA QUE SE QUIERE CONSTRUIR:




```
// rellena los argumentos y tipo que recibe la función
void crea_Cola_dada_una_Pila(                                    )
{
// rellena las instrucciones necesarias
……
}
```

```
void crea_Cola_dada_una_Pila(TPILA *p, TCOLA *c){
	TELEMENTO e;
	TPILA aux;
	ColaVacia(c);

	if(EsVacia(*p))
		return;
	PilaVacia(&aux);

	while(!EsVacia(*p)){
		Cima(*p, &e);
		Pop(p);
		Push(&aux,e);
	}

	while(!EsVacia(aux)){
		Cima(aux, &e);
		Pop(&aux);
		AñadirCola(c,e);
	}
}
```

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

Programa A:

```
....
int main()
{
....
k = f1(N)+f2(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;
}
```

Programa B:

```
....
int main()
{
....
k = f1(N);

k = k + f2(N);
}
```

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

if (num < 5)
   return (num*1.1)

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

return(rdo);
}
```

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

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

return rdo;
}
```

<u>Programa A:</u>

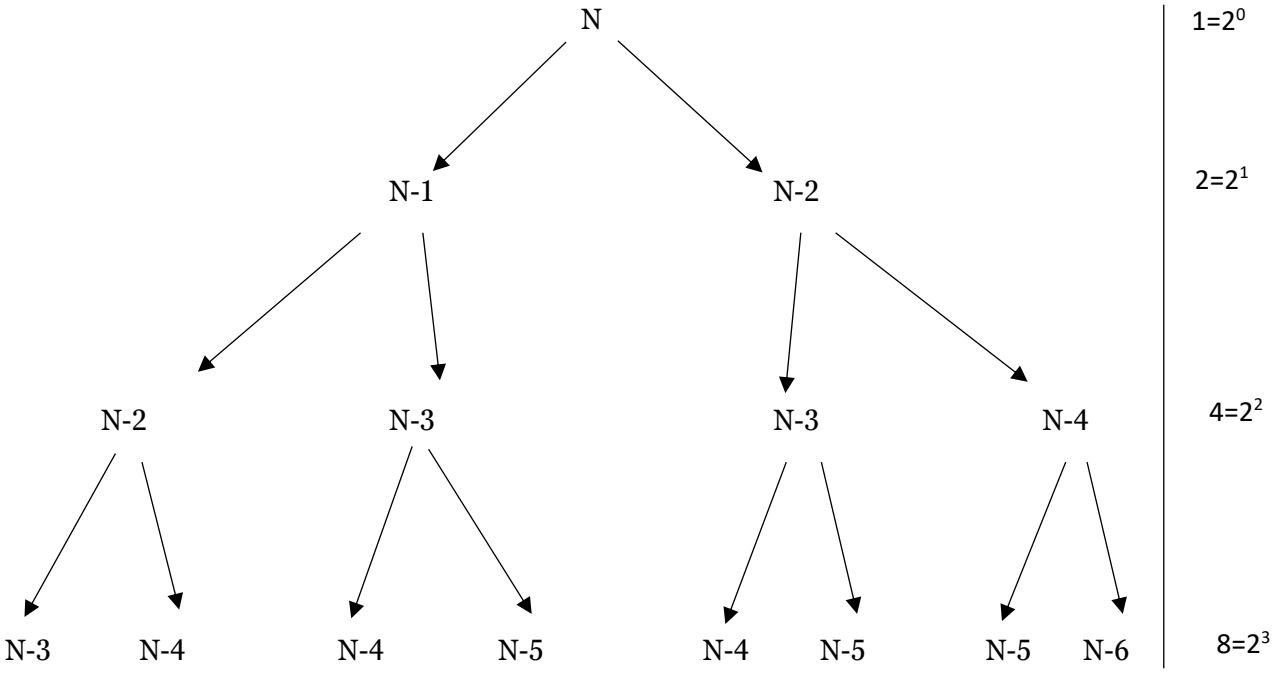
$máx(f_1, f_2) = máx(O(N^2), O(N)) = O(N^2)$ → Complejidad cuadrática

<u>Programa B:</u>

$máx(f_1,f_2)$

f_1?

f_1 es un algoritmo recursivo que divide un problema de tamaño N en 2 problemas de tamaño N-1 y N-2:



Peor caso: 2^N → $f_1 \in O(2^N)$

$máx(f_1, f_2) = máx(O(2^N),O(N)) = O(2^N)$ → Complejidad exponencial

EXAMEN DE JULIO DE 2010 DE PROGRAMACIÓN II

1. **Test (3 puntos. Cada respuesta correcta suma 0.2 puntos. Cada respuesta incorrecta resta 0.1 puntos)**

1. Un Tipo Abstracto de Dato (TAD) permite

   a. ocultar los detalles de implementación de un nuevo de tipo de datos

   b. hacer explícitos los detalles de programación asociados a un nuevo de tipo de datos

   c. abstraerse de las operaciones que se harán sobre el tipo (esto es, no se concretan)

   d. ninguna de las anteriores

2. En verificación de software una falla es

   a. un estado intermedio incorrecto en que entra un programa al ejecutarse

   b. un síntoma de que existe un error

   c. un caso de prueba

   d. ninguna de las anteriores

3. La verificación formal de programas está basada

   a. en probar los programas con casos de prueba completos

   b. en conseguir variedad de usuarios que prueben los programas

   c. en especificaciones matemáticas sobre lo que se espera de las operaciones del programa

   d. ninguna de las anteriores

4. La técnica de ramificación y poda…

   a. es una variante de la fuerza bruta

   b. realiza un recorrido ciego del espacio de búsqueda

   c. realiza un recorrido informado del espacio de búsqueda

   d. ninguna de las anteriores

5. Los algoritmos de vuelta atrás o backtracking...

a. son similares en filosofía a los algoritmos voraces (no reconsidera deciciones tomadas previamente)

b. realizan una búsqueda dentro de un espacio de búsqueda que se suele representar de forma arbórea

c. nunca realizan un recorrido en profundidad del espacio de búsqueda

d. ninguna de las anteriores

6. Un algoritmo voraz

a. divide el problema en subproblemas de modo recursivo

b. suele ser más ineficiente temporalmente que uno de backtracking

c. es un caso especial de ramificación y poda

d. ninguna de las anteriores

7. La búsqueda binaria

a. es un ejemplo de búsqueda secuencial

b. es un ejemplo de búsqueda lineal

c. es un ejemplo de fuerza bruta

d. ninguna de las anteriores

8. Se dispone de un algoritmo superlineal, un sublineal y un cuadrático que resuelven el mismo problema. en general, ¿cual elegimos?

a. sublineal

b. superlineal

c. cuadrático

d. depende de las constantes multiplicativas

9. El orden inferior de complejidad (omega) nos da una idea de

a. el comportamiento del algoritmo en el mejor de los casos

b. el comportamiento del algoritmo en el caso promedio

c. el comportamiento del algoritmo en el caso de tallas pequeñas

d. ninguna de las anteriores

10. Si tenemos un algoritmo lineal y otro logarítmico que resuelven el mismo problema

a. el logarítmico siempre es mejor

b. el lineal siempre es mejor

c. para tallas grandes el lineal pudiera ser mejor

d. ninguna de las anteriores

11. ¿por qué se suele realizar en análisis de algoritmos centrándonos en el comportamiento para problemas de talla grande?

