



Fecha: 4 de junio del 2024

Alumno: Zavaleta Guerrero Joshua Iván

Grupo: 2BM2 No. de Boleta: 2024630163

Aspas de Acero. Maquinaria de Viento. Bienvenido a la Fuerza Aérea, piloto. Ya ha pasado tiempo desde que su escuadrón ha vuelto a casa, después de una batalla sin cuartel librada en la dimensión desconocida. Después de un periodo de descansado y recuperación, ya con la mente despejada, los pilotos han recibido un adiestramiento, mientras las aeronaves han sido inspeccionadas y reparadas en los talleres de la base aérea. Su avión de caza ha quedado como nuevo; con la maquinaria afinada y funcionando a la perfección, junto con el arsenal e instrumental necesario para enfrentar cualquier reto que llegara a presentarse. Su conocimiento, experiencia y pericia serán necesarios para emprender de nuevo el vuelo para revisar el funcionamiento de su aeronave, surcar los aires y concretar exitosamente esta primera asignación, como parte de un escuadrón aliado. Su misión específica es reiniciar de nuevo su aeronave y realizar una serie de pruebas para evaluar el funcionamiento óptimo de su avión de caza. Respire profundamente y ejecute cada acción del plan. Mantenga la mente clara, enfocada y actúe con asertividad. Contará, en todo momento, con comunicación con el centro de comando y con soporte aéreo en caso de ser necesario. Las respuestas que proveerá serán sumamente valiosas para salir avante de esta emergencia. ¡Vuelen rápido y ágiles como el viento, pilotos! Cuídense unos a otros. Todos confiamos en ustedes.

Instrucciones: Por favor lea cuidadosamente cada pregunta y resuelva 4 de los siguientes 6 problemas. Cada pregunta tiene un valor de 2.5 puntos (Total de 10 puntos). ¡Mucho éxito, comencemos!

0. Turbinas y engranaje. Talleres de la base aérea, 06:00 hrs. Primeros rayos de luz por la mañana. La maquinaria del avión será revisada y su funcionamiento analizado. El sistema del avión inspecciona el encendido de turbinas, apertura de las alas, administración del combustible, ascenso y descenso del tren de aterrizaje, entre otras funcionalidades. Los datos del análisis son almacenados en 18 variables, las cuales residen en las siguientes direcciones de memoria: a (#5620), b (#4461), c (#3242), d (#2511), e (#1830), f (#0215), x (#A916), y (#B547), z (#C158), u (#DC21), v (#EB54), w (#FA29). Determine los valores de las 18 variables después de la ejecución del siguiente fragmento de código:

```
int *x=NULL, *y= NULL, *z= NULL;
int **u=NULL, **v=NULL, **w= NULL;
int a= 12345678901234567890, b= 1, c= 2, d= 3, e= 4,
f= 5;
x= &f; y= &e; z= &d; c=3
a*= *z; b+= *y - 42; c-= 13 * (*x); 4
u= &x; y= *u; *y= 3*f; 5
v= &y; w= &z; u= w, w= v, v= &x; 6
**u= a + 1; **v= b + 2; 7
**w= c + 3; 8
```

1. Fuselaje aerodinámico. Túnel de viento. Después del scaneo general, el avión es conducido a través de un túnel de viento en donde es sometido a fuertes ráfagas de viento y corrientes violentas, con la presencia de vórtices salvajes y alta turbulencia con la finalidad de verificar

337433
6 | 2024630163
 22
 44
 26
 23
 50
 21

la resistencia aerodinámica de toda la aeronave. Se han distribuido varios sensores a través del fuselaje y sus valores registrados en las celdas de una hipermatriz de 6 dimensiones $Ax Bx Cx Dx Ex F$. Codifique un código para reservar y liberar la memoria de manera correcta en un apuntador de 6 dimensiones int *****tensor. En la celda tensor[i][j][k][l][m][n] solo se guarda el valor de 1 si el sensor registra un cambio en la velocidad del viento y 0 si no hay cambio. ¿Cuántas combinaciones diferentes, en conjunto, pueden adquirir las celdas de la matriz 6-dimensional?

