Problema 0-1

```
int *u = NULL, *v=NULL, *w=NULL, *x=NULL, *y=NULL, *z=NULL
int a=2, c=5, d=7, e=11, f=13; b=3;

u= &a;  // *u=2
u*=90;  // *u=90
v= &e;  // *v=e → *=11
e++;    // e=11+1=12
x=v;    // x=v=&e=12
*u=(*u)--;  // *u=90
*x=(*u)+23;  // *x=90+23=113
y=&c;  // *y=c → 5
w=&b;  // *w=b→3
d*=e  // d=d*e=7x113=791
f--;  // f=12
z=&d;  // *z=791
(*z)+=0;  // *z=79
u=v;  // *u=*v *u=113  x=v ∴ v=113
(*z)*=6;  // *z=*z.6=791.6=4746
z=u;  // *z=*u  *z=113
*w=a+b+c+d+e+f;  // *w=90+3+5+4746+113+12=4969
*w-=2019;  // *w=4969-2019=2950
c++;  // 5+1=6
e--  // 113-1=112
u=&a;  // *u=a => *u=90
v=&b;  // w=&b => 2950=b=*w  *v=2950
w=&c;  // *w=c => 6
*x=d+(*u);  // *x=4746+90=4836
*y=e+(*v);  // *y=4836+2950=7786
*z=f+(*w);  // *z=12+7786=7798
```

*u=90 *v=2950 *w=7786 *x=7798 *y=7786 *z=7798

a=90, b=2950, c=7786, d=4746, e=7798, f=12

Problema 0-2

```
int main(int argc, char *argv[])
{
  int *u=NULL, *v=NULL, *w=NULL, *q=NULL;
  int a=101, b=201, c=301;

  u=&c; v=&b; w=&a;
  (*w)+=a+b+c;   101+=101+201+301   *w=704
  (*u)++;        301+1=302
  (*v)=u;        (*v)=201*4=804

  q=w;   q=704
  w=u;   w=302
  u=v;   u=804
  v=q;   v=704
  (*u)-=(*q)%5-a;   (*u)=804-[704%5]+704   (*u)=1504
  (*v)-=(*q)%3-b;   (*v)=704-704%3+1504    (*v)=2206
  (*w)-=(*q)%2-c;   (*w)=302-2206%2+302    (*w)=604

  printf("a=%d b=%d c=%d\n",a,b,c);

                                  a=2206  b=1504  c=604

  return 0;
}
```

Problema 3-2

```
int main (int argc, char *argv[]){
    int **q=NULL, *p=NULL;
    int *a = NULL, *b=NULL, *x = NULL, *y = NULL;
    int N=10, k=0;
10 → a = (int *) malloc (N*sizeof(int));
10 → b = (int *) malloc (N*sizeof(int));
    for(x=a, y=b, k=0; k<N; k++, x++, y++)
      {
18 →  (*x)= 2*k;
27 →  (*y)= 3*k;
      }
    for (k=0; k<N; k++)
      { if (k%2==0)
10 →    q= &a;
        else
10 →    q = &b;
10 →  p = *q;
19 →  p=p+k;
-20 → (*p)*= -1;
-20 → (**q)+=*p;
      }
    for(k=0; <N; k++)
      printf("%x :: a[%d]= %d\n", &(a[k]), k, a[k]);
      printf("\n");
    for(k=0; k<N, k++)
      printf("%x :: b[%d]= %d\n", &(b[k], k, b[k]);
0 →  free(a);
0 →  free(b);
     return 0;
}
```

Problema 6.-2

¿Considere un problema computacional X. En general ¿siempre podrá encontrar un algoritmo para poder resolverlo?

R: No, ya que los problemas computacionales tienen clases de complejidad, tal como los NP (Nondeterministic Polynomial time del inglés) que hasta la fecha no existe un algoritmo que los pueda resolver.

Problema 7-2

Describa los clase computacionales P, NP, NP-Completo y NP-Difícil Por cada clase, de un ejemplo de un problema que pertenezca a la misma.

R:

Clase P: representa a los problemas tratables (faciles) que una computadora pueda resolver, estos pueden encontrar soluciones en un tiempo razonable. Ejemplo: Multiplicación de matrices

Clase NP: Son problemas que no son faciles de encontrar una solución, pero una vez que se encuentran es facil de comprobar. Ejemplo: Problema de las 8 reinas

Clase NP-Completo: Contienen los problemas mas dificiles en NP, en el sentido de que son los que esten más lejos de estar en P, indica que no se conoce solucion en tiempo polinomial. Ejemplo: Problema del viajero

Clase NP-Hard: Cuando el método de solución de un problema se puede convertir en un método de solución NP-Completo se dice que es "NP-difícil" (tan difícil como cualquier problema de NP, o talvez más difícil)
Ejemplo: Problema de satisfactibilidad booleana

Problema 8-2

¿Existe algún problema computacional que se encuentre tanto en la clase P como en la clase NP? Si es así, de un ejemplo y proponga un algoritmo para resolverlo ¿Esto significa que ambas clases de complejidad son iguales o son diferentes? ¿Su respuesta puede considerarse como una solución general para responder el problema P vs NP?