a. el análisis de algoritmos no se realiza de ese modo

b. porque para problemas pequeños la elección del algoritmo es menos importante porque los problemas pequeños se resuelven rápidamente en todo caso

c. porque así aseguramos el comportamiento del algoritmo en el mejor de los casos

d. ninguna de las anteriores

12. La búsqueda con centinela respecto a la búsqueda sin centinela

a. mejora el orden de complejidad

b. empeora el orden de complejidad

c. el orden de complejidad es el mismo pero se realizan menos pasos

d. ninguna de las anteriores

13. El algoritmo quicksort de ordenación

a. es un ejemplo de algoritmo voraz

b. es un ejemplo de divide y vencerás

c. es un ejemplo de vuelta atrás

d. ninguna de las anteriores

14. Un TAD pila se implementa de dos maneras alternativas (A y B). La alternativa A es una lista enlazada de elementos con un puntero a la cima de la pila. La alternativa B es mediante un bloque de memoria contigua. ¿cual de las siguientes afirmaciones es cierta?

a. un problema con la alternativa B es que tengo que decidir cuantos elementos reservo a priori

b. un problema con la alternativa B es que necesito un puntero adicional para cada valor almacenado en la pila

c. un problema con la alternativa A es decidir cuantos elementos reservo memoria en el bloque

d. ninguna de las anteriores

15. Un TAD lista se implementa de dos maneras alternativas (A y B). La alternativa A es un puntero a un bloque contiguo de elementos. La alternativa B es mediante una lista doblemente enlazada con un puntero al primer elemento de la lista y doble enlace entre cada elemento de la lista (hacia delante y hacia atrás). ¿cual de las siguientes afirmaciones es cierta?

a. la alternativa A me facilita la gestión de los accesos, tanto para inserciones como para eliminaciones de elementos de la lista

b. para obtener el n-ésimo elemento la alternativa A es más eficaz pues los elementos están dispuestos de manera contigua en memoria

c. un problema con la alternativa A es que utiliza más memoria que la B.

d. ninguna de las anteriores

2. (1.5 punto) Construye la especificación formal del TAD Natural con los constructores Cero y Sucesor y las operaciones EsCero, Igual, Suma, Antecesor y Diferencia.

ESPECIFICACIÓN FORMAL DEL TAD NATURAL

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: ∀a,b ∈ Natural

EsCero(Cero)	⇒	True
EsCero (Sucesor(a))	⇒	False
Igual(a, Cero)	⇒	EsCero(a)
Igual(Sucesor(a), Cero)	⇒	False
Igual(Sucesor(a), Sucesor(b))	⇒	Igual(a,b)
Suma(a, Cero)	⇒	a
Suma(Sucesor(a), b)	⇒	Sucesor(Suma(a,b))
Antecesor(Sucesor(a))	⇒	a
Antecesor(Cero)	⇒	ERROR
Diferencia(Sucesor(a), Sucesor(b))	⇒	Diferencia(a,b)
Diferencia(a, Antecesor(a))	⇒	ERROR

3. (2.5 puntos) Una bibliotecaria dispone de tres pilas de libros y quiere construir una lista con todos los libros distintos que hay en las tres pilas. Cada libro se identifica en la pila por su isbn (esto es, el tipoelemento es un long que almacena el isbn del libro). Construye una función que tome esas tres pilas, construya una lista con los libros distintos que hay (sin repeticiones) y, finalmente, deje las tres pilas como estaban. Utiliza para ello las operaciones conocidas en los TADs Pila (PilaVacia, EsVacia, Cima, Push y Pop) y Lista (crea, fin, primero, siguiente, anterior, esVacia, recupera, longitud, inserta, suprime, modifica) realice esta operación.

```
// rellena los argumentos y tipo que recibe la función
void crea_Lista_dadas_3_Pilas(                    )
{
// rellena las instrucciones necesarias
.....
}
```

```
void crea_Lista_dadas_3_Pilas(TPILA p1, TPILA  p2, TPILA  p3, TLISTA *l){
  TIPOELEMENTO isbn, e;
   int count=0;
   TPILA aux=NULL;
   TNODOLISTA p;
   crea(l);

   while(!EsVacia(p1)){
         count=0;
         Cima(p1,&isbn);
         Pop(&p1);
         Push(&aux,isbn);
         if(!esVacia(*l))
               for(p=primero(*l);p!=fin(*l);p=siguiente(*l,p)){
                     recupera(*l,p,&e);
                     if(e==isbn){
                           count=1;
                           break;
                     }
               }
               if(!count)
                        inserta(l,fin(*l), isbn);
   }

   while(!EsVacia(aux)){
         Cima(aux,&isbn);
         Pop(&aux);
         Push(&p1,isbn);
   }
```

```
while(!EsVacia(p2)){
    count=0;
    Cima(p2,&isbn);
    Pop(&p2);
    Push(&aux, isbn);
    if(!esVacia(*l))
        for(p=primero(*l);p!=fin(*l);p=siguiente(*l,p)){
            recupera(*l,p,&e);
            if(e==isbn){
                count=1;
                break;
            }
        }
        if(!count)
            inserta(l,fin(*l), isbn);
}
while(!EsVacia(aux)){
    Cima(aux,&isbn);
    Pop(&aux);
    Push(&p2,isbn);
}
while(!EsVacia(p3)){
    count=0;
    Cima(p3,&isbn);
    Pop(&p3);
    Push(&aux, isbn);
    if(!esVacia(*l))
        for(p=primero(*l);p!=fin(*l);p=siguiente(*l,p)){
            recupera(*l,p,&e);
            if(e==isbn){
                count=1;
                break;
            }}
```

```
            if(!count)
                inserta(l,fin(*l), isbn);
    }

    while(!EsVacia(aux)){
        Cima(aux,&isbn);
        Pop(&aux);
        Push(&p3,isbn);
    }
}
```

4. (3 puntos) Dado los siguientes programas, determina el orden de complejidad de los mismos siendo la talla del problema el número N

Programa A:

```
....

int main()
{
....

k = f1(N)+f2(N);


}
```

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

for (aux=0;aux<num;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;

if (num < 5)
   return (num*1.1)

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

return(rdo);
}
```

Programa C:

```
....

int main()
{
....