2. Aviones de papel. Lanzamientos recursivos. En lo que espera el análisis de la aeronave, juega a hacer aviones de papel y lanzarlos dentro de las inmediaciones del hangar. Cuenta con una torre de N hojas de papel enumeradas de 0 a $N - 1$ de forma consecutiva. Dibuja 3 franjas en el patio del hangar donde aterrizan los aviones después de volar ligeramente y realizar algunas piruetas. Codifique un algoritmo recursivo para, que particione la torre de hojas en 6 partes, invoque la función 6 veces y en cada llamada recursiva tome la sexta parte de las hojas. Entre llamadas recursivas, se realizan los pliegues necesarios para formar cada avión, los lanza y la recursión concluye cuando aterrizan aleatoriamente en las franjas, después de realizar M piruetas en el aire. Calcule la complejidad temporal de su algoritmo.
3. Ataque disruptivo. Ejercicios de combate. N aviones han despegado desde las pistas para realizar ejercicios de maniobras de vuelo y combate aéreo. Se han asignado a cada avión un número de identificación único. Si bien en el aire se han mezclan los aviones, el radar puede registrar sus posiciones y ordenar sus ubicaciones a partir de sus números de identificación guardados en un arreglo de enteros (int *aviones) mediante un algoritmo de Mergesort. Implemente iterativamente el algoritmo, a partir de la versión recursiva como guía. Calcule la complejidad de su implementación.
4. Estallido supersónico. Disparos a toda velocidad. Los aviones surcan los aires vertiginosamente a altas velocidades y todo sucede con suma rapidez. La computadora de vuelo registra todos los eventos de forma numérica en un arreglo de N enteros, los ordena, continua con los cálculos de vuelo y determinar la siguiente acción a ejecutar. Implemente ambas versiones, recursiva e iterativa, del algoritmo de Quicksort para ordenar rápidamente los números y calcule su complejidad de cada versión.
5. Onda de expansión. Laberinto de propagación. Al realizar un disparo supersónico, se forma una onda de choque que surca el aire. La onda de choque a traviesa N puntos en total, distribuidos en un árbol de N nodos de profundidad M . En cada punto (nodo) de propagación, se generan L nuevas ondas. La propagación continúa gradualmente siempre y cuando haya zonas de aire libre y concluye cuando ha impactado un objeto o la intensidad de la onda se ha atenuado y desvanecido por completo. Empleando la técnica de backtracking, codifique un algoritmo para representar la propagación recursiva de la onda.
6. ¡Ha sido extraordinario el trabajo que han realizado, pilotos! Su avión se encuentra en condiciones óptimas y listo para nuevos retos. Se les agradece mucho el valor, entrega y el esfuerzo ante esta asignación y prepararse para hacer frente a las adversidades futuras. ¡Sigan con los ánimos en alto y avanzando! Un nuevo día vendrá. ¡Qué haya cielos claros y despejados en su horizonte!

$\begin{array}{r} 337438360 \\ \hline 6 \mid 2024630163 \\ 22 \\ 44 \\ 26 \\ 23 \\ 50 \\ 21 \\ 36 \\ 03 \end{array}$

$\begin{array}{r} 337438360 \\ \hline 6 \mid 2024630163 \\ 22 \\ 44 \\ 26 \\ 23 \\ 50 \\ 21 \\ 36 \\ 03 \end{array}$

Zavaleta Guerrero
 Joshua Ivan
 2BM2
 2024630163

O. Pd. Solo hay 14 variables

int** u(#D(21), v(#EB54), w(#FA29)

int* x(#A916), y(#B547), z(#C158)

int a=3, b=1, c=2, d=3, e=4, f=5

Tarea de valores

n.	a	b	c	d	e	f	x	y	z	u	v	w
0.	3	1	2	3	4	5	null	null	null	null	null	null
1.							8f	8e	8d	8	8	8

2. 9 -37-63

recap 9 -37-63 3 4 5 8f 8e 8d null null null

3.

15 8 8f 8x 8

4.

8z 8y 8z
8x 8y

recap 9 -37-63 3 4 15 8f 8f 8d 8z 8x 8y

5. 0 10 -35

6. -60

9	-37	-63	10	4-60	8f	8f	8d	8z	8x	8y
---	-----	-----	----	------	----	----	----	----	----	----

$a = 9$	$d = 10$	$x = \#0215 (f)$	$u = \#C158 (z)$
$b = -37$	$e = 4$	$y = \#0215 (f)$	$v = \#A916 (x)$
$c = -63$	$f = -60$	$z = \#2511 (d)$	$w = \#B547 (y)$