R: P y NP son 2 clases de complejidad que agrupan problemas distintos. Un problema que se encuentra dentro podría ser el ordenamiento QuickSort ya que en la mayoría de casos su complejidad es $O(n \log n)$, pero existen raros casos donde su complejidad es $O(n^2)$.

- No considero que mi respuesta anterior sea una solución general ya que existen diferentes clases NP (NP-completos y NP-Hard).

Problema 9-2

```
int main (int argc, char *argv[]){
    int k=0, x=0, i=0, j=0;
    for (k=0, x=0; k<N; k++)
        x+=k;
    printf("x=%d \n");
```

$2+N+1+N=3+2N=2N+3$

$O(n)$

x = 1643424

```
for (k=1, x=0; k<N; k*=6)
    x+=k;
    printf("x=%d \n);
```

$2+N+1+\log 6^n = 3+2N+\log 6^n$

$O(\log 6^n)$

x = 6

```
for (j=0, x=0; i<N; i+=2)
    for(j=0; j<N; j+=3)
        x+=j*j;
    printf("x=%d \n");
```

$2+N+1+N/2$
$3+2N/2$
$1+N+1+N/3$
$3+2N/2+2+2N/3$
} $5+4/3N$

$O(n)$

x = 630585904

```
for(i=N, x=0; i>0; i/=2)
    for(j=N; j>0; j--)
        x+=i*j;
    printf("x=%d \n");
}
```

$2+N+1+N/2$
$3+2N/2$
$1+N+1+N$
$2+2N$
$3+2N/2+2+2N$
} $5+4N/2$

$O(n)$

x = 0

Problema 12-2

Busqueda Binaria Interativa

Para llegar al conjunto de búsqueda de tamaño 1, hacemos "x" interaciones. Entonces

$$\frac{N}{2^{x-1}} = 1$$

$$N = 2^{x-1}$$

$$x = 1 + \log_2 n$$

Complejidad = $O(\log_2 n)$

Problema 14 -2

(62) Nivel 1
(32) Nivel 2
(16) Nivel 3
(8) Nivel 4
(4) Nivel 5
(2) Nivel 6
Nivel 7