k = f1(N)+f2(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;
}
```

Programa A:

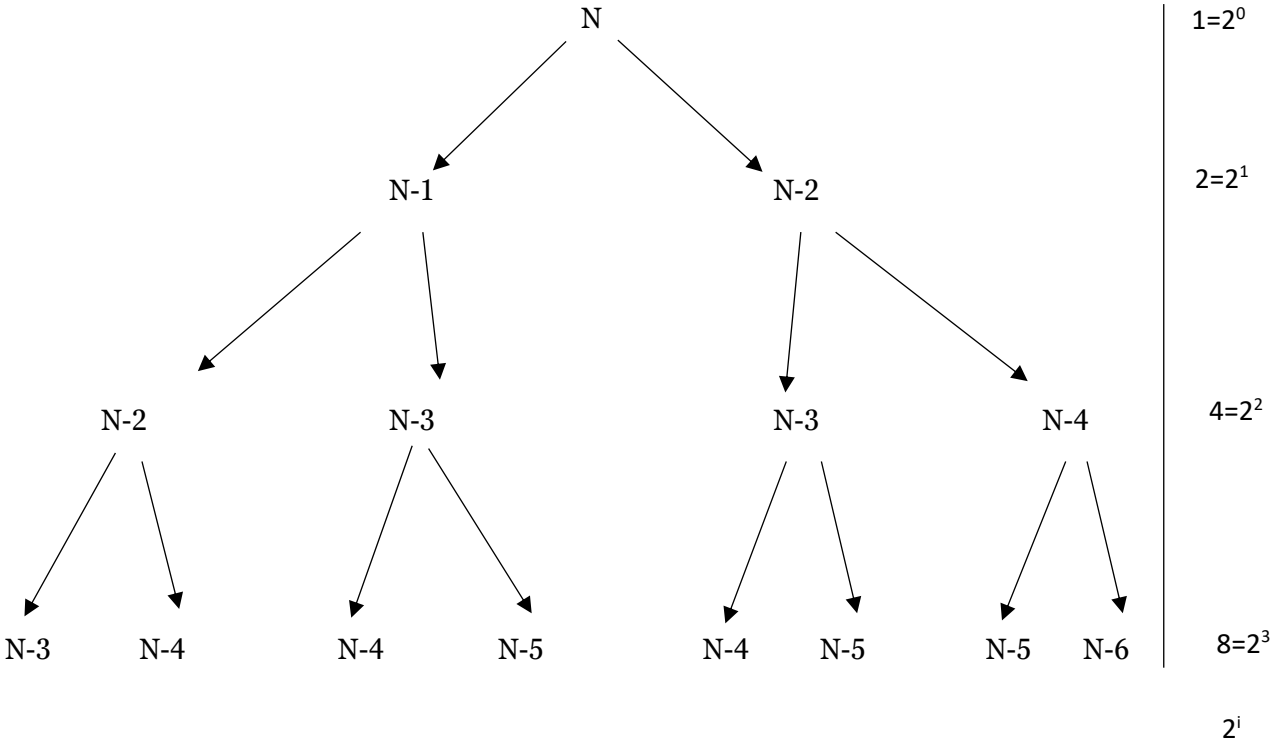
$máx(f_1,f_2) = máx(O(N^3) , O(N^2)) = O(N^3)$ → Complejidad cúbica

Programa B:

máx(f_1,*O*f_2)

f_1?

f_1 es un algoritmo recursivo que divide un problema de tamaño N en 2 problemas de tamaño N-1 y N-2:



Peor caso: 2^N → $O(f_1) = O(2^N)$

máx(f_1,f_2) = *máx*($O(2^N)$,$O(N)$) = $O(2^N)$ → Complejidad exponencial

Programa C:

máx(f_1,f_2) = *máx*($O(N^2)$, $O(N)$) = $O(N^2)$ → Complejidad cuadrática

EXAMEN DE MAYO DE 2011 DE PROGRAMACIÓN II

1. **Test (3 puntos. Cada respuesta correcta suma 0.15 puntos. Cada respuesta incorrecta resta 0.07 puntos)**

1. La parte sintáctica de la especificación formal de un Tipo Abstracto de Dato (TAD)

a. explicita los detalles de implementación de las operaciones del nuevo de tipo de datos
b. explicita las operaciones (argumentos que tienen y valores que devuelven) del TAD
c. explicita el comportamiento o semántica que tienen las operaciones del TAD
d. ninguna de las anteriores

2. La parte semántica de la especificación formal de un Tipo Abstracto de Dato (TAD)

a. explicita los detalles de implementación de las operaciones del nuevo de tipo de datos
b. explicita exclusivamente las operaciones (argumentos que tienen y valores que devuelven) del TAD
c. describe de manera informal que tipo de argumentos se esperan en las operaciones del TAD
d. ninguna de las anteriores

3. Un Tipo Abstracto de Dato (TAD) permite

a. que los programan que utilicen la especificación del TAD no tienen que ser cambiados cada vez que la implementación del TAD cambie
b. obligar a cambiar los programas que utilicen la especificación del TAD cada vez que haya cambios en la implementación del TAD
c. que los programas implementen directamente las operaciones del TAD para cambiar su comportamiento
d. ninguna de las anteriores

4. En verificación de software una falla es

a. un estado intermedio incorrecto en que entra un programa al ejecutarse
b. un síntoma de que existe un error
c. un caso de prueba
d. ninguna de las anteriores

5. La verificación formal de programas está basada

a. en probar los programas con casos de prueba exhaustivos
b. en utilizar laboratorios de usuarios que prueben recurrentemente los programas
c. en especificaciones matemáticas (p.e. algebraicas) sobre lo que se espera de las operaciones del programa
d. ninguna de las anteriores

6. En un TAD pila
    a. es posible insertar elementos por el tope o por el fondo de la pila
    b. solo es posible insertar elementos por el fondo de la pila
    c. es posible acceder de manera directa a cualquier elemento de la pila
    d. ninguna de las anteriores

7. En un TAD cola
    a. es posible eliminar cualquier elemento bajo cualquier circunstancia
    b. solo es posible eliminar el primero de la cola
    c. solo es posible eliminar el último de la cola
    d. ninguna de las anteriores

8. La búsqueda con centinela
    a. tiene el mismo orden de complejidad que la búsqueda sin centinela pero realiza más pasos
    b. realiza una búsqueda binaria y, por tanto, mejora el orden de complejidad con respecto a la búsqueda sin centinela
    c. consume menos memoria que la búsqueda sin centinela
    d. ninguna de las anteriores

9. La búsqueda binaria
    a. tiene orden de complejidad cuadrático
    b. realiza dos pasadas para ordenar los datos
    c. tiene orden de complejidad lineal
    d. ninguna de las anteriores

10. Para un mismo problema disponemos de un algoritmo de orden n^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

11. Dos algoritmos A y B tienen ambos orden superior lineal, el algoritmo A tiene orden inferior 1 y el algoritmo B tiene orden inferior sublineal, ¿Cuál elegimos?
    a. A
    b. B
    c. es irrelevante pues en promedio ambos darán un rendimiento equivalente
    d. ninguna de las anteriores

12. El orden exacto de complejidad (Θ) nos da una idea de
    a. el comportamiento del algoritmo en el mejor de los casos
    b. el comportamiento del algoritmo en el peor de los casos
    c. el comportamiento del algoritmo en el mejor de los casos y en el peor de los casos
    d. ninguna de las anteriores

13. ¿por qué se suele realizar el análisis de algoritmos centrándonos en el comportamiento para problemas de talla grande?

a. porque nos permite comprender como se comportan los algoritmos en condiciones extremas
b. porque en la realidad nunca se presentan problemas de talla pequeña
c. porque así aseguramos el comportamiento del algoritmo en el mejor de los casos
d. ninguna de las anteriores

14. El algoritmo quicksort de ordenación

a. es menos eficiente temporalmente que el algoritmo de la burbuja
b. es un ejemplo de algoritmo lineal
c. es un ejemplo de algoritmo cuadrático
d. ninguna de las anteriores

15. Los algoritmos de fuerza bruta

a. suelen implicar alto coste computacional
b. toman una decisión y nunca la reconsideran
c. realizan una poda del espacio de búsqueda de soluciones
d. ninguna de las anteriores

16. Un algoritmo voraz

a. divide el problema en subproblemas de modo recursivo
b. suele ser más eficiente temporalmente que uno de backtracking
c. es un caso especial de ramificación y poda
d. ninguna de las anteriores

17. Un algoritmo voraz

a. nunca reconsidera una decisión tomada previamente
b. puede reconsiderar una decisión previa en caso de llegar a un callejón sin salida
c. es un caso especial de fuerza bruta
d. ninguna de las anteriores

18. La programación dinámica

a. consiste en sacrificar tiempo computacional para conseguir ahorrar uso de memoria
b. consiste en sacrificar memoria para conseguir mejor tiempo computacional
c. consiste en decidir dinámicamente (en tiempo de ejecución) entre ahorrar memoria o ahorrar tiempo
d. ninguna de las anteriores

19. La técnica de ramificación y poda…

a. es una variante de la fuerza bruta
b. es una variante de los algoritmos voraces
c. realiza un recorrido informado del espacio de búsqueda
d. ninguna de las anteriores

20. Los algoritmos de vuelta atrás o backtracking…
    a. son similares en filosofía a los algoritmos fuerza bruta
    b. realizan una búsqueda dentro de un espacio de búsqueda que se suele representar de forma lineal
    c. nunca realizan un recorrido en profundidad del espacio de búsqueda
    d. ninguna de las anteriores

2. (1 pto) En la ETSE, se desea diseñar una aplicación web de recepción de solicitudes para usar un ordenador portátil. Se quiere poder almacenar las solicitudes recibidas por parte de profesores de la manera más efectiva y eficaz. Cuando el ordenador es devuelto por un profesor, se pasa al siguiente que lo hubiese pedido (según el orden establecido por la fecha y hora de solicitud). Responde brevemente a las siguientes preguntas:

a. ¿Qué TAD usarías para gestionar el conjunto de personas a la espera? ¿por qué?

Un TAD cola, ya que se seguiría una estructura FIFO (First-In, First-Out), pues el primero en pedir ordenador es el primero que lo obtiene, y el último en solicitarlo, el último en recibirlo.

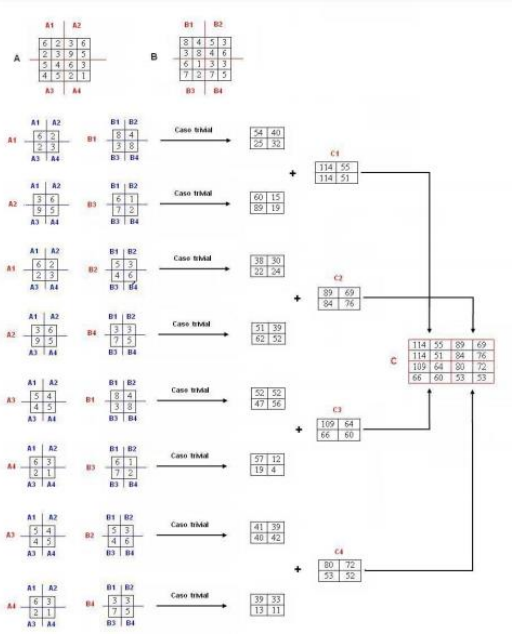
b. La gente que puede pedir el ordenador es un conjunto restringido de personas (profesores de la ETSE) que tiene un tamaño reducido y además hay pocos cambios en la plantilla de profesorado. Teniendo en cuenta esto, ¿qué estrategia utilizarías internamente en el TAD para almacenar los datos: memoria contigua o memoria enlazada? ¿por qué?

La estrategia más eficiente de almacenamiento en este caso es la contigua, debido a que se conoce el número de profesores a priori. Las inserciones y eliminaciones en este caso podrían suponer una mayor dificultad, pero ya que se especifica que son poco frecuentes, emplear memoria enlazada no supondría demasiada ventaja, pues no sería necesario expandir la longitud de la cola.

3. (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?

Al ser matrices cuadradas, estas serán de orden ‘n’ (n filas y n columnas). El orden de complejidad será cúbico ($O(n^3)$) debido al empleo de 3 bucles que recorren la fila de la primera matriz, la columna de la segunda y la columna de la primera, respectivamente.

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?



Según este algoritmo se realizarán 8 productos de matrices de tamaño n/2. Estos productos finalmente se sumarán de 2 en 2, obteniendo 4 matrices n/2 que se colocaran en los cuadrantes correspondientes de la matriz resultado. La suma, al tratarse de una suma de matrices, así como el proceso de formación de las matrices n/2 y de sustitución de los cuadrantes de la matriz resultado, requerirán de dos bucles hasta n/2, por lo que serán de complejidad cuadrática. De esta forma el orden de complejidad de la multiplicación mediante el método divide y vencerás será de la forma: $t(n) = 8*t(n/2) + O(n^2)$; ya que $8>2^2$ → $t(n) \in O(n^{\log_2 8}) = O(n^3)$

4. (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<num;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;

if (num == 1)
   return (1.1)

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<100;aux++)
  for (aux2=0;aux2<num;aux2++)
      rdo += aux*aux2;

return (rdo);
}
```