Ramírez Guerrero Joshua Iván 2B M2
O.
Procedimiento 2024630(6)

$$\begin{array}{r} 13 \\ \times 5 \\ \hline 65 \end{array}$$

$$\begin{array}{r} 42 \\ \times 7 \\ \hline 294 \end{array}$$

1. $x = 8f, y = 8e, z = 8d$

2. $a = *x + z \Rightarrow a = 3 * 3 = 9 = 3g = 9$

$$b = *y - 42 \Rightarrow b = 1 + 4 - 42 = 5 - 42 = -37 \Rightarrow b = -37$$

$$c = 13 * (*x) \Rightarrow c = 2 - [13 * 5] = 2 - 65 = -63 \Rightarrow c = -63$$

3. $u = 8x$ $y = *u;$ $*y = 3 * f$
 ~~$\times 3$~~
 $\hookrightarrow y = x$ $f = 3 * 5 = 15 \Rightarrow f = 15$
 $y = 8d$

4. $v = 8y, w = 8z, u = w, w = v, v = 8x$

5. $*u = a + 1$ $*v = b + 2;$
 $d = a + 1$ $f = b + 2$
 $d = a + 1 = 10$ $f = -37 + 2$
 $f = -35$

6. $*w = c + 3$

$$f = c + 3$$

$$f = -63 + 3$$

$$f = -60$$

1. Código

```
#include <stdio.h>
#include <stdlib.h>

void hyp6 (int****hyp, int ii, int jj, int kk, int ll, int mm, int nn)
{
    int i, j, k, l, m, n;

    *hyp = (int****) malloc (ii * sizeof(int****));
    for (i=0; i<ii; i++)
    {
        *(hyp+i) = (int****) malloc (jj * sizeof(int****));
        for (j=0; j<jj; j++)
        {
            *(*(hyp+i)+j) = (int****) malloc (kk * sizeof(int****));
            for (k=0; k<kk; k++)
            {
                *(*(hyp+i)+j)+k = (int****) malloc (ll * sizeof(int****));
                for (l=0; l<ll; l++)
                {
                    *(*(hyp+i)+j)+k+l = (int****) malloc (mm * sizeof(int));
                    for (m=0; m<mm; m++)
                        *(*(hyp+i)+j)+k+l+m = (int****) malloc (nn * sizeof(int));
                }
            }
        }
    }
}

=>
```

Guerrero Joshua Turn 21M2 2024630(63)

```
void Freehyp6(int ****hyp, int ii, int jj, int kk, int ll, int mm)
{
    int i, j, k, l, m;
    for(i=0; i < ii; i++)
    {
        for(j=0; j < jj; j++)
        {
            for(k=0; k < kk; k++)
            {
                for(l=0; l < ll; l++)
                {
                    for(m=0; m < mm; m++)
                    {
                        free(*(*(*(*hyp+i)+j)+k)+l)+m);
                        free(*(*(*(*hyp+i)+j)+k)+l));
                        free(*(*(*hyp+i)+j)+k));
                        free(*(*hyp+i)+j));
                    }
                }
            }
        }
    }
    free(*hyp);
    *hyp = NULL;
}
```

=>

```

int main()
{
    int *****hyp;
    int i, j, k, l, m, n;
    scanf("%d%d%d%d%d%d", &i, &j, &k, &l, &m, &n);
    hyp6(&hyp, i, j, k, l, m, n);
    fillhyp6(&hyp, i, j, k, l, m, n); } funciones conceptuales
    printhyp6(hyp, i, j, k, l, m, n); } no codificadas en
    free(&hyp, i, j, k, l, m); } papel
    return 0;
}

```

2. Combinaciones

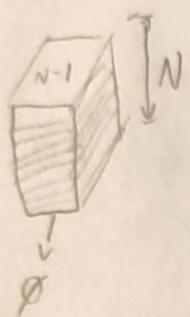
Dimensiones: $A \times B \times C \times D \times E \times F \Rightarrow$ Total de elementos (T)