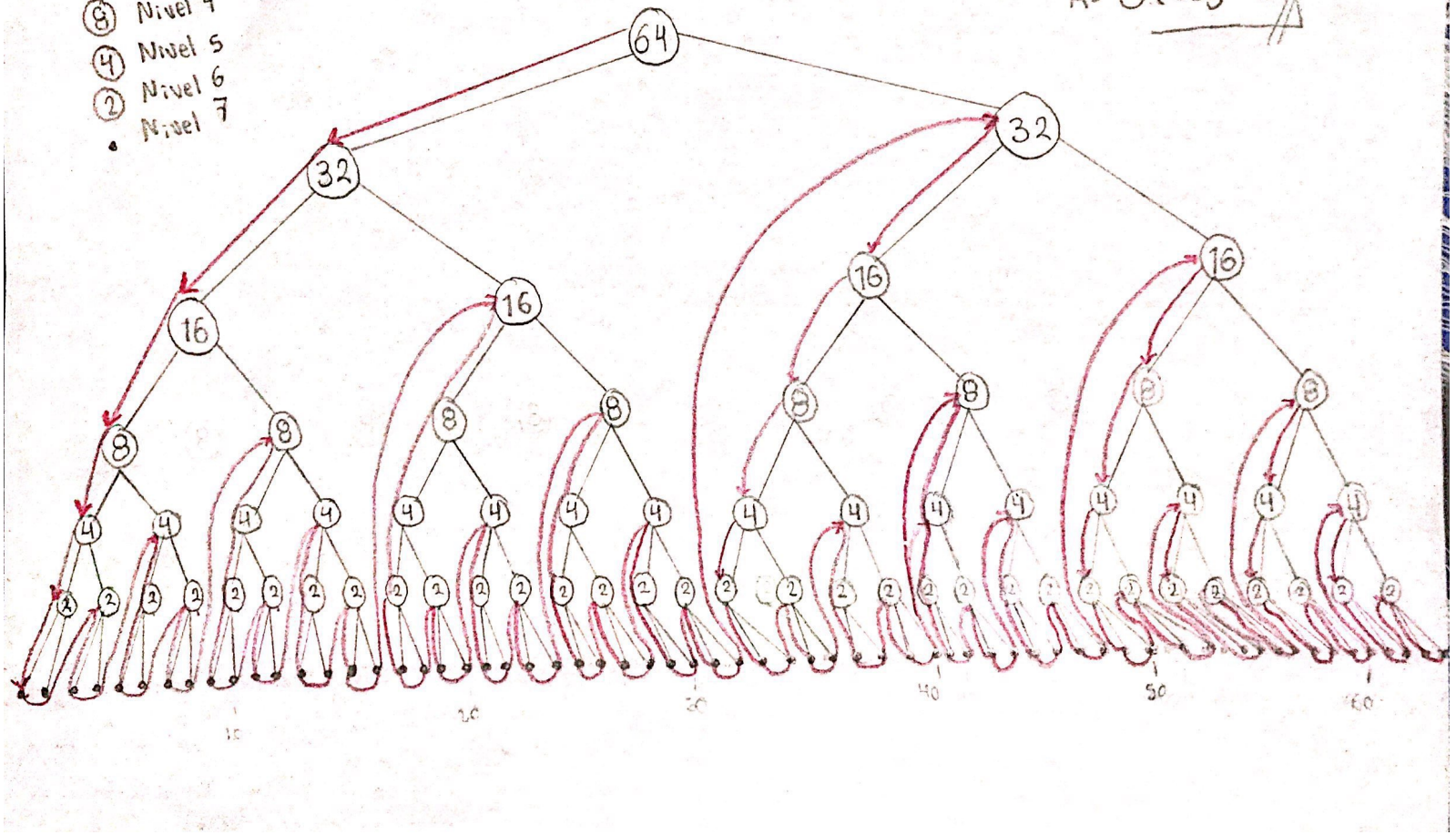
Árbol doble

Notemos que un árbol doble

Nodo = $2^A - 1$ A = altura = nivel

$A = \log_2(n+1)$

$A = O(\log n)$



Problema 14 - 2

Árbol triple

Complejidad

$n = \frac{3^A - 1}{2}$

n = nodos

A = altura o nivel

$A = \log_3(n+1)(2)$

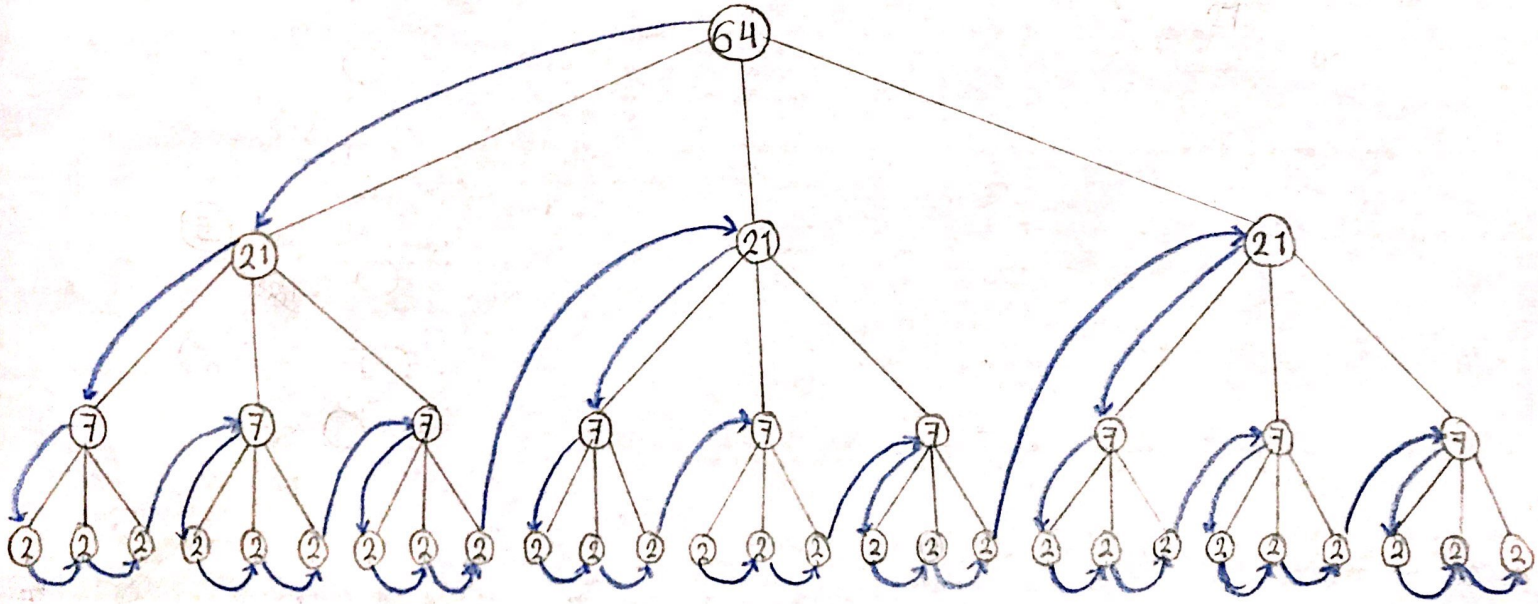
$A = O(\log n)$

(64) Nivel 1

(21) Nivel 2

(7) Nivel 3

(2) Nivel 4



Problema 16 - 2

Torres de Hanoi

¿Cuantos pasos tomaria para resolver el problema con N = 8, 16, 32 y 64 discos?

Como observamos, los pasos sigue una secuencia:

$$2^n - 1$$

Disco = 8 Pasos = $2^8 - 1 = 255$

Disco = 16 Pasos = $2^{16} - 1 = 65535$

Disco = 32 Pasos = $2^{32} - 1 = 4,294,967,295$

Disco = 64 Pasos = $2^{64} - 1 = 1.84 \times 10^{19}$

Complejidad = $O(2^n - 1)$