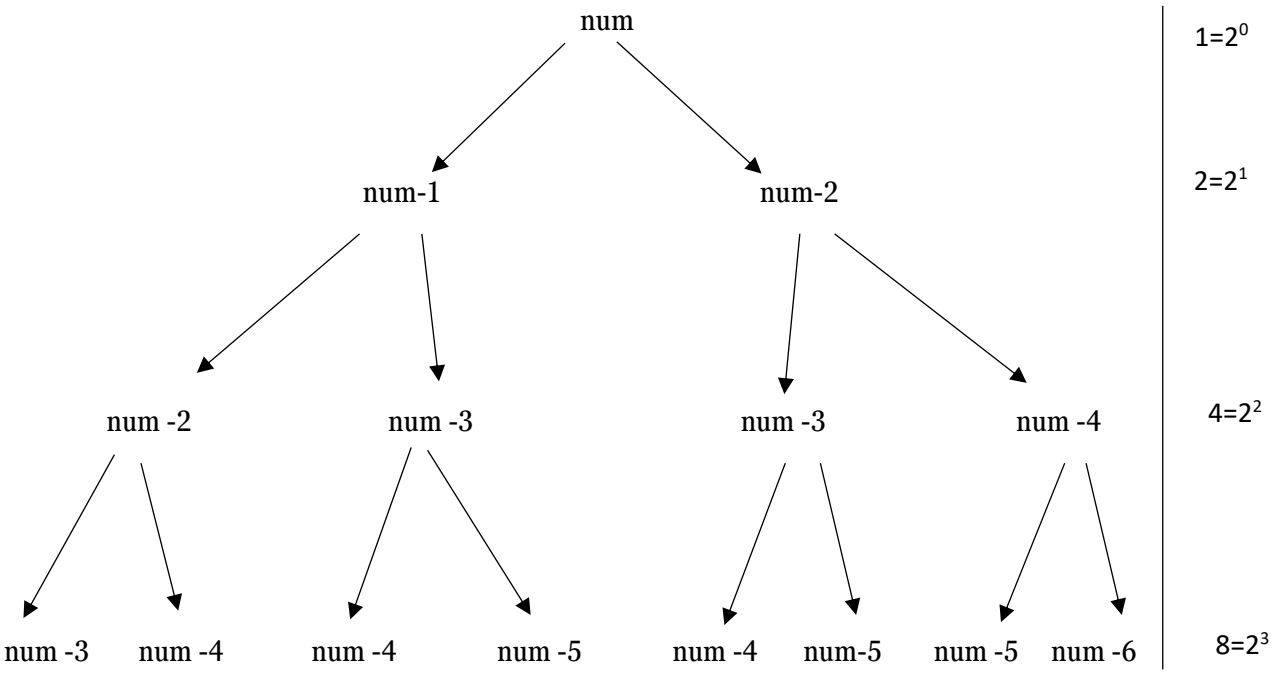
```
float f2(int num)
{
int aux;
float rdo=0;

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

return rdo;
}
```