Estatos: $\gamma, \emptyset \Rightarrow 2$ estatós

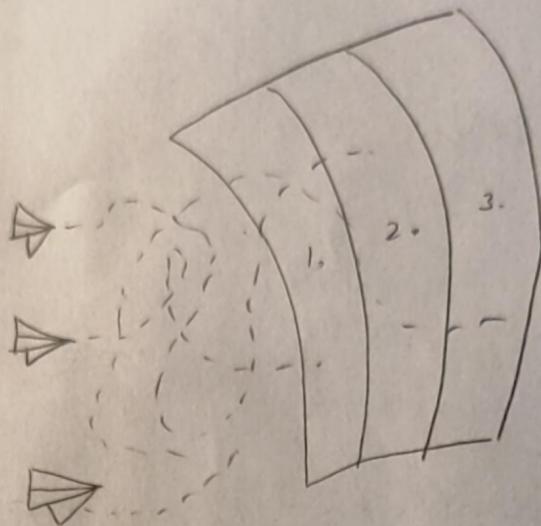
Combinaciones: $\frac{T(A \times B \times C \times D \times E \times F)}{2}$

$$= \frac{1}{2} \cdot T = \frac{1}{2} \cdot (A \times B \times C \times D \times E \times F)$$

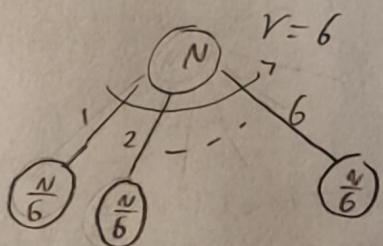
2.



1.



Arbol base 6



Código:

```
#include<csdio.h>
#include<csfdis.h>
```

```
typedef struct Avion {
    int num;
    int piroetas;
    int aterrizaje;
} *Avion, Plane;
```

-Scontinuación del código sig. página

Eduardo Guerrero Torreón Ruiz,
213712 2024630(63)

```

void vueloAvion(Avion av) {
    int x;
    #ifdef NDEBUG
        srand(time(NULL));
    #endif
    x = rand() % 5;
    if(x > 2) {
        av->piruetas++;
        vueloAvion(av);
    } else
        av->aterrizaje = x + 1 - 6;
}

void lanzamientoAvion(Avion arr, int n)
{
    int division, residuo; -- O(1)

    division = n / 6; ----- O(1)
    residuo = n % 6; -- O(1)

    if(division) { -- O(1)
        lanzamientoAvion(arr, division);
        lanzamientoAvion(arr + division, division);
        lanzamientoAvion(arr + 2 * division, division);
        lanzamientoAvion(arr + 3 * division, division);
        lanzamientoAvion(arr + 4 * division, division);
        lanzamientoAvion(arr + 5 * division, division + residuo));
    }
    else {
        if(residuo == 0) -- O(1)
            return; -- O(1)
        else {
            vueloAvion(arr); -- O(1)
            lanzamientoAvion(arr + 1, n - 1); -- O(n - 1)
        }
    }
}

```

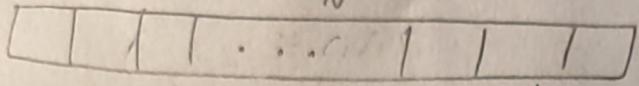
complejidad = $O(1) + O(5)$
 $= 8O(1) + 5O(1)$
 $\leq 2O(1)$
 $= O(1) \rightarrow$ complejidad
de vueloAvion

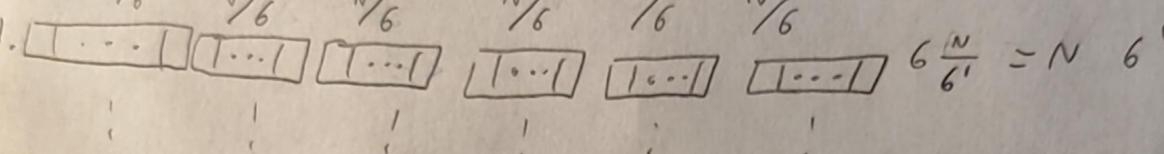
caso base 6

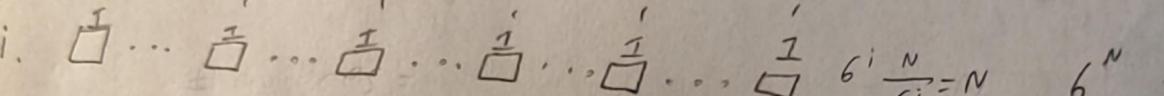
2. Zúñiga Guerrero Joshua Iván 2DM2 2024630163

Complejidad de función Lanzamiento Avión

árbol base 6

O.  $\frac{N}{6^0} = N$ 6^0

I.  $6 \frac{N}{6^1} = N$ 6^1

i.  $6^i \frac{N}{6^i} = N$ 6^i

$$\frac{n}{6^i} = 1 \Rightarrow 6^i = n \quad \text{lanzamientoAvion}$$

$$\log_6 n = i \quad T(n) = n + n + \dots + n \\ = i n \\ \rightarrow = n \log_6 n$$