PROGRAMA A:

$f_1 \in O(num^3)$; $f_2 \in O(num^2)$ → Programa A $\in O(num^3)$

$f_1 \in \Omega(num^3)$; $f_2 \in \Omega(num^2)$ → Programa A $\in \Omega(num^2)$

PROGRAMA B:



Peor caso: 2^{num} → $f_1 \in O(2^{num})$ → Programa B $\in O(2^{num})$

Mejor caso: Programa B $\in O(1)$

PROGRAMA C:

$f_1 \in O(num)$; $f_2 \in O(num)$ → Programa C $\in O(num)$

$f_1 \in \Omega(num)$; $f_2 \in \Omega(num)$ → Programa A $\in \Omega(num)$

5. (1.5 punto) Construye la especificación formal del TAD Conjunto (que modele conjuntos de elementos todos ellos de tipo elemento), con las operaciones crear (crea el conjunto vacío), añadir (añade un elemento al conjunto), borrar (elimina un elemento del conjunto), pertenece (saber si un elemento está en el conjunto) y esvacio (saber si el conjunto está vacío).

ESPECIFICACIÓN FORMAL DEL TAD CONJUNTO

Tipo: Conjunto

Sintaxis:

*crear		→	Conjunto
*añadir	(Conjunto, Elemento)	→	Conjunto
borrar	(Conjunto, Elemento)	→	Conjunto
pertenece	(Conjunto, Elemento)	→	Boolean
esvacio	(Conjunto)	→	Boolean

Semántica: $\forall e \in$ Elemento, $\forall c \in$ Conjunto

borrar(crear, e)	⇒	ERROR
borrar(añadir(crear,e), e)	⇒	crear
pertenece(crear, e)	⇒	False
pertenece(añadir(c,e), e)	⇒	True
esvacio(añadir(c,e))	⇒	False
esvacio(crear)	⇒	True

6. (2 puntos) Una gestora de reservas de una sala de juntas dispone de tres listas independientes donde se almacena qué personas usaron la sala en el pasado. Una lista almacena los directivos que la usaron, otra lista almacena los administrativos que la usaron y la última lista almacena los informáticos que la usaron. La única información que se guarda de cada persona es su DNI. En ninguna de las listas existen individuos repetidos y tampoco aparece ningún individuo en más de una lista. Para evitar gestionar 3 listas quiere un procedimiento informático que le genere una lista con todos los individuos que han usado la sala en el pasado. Utiliza las operaciones necesarias del TAD Lista para resolverle su problema implementando la función….

// rellena los argumentos y tipo que recibe la función

void crea_Lista_Unica( )

{

// rellena las instrucciones necesarias

……

}

```
void crea_Lista_Unica(TLISTA *L1, TLISTA *L2, TLISTA *L3, TLISTA *LU){
  TIPOELEMENTOLISTA DNI;
  TNODOLISTA p;
  crea(LU);

  p=primero(*L1);
  while(!esVacia(*L1)){
    recupera(*L1,p,&DNI);
    suprime(L1,p);
    inserta(LU,fin(*LU),DNI);
  }

  p=primero(*L2);
  while(!esVacia(*L2)){
    recupera(*L2,p,&DNI);
    suprime(L2,p);
    inserta(LU,fin(*LU),DNI);
  }
```

```
    p=primero(*L3);
    while(!esVacia(*L3)){
        recupera(*L3,p,&DNI);
        suprime(L3,p);
        inserta(LU,fin(*LU),DNI);
    }

    destruye(L1);
    *L1=NULL;
    destruye(L2);
    *L2=NULL;
    destruye(L3);
    *L3=NULL;
}
```

EXAMEN DE JULIO DE 2011 DE PROGRAMACIÓN II

1. **Test (2 puntos. Cada respuesta correcta suma 0.2 puntos. Cada respuesta incorrecta**

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 (Θ) 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 el 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

La complejidad espacial estudia el almacenamiento requerido por un programa para resolver un problema propuesto.

En cambio, la complejidad temporal estudia el tiempo requerido para alcanzar la solución al problema.

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?

Bloques contiguos:

- *Se necesita conocer el tamaño de la lista a priori, en caso de no saberlo se apunta al alza, por lo que se desperdiciaría memoria o incluso podría llegar a no ser suficiente, por no haber un bloque de memoria contiguo tan grande.*
- *El acceso a los elementos de forma secuencial es más veloz.*
- *La inserción y eliminación de elementos es lineal debido al desplazamiento necesario.*
- *La operación de destrucción es instantánea.*

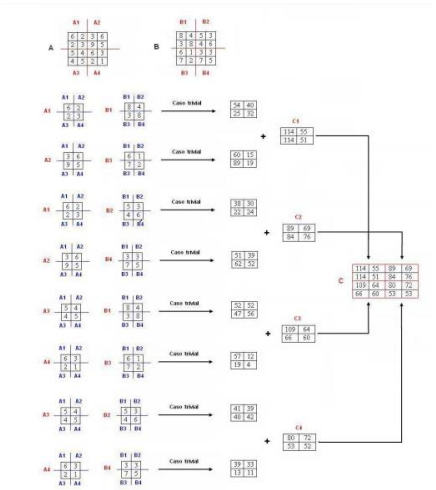
Representación enlazada:

- *No se necesita conocer el tamaño a priori.*
- *Permite aprovechar toda la memoria.*
- *La inserción y eliminación es instantánea, ya que solo se requiere ajustar los punteros relacionados con el elemento insertado/eliminado.*
- *La operación de destrucción es lineal, se deben liberar las celdas de forma individual.*

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?

Al ser matrices cuadradas, estas serán de orden 'n' (n filas y n columnas). El orden de complejidad será cúbico ($O(n^3)$) debido al empleo de 3 bucles que recorren la fila de la primera matriz, la columna de la segunda y la columna de la primera, respectivamente.

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?



Según este algoritmo se realizarán 8 productos de matrices de tamaño n/2. Estos productos finalmente se sumarán de 2 en 2, obteniendo 4 matrices n/2 que se colocaran en los cuadrantes correspondientes de la matriz resultado. La suma, al tratarse de una suma de matrices, así como el proceso de formación de las matrices n/2 y de sustitución de los cuadrantes de la matriz resultado, requerirán de dos bucles hasta n/2, por lo que serán de complejidad cuadrática. De esta forma el orden de complejidad de la multiplicación mediante el método divide y vencerás será de la forma: $t(n) = 8*t(n/2) + + O(n^2)$; ya que $8>2^2$ → $t(n) \in O(n^{\log_2 8}) = O(n^3)$

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<num;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;

if (num == 1)
   return (1.1)

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<100;aux++)
  for (aux2=0;aux2<num;aux2++)
      rdo += aux*aux2;

return (rdo);
}
```

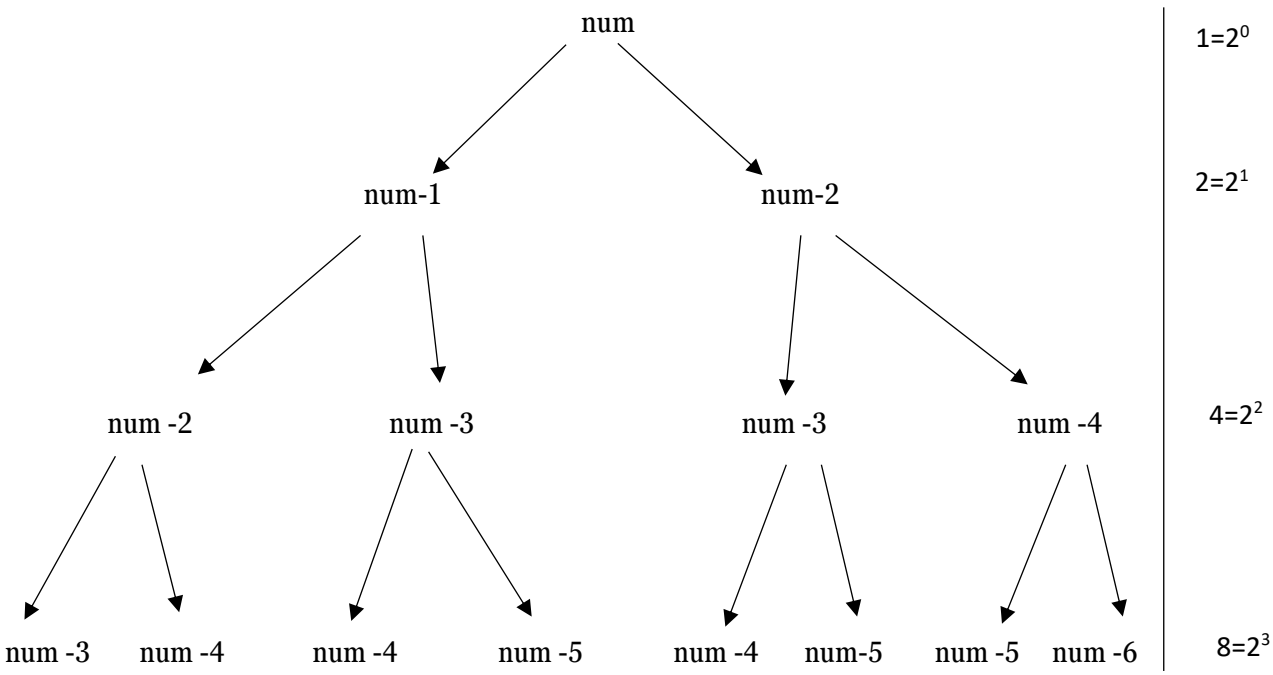
```
float f2(int num)
{
int aux;
float rdo=0;

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

return rdo;
}
```

PROGRAMA A:

$f_1 \in O(num^3)$; $f_2 \in O(num^2)$ → Programa A $\in O(num^3)$

$f_1 \in \Omega(num^3)$; $f_2 \in \Omega(num^2)$ → Programa A $\in \Omega(num^2)$

PROGRAMA B:



Peor caso: 2^{num} → $f_1 \in O(2^{num})$ → Programa B $\in O(2^{num})$

Mejor caso: Programa B $\in O(1)$

PROGRAMA C:

$f_1 \in O(num)$; $f_2 \in O(num)$ → Programa C $\in O(num)$

$f_1 \in \Omega(num)$; $f_2 \in \Omega(num)$ → Programa A $\in \Omega(num)$

6. (1.5 punto) Partiendo de la especificación formal del TAD Natural vista en clase (ver a continuación) completarla mediante 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)**

ESPECIFICACIÓN FORMAL DEL TAD NATURAL

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
Producto	(Natural, Natural)	→	Natural
Potencia	(Natural, Natural)	→	Natural

Semántica: ∀a,b ∈ Natural

EsCero(Cero)	⇒	True
EsCero (Sucesor(a))	⇒	False
Igual(a, Cero)	⇒	EsCero(a)
Igual(Sucesor(a), Cero)	⇒	False
Igual(Sucesor(a), Sucesor(b))	⇒	Igual(a,b)
Suma(a, Cero)	⇒	a
Suma(Sucesor(a), b)	⇒	Sucesor(Suma(a,b))
Antecesor(Sucesor(a))	⇒	a
Antecesor(Cero)	⇒	ERROR
Diferencia(a,Cero)	⇒	a
Diferencia(Sucesor(a), Sucesor(b))	⇒	Diferencia(a,b)
Diferencia(a, Antecesor(a))	⇒	ERROR
Producto(a, Cero)	⇒	Cero
Producto(a, Sucesor(Cero))	⇒	a
Potencia(a, Cero)	⇒	Sucesor(Cero)
Potencia(a, Sucesor(Cero))	⇒	a

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
……
```

```
long primeroColaPrio(TCOLAPRIO   procesosencolados  ){
        short maxprio;
        long primerprocesomaxprio;
        POSICION p;

        if(esVacia(procesosencolados))
                return -1;
        p = primero(procesosencolados);
        recupera(procesosencolados, p, &e);
        maxprio = e.prioridad;
        primerprocesomaxprio= e.proceso;
        p = siguiente(procesosencolados,p);
```

```
        while(p!=fin(procesosencolados)){
                recupera(procesosencolados, p, &e);
                if(e.prioridad > maxprio){
                        maxprio = e.prioridad;
                        primerprocesomaxprio = e.proceso;
                }

                p = siguiente(procesosencolados, p)
        }
        return primerprocesomaxprio;
}
```

EXAMEN DE MAYO DE 2012 DE PROGRAMACIÓN II

1. (1 pto) Explica brevemente qué significa que la implementación de un TAD sea opaca y qué ventajas supone eso. Haz la explicación genérica (independiente de cualquier lenguaje de programación).

Que la implementación de un TAD sea opaca, implica que el usuario desconoce los detalles de implementación del mismo, de tal forma que solo tiene acceso a su interfaz para poder emplear sus operaciones en otros programas. Esto proporciona una mayor seguridad y privacidad al código.

2. (1 pto) Se dispone de dos implementaciones distintas de un TAD cola. Una de ellas (implementación A) mediante un bloque contiguo de elementos en memoria, en la que la creación de la cola implica la reserva de un tamaño máximo de elementos (tamaño_max). En esta implementación A nunca podrá haber más de tamaño_max elementos encolados. Otra implementación (implementación B) es a través de una lista enlazada de nodos en la que cada vez que encolamos un elemento se crea dinámicamente el nodo correspondiente en memoria (esto es, la creación de la lista no pide memoria para elemento alguno, se pide memoria al introducir individualmente los elementos). Responde a las siguientes cuestiones de manera breve:

· En igualdad del resto de circunstancias, si sabemos que el tamaño en bytes del TIPOELEMENTO a guardar en la cola es muy grande, ¿que implementación deberíamos preferir y por qué?