∴ Complejidad de lanzamientoAvion = $O(n \log_6 n)$ //

3. Código

```

void mSortProcess(int *arr, int inf, int sup) {
    int i, j, k, l = sup - inf, middle;
    int *aux;
    aux = crearArreglo(l+1);
    fillZeroArr(aux, l+1);
    i = inf;
    if (l == 1)
        j = inf + 1;
    else
        if (l % 2 == 0)
            j = inf + l/2;
        else
            j = inf + (l/2) + 1;
    middle = j;
    k = 0;
    while (k <= l) O(l)
    {
        if (i < middle && j <= sup)
        {
            if (*arr+i) >= *(arr+j)
            {
                *(aux+k) = *(arr+j);
                k++;
                j++;
            }
            else
            {
                if (i < middle)
                {
                    *(aux+k) = *(arr+i);
                    i++;
                    k++;
                }
            }
        }
    }
}

```

para merge | para recursive {j = inf + l/2;
iterativo }

→ else
 {
 *(aux+k) = *(arr+i);
 j++;
 k++;
 }
 }
 for(i = inf; k=0; k<=l; k++, it++) O(l)
 *(arr+i) = *(aux + k);
} // termina función mSortProcess, función encargada de ordenar
// los elementos de un arreglo con límites definidos
L > T(n) = O(n) + O(n) = O(2n) => O(n) //
// mergeSort recursivo de Guía

```

void mergesort_R(int *arr, int inf, int sup) - O(n log2n)
{
    int x;
    if(sup - inf >= 1)
    {
        x = (sup - inf)/2;
        if(sup - inf >= 2)
        {
            mergesort_R(arr, inf, inf+x-1); error de escritura
            mergesort_R(arr, inf+x, sup);
        }
        mSortProcess(arr, inf, sup);
    }
} // fin de la función

```

Zavaleta Guerrero Joshua Iván 2B M2 2024630163

continuación código 3.

int powJ(unsigned int base, unsigned int exponente) $O(n)$

{ if(exponente == 0)
 return 1;

else return base * powJ(base, exponente - 1);
}

int detectPow(int n) // detecta la potencia de dos que ve después
del número insertado

{ int x = 2;

while($\%x \neq 0 \text{ || } \%x == 1$) { $O(\log_2 N)$

if($\%x == 1 \text{ & } \%0x == 0$)
 return x;

else

$x *= 2;$

}

return x;

} // fin función

int detectExp2(int n) \rightarrow tiene que ser una potencia de 2.

{

int count = 0; \rightarrow otro error de escritura

int count = 0; \rightarrow otro error de escritura

if(n == 1)

return 1;

while(x != n) $O(\log_2 N)$

{

$x *= 2;$

count++;

return count;

} // función que detecta el exponente de la potencia
de 2 ingresada.

```
int sumNivelesB2(int n) //sumar de las potencias de 2 que  
{  
    rellenaran el numero total de arreglos  
    que tendra cada arbol (mergesort)  
    int x, i, count;  
    x = defectExp2(n); O(log, N)  
for  
    x--;  
    count = 0;  
    for(i = x; i >= 0; i--) O(N)  
        count += powJ(2, i);  
    return count;  
} O(N + log, N) => O(N)
```

~~void~~
mergesort iterativo sig. haga

3. Continuación de código

void mergesort_I(int *arr, int n)

{

int limn, i, j, x, count;

int *lims, *limi, *dif;

limn = sumNiveles_B2(detectPow2(n)); $O(\log_2 n)$

lims = crearArreglo(limn);

limi = crearArreglo(limn);

dif = crearArreglo(limn);

FillZeroArr(lims, limn);

FillZeroArr(limi, limn);

FillZeroArr(dif, limn);

for(i = 0; i < limn; i++){ $O(\log_2 n)$

if(i == 0)

{

*limi = 0;

*lims = n-1;

*dif = *lims - *limi;

count = 1;

}

else

{

*(limi + i) = *(limi + i - count);

if(*(dif + i - count) % 2 != 0)

(lims + i) = ((dif + i - count)/2) + *(limi + i);

else

(lims + i) = ((dif + i - count)/2) - 1 + *(limi + i);

*(dif + i) = *(lims + i) - *(limi + i);