Deberíamos decantarnos por lista enlazada, debido a que es posible que no existan bloques de memoria contiguos para tamaños muy grandes.

· En igualdad del resto de circunstancias, si desconocemos el número típico de elementos que será necesario encolar y además sabemos que en tiempo de ejecución este número va a ser muy variable y puede llegar a ser muy grande, ¿que implementación deberíamos preferir y por qué?

Nuevamente, deberíamos decantarnos por lista enlazada debido a que no es preciso definir un valor máximo de antemano que al final pueda resultar insuficiente o por el contrario, produzca un desperdicio de memoria durante la ejecución, sino que permite aprovechar toda la memoria. Además, las inserciones y eliminaciones de elementos son bastante rápidas, ya que solo requieren modificar la dirección a la que apuntan los punteros relacionados con el elemento a insertar/eliminar.

· En igualdad del resto de circunstancias, si sabemos que la operación más típica que se va a invocar sobre la cola es la de destrucción, pues la aplicación para que usamos el TAD va a estar constantemente creando y destruyendo colas, ¿que implementación deberíamos preferir y por qué?

En este caso, debemos decantarnos por memoria contigua, ya que la destrucción en este caso es inmediata (de orden de complejidad constante), mientras que en una lista enlazada habría que destruir individualmente cada una de las celdas reservadas (orden de complejidad lineal).

· Si la implementación A opta por mover todos los elementos del bloque continuo una posición a la izquierda cada vez que eliminamos el primero del bloque (esto es, cada vez que extraemos el primero de la cola), que operación es más eficiente: sacar el primero de la cola en la implementación A o sacar el primero de la cola en la implementación B? por qué?

Sacar el primero de la cola en la implementación B, ya que en este caso no habría que realizar ningún desplazamiento, solamente ajustar el puntero de inicio.

3. Test (3 puntos. Cada respuesta correcta suma 0.2 puntos. Cada respuesta incorrecta resta 0.1 puntos)

1. Cual de las siguientes afirmaciones es cierta respecto a un Tipo de Datos primitivo convencional (p.e. float o int) frente a un Tipo Abstracto de Dato (TAD)

a. el TAD no tiene operaciones asociadas

b. el TAD no oculta la implementación del tipo y el TD primitivo si.

c. no existe ninguna diferencia entre ambos, son dos formas equivalentes de referirnos a lo mismo

d. ninguna de las anteriores

2. La parte semántica de la especificación formal de un Tipo Abstracto de Dato (TAD)

a. describe el comportamiento esperado de las operaciones del TAD

b. explicita los detalles de implementación de las operaciones del nuevo de tipo de datos

c. explicita exclusivamente las operaciones (argumentos que tienen y valores que devuelven) del TAD

d. describe de manera informal que tipo de argumentos se esperan en las operaciones del TAD

3. Dadas dos implementaciones distintas de un TAD (p.e. un TAD lista implementado como un bloque contiguo en memoria o como una lista enlazada de nodos), cual de las siguientes afirmaciones es cierta:

a. puede ser que una implementación cumpla la especificación semántica y otra implementación no la cumple. En todo caso, ambas implementaciones serían correctas.

b. puede ser que una implementación cumpla la especificación sintáctica y otra implementación no la cumple. En todo caso, ambas implementaciones serían correctas.

c. puede ser que una implementación sea más eficiente que otra, aún realizando la misma labor

d. ninguna de las anteriores

4. En referencia a las operaciones permitidas por un TAD pila, un TAD lista y un TAD cola...

a. el TAD cola es el más flexible de los tres pues soporta una variedad más amplia de operaciones posibles

b. el TAD lista es el más flexible de los tres pues soporta una variedad más amplia de operaciones posibles

c. el TAD pila es el más flexible de los tres pues soporta una variedad más amplia de operaciones posibles

d. ninguna de las anteriores

5. Una aplicación necesita almacenar objetos en una estructura que permita inserciones al principio de la estructura, inserciones al final de la estructura y, en ocasiones, inserciones en el medio de la estructura. ¿Qué TAD puedo usar?

a. un TAD cola

b. un TAD pila

c. un TAD cola con prioridad

d. un TAD lista

6. La búsqueda sin centinela

a. tiene el mismo orden de complejidad que la búsqueda con centinela pero realiza más pasos

b. realiza una búsqueda binaria y, por tanto, mejora el orden de complejidad con respecto a la búsqueda con centinela

c. consume más memoria que la búsqueda con centinela

d. tiene un orden de complejidad mayor que la búsqueda con centinela

7. La búsqueda binaria

a. tiene orden de complejidad lineal

b. realiza dos pasadas para buscar los datos

c. tiene orden de complejidad sublineal

d. ninguna de las anteriores

8. El algoritmo quicksort de ordenación

a. es menos eficiente temporalmente que el algoritmo de la burbuja

b. es un ejemplo de algoritmo lineal

c. es menos eficiente temporalmente que el algoritmo de selección

d. ninguna de las anteriores

9. ¿qué significa que se realice el análisis de algoritmos de manera asintótica?

a. que se realiza un estudio de los algoritmos centrándonos en su comportamiento en el mejor de los casos

b. que se realiza un estudio de los algoritmos centrándonos en su comportamiento en el peor de los casos

c. que se realiza un estudio de los algoritmos centrándonos en su comportamiento en el caso promedio

d. que se realiza un estudio de los algoritmos centrándonos en su comportamiento para tallas grandes

10. El orden de complejidad omega (Ω) nos da una idea de

a. como estan acotados superiormente los casos del problema (peores casos)

b. como estan acotados inferiormente los casos del problema (mejores casos)

c. como estan acotados superior e inferiormente los casos del problema

d. ninguna de las anteriores

11. Para solucionar el mismo problema disponemos de un algoritmo de orden n^3, otro de orden n! y otro de orden 2^n . ¿Cuál elegimos?

a. el cúbico (n^3)

b. el cuadrático (2^n)

c. el de orden n!

d. ninguno es viable como solución computacional

12. Los algoritmos de fuerza bruta

a. suelen implicar bajo coste computacional

b. suelen dividir el espacio de búsqueda en subproblemas

c. realizan una poda del espacio de búsqueda de soluciones

d. ninguna de las anteriores

13. Un algoritmo voraz

a. puede reconsiderar decisiones tomadas previamente mediante vuelta atrás

b. suele ser más eficiente temporalmente que uno de fuerza bruta

c. es un caso especial de ramificación y poda

d. ninguna de las anteriores

14. Un algoritmo divide y vencerás

a. siempre es más eficaz computacionalmente que una resolución directa del problema

b. aplica una heurística para resolver siempre sólo uno de los subproblemas involucrados

c. es un caso especial de algoritmos voraces

d. ninguna de las anteriores

15. La programación dinámica

a. consiste en sacrificar tiempo computacional para conseguir ahorrar uso de memoria

b. consiste en sacrificar memoria para conseguir mejor tiempo computacional

c. consiste en decidir dinámicamente (en tiempo de ejecución) entre ahorrar memoria o ahorrar tiempo

d. ninguna de las anteriores

4. (1.4 puntos) Dado el siguiente programa…

```
int main()
{
float saldo=124;
float *p;
float *q;

p=&saldo;
procesa_saldo(p);
q=(float *)malloc (sizeof(float));
*q = *p;

if (saldo > 150)
   saldo = saldo * 1.2;
}
```

```
void procesa_saldo(float *p)
{

if (*p > 100)
   *p = *p + 50;
else
   *p = *p + 5;
}
```

Escribe el valor de las siguientes variables <u>justo antes de finalizar el programa</u>:

a) saldo = 208.8

b) *p = 208.8

c) *q = 174

Escribe el tipo de datos correspondiente a las siguientes expresiones:

a) p: float*

b) saldo: float

c) &p: float**

d) *q: float

5. (2 puntos) Dado los siguientes programas, determina el orden superior (O) de complejidad de los mismos siendo la talla del problema el número N. Se supone que la operación de generación de un número aleatorio (rand) es una operación que se puede realizar de manera unitaria (1 único paso).

Programa A:

```
void main()
{
int x;
int v[N];
int encontrado;

for (i=0; i< N ; i++)
{
      x = rand();
      v[i]=x;
}

encontrado=0;
for (i=1; i<= N ; i++)
{
      x = rand();
      encontrado=encontrado+busca(v,x);
}

}
```

```
short busca(int v[N],int n)
{
int aux;
for (aux=0;aux<N;aux++)
   if v[aux]==n return 1;

return 0;
}
```

Programa B:

```
void main()
{
int x, encontrado;
int v[N];
for (i=0; i< N ; i++)
{
      x = rand();
      v[i]=x;
}
encontrado=0;
for (i=1; i<= N ; i++)
{
      x = rand();
      encontrado=encontrado+buscabin(v,x);
}
}
```

```
short buscabin(int v[N],int n)
{
int ini=0,fin=N-1,mitad;

mitad=(ini+fin)/2;
while ((ini<=fin)&&(v[mitad]!=n))
 {
 if (dato < v[mitad])
          fin=mitad-1;
 else
          ini=mitad+1;
 mitad=(ini+fin)/2;
 }
if (dato==v[mitad]) return 1;
else return 0;
}
```

Programa C:

```
....

int main()
{
....

k = f1(N);


}
```

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

for (aux=0;aux<100;aux++)
  for (aux2=0;aux2<num;aux2++)
     rdo += aux*aux2;

return (rdo);
}
```

Programa D:

```
void main()
{
....

k = f1(N);

}
```

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

if (num == 1)
   return (1.1);

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

return(rdo);
}
```

PROGRAMA A:

busca ∈ O(N) → Programa A ∈ O(N^2) //se llama a la función busca dentro de un bucle hasta N

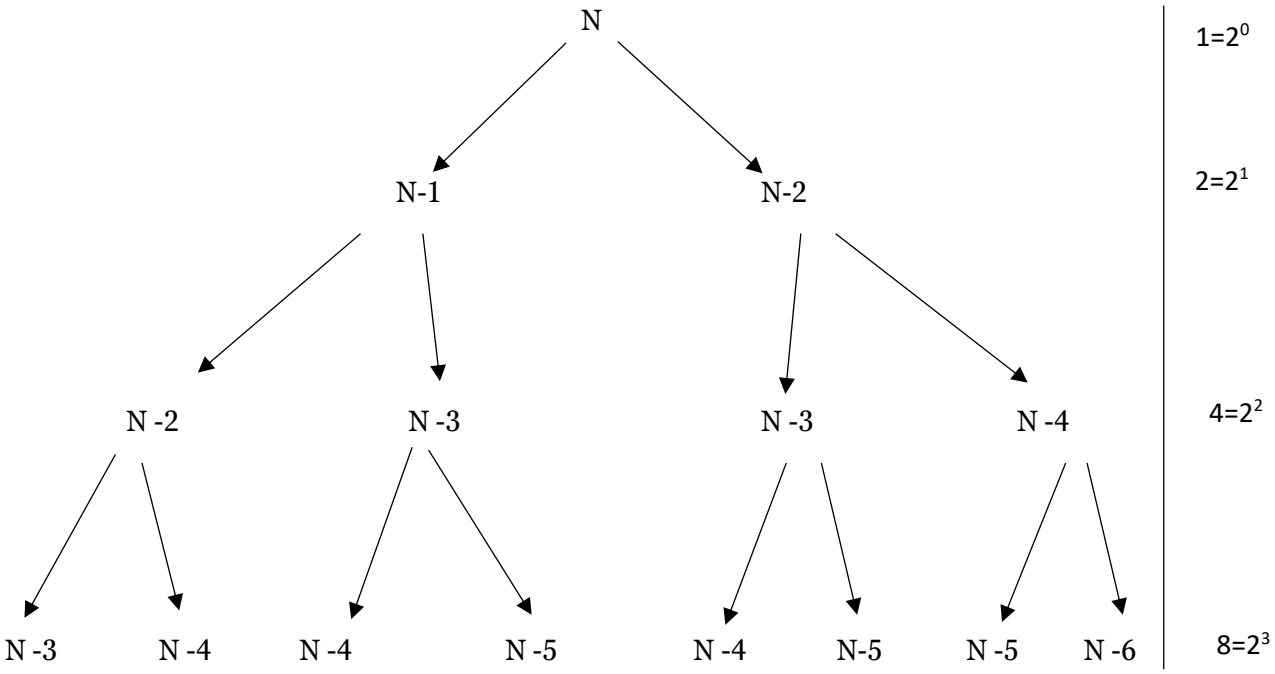
PROGRAMA B:

buscabin∈O(N) → Programa B ∈ O(N^2) //se llama a la función buscabin dentro de un bucle hasta N

PROGRAMA C:

f_1 ∈ O(N) → Programa C ∈ O(N)

PROGRAMA D:



Peor caso: 2^N → $f_1 \in O(2^N)$ → Programa D ∈ $O(2^N)$

6. (1.6 puntos) Dado un TAD Pila, que cuenta con las conocidas operaciones de PilaVacia, EsVacia, Cima, Push y Pop, y dada una Pila que contiene números enteros escribe una función VaciaImparesEnElTope que lo que hace es eliminar elementos impares (valor impar) del tope de la pila hasta que se quede vacía o encuentre un par (quedaría ese par como tope de la pila). La funcionalidad del TAD Pila se supone que ya está disponible en la librería correspondiente (no hay que implementar las funciones, solo usarlas)

// rellena los argumentos y su tipo que recibe la función

void VaciaImparesEnElTope ( )

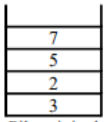
{

// rellena las instrucciones necesarias

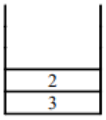
…….

}

Ejemplo:



Pila original



Pila tras ejecutar la función

```
void VaciaImparesEnElTope (TPILA *p){
        TIPOELEMENTOPILA e;
        while(!EsVacia(*p)){
                Cima(*p,&e);
                if(e%2==0)
                        break;
                Pop(p);
        }
}
```