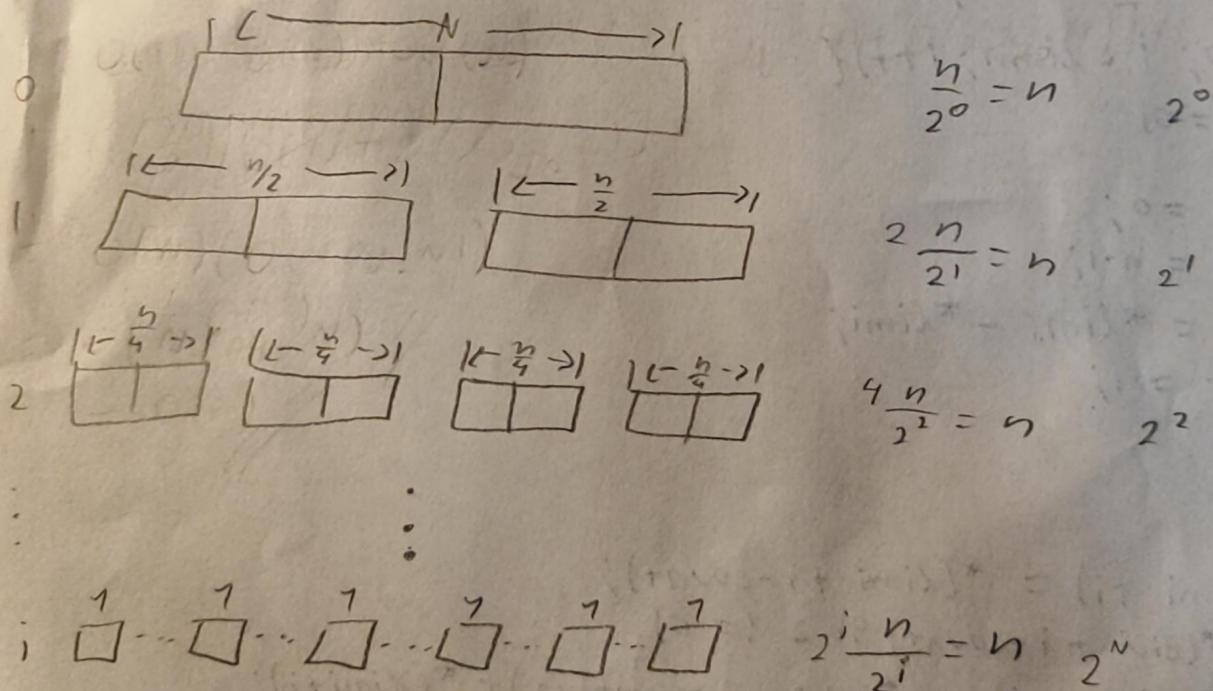
count ++;

i ++;

-> continuación de código

$\rightarrow *(\text{limi} + i) = *(\text{lims} + i - 1) + 1;$
 $*(\text{lims} + i) = *(\text{lims} + i - \text{count});$
 $*(\text{dijs} + i) = *(\text{lims} + i) - *(\text{limi} + i);$
 }
 }
 for(j = limn - 1; j >= 0; j--) $O(\log_2 n)$
 if($*(\text{limit} + j) \neq *(\text{lims} + j)$)
 mSortProcess(arr, $*(\text{limi} + j), *(\text{lims} + j)$); $O(n)$
 liberarArreglo(&limi);
 liberarArreglo(&lims);
 liberarArreglo(&dijs);
 }

Complejidad:



$$\frac{n}{2^i} = 1$$

$$2^i = n$$

$$\log_2 n = i$$

$$\begin{aligned}
 T(n) &= O(n) + O(n) + \dots + O(n) \\
 &= O(in) \\
 &= O(n \log_2 n) //
 \end{aligned}$$

4. Quicksort

```
void swapInt(int *a1, int *a2)
```

```
{
```

```
    int aux;
```

```
    aux = *a1;
```

```
    *a1 = *a2;
```

```
    *a2 = aux;
```

```
} O(1)
```

valor inicial: $n - 1$

```
Void quickSort_R(int *arr, int pivotc)
```

```
{
```

```
    int check = arr - 1;
```

```
    int i, count = -1;
```

```
    for(i=0; i <= pivotc; it++) O(n)
```

```
        if(*(arr+i) <= *(arr + pivotc))
```

```
{
```

```
            check++;
```

```
            count++;
```

```
            if(arr+i > check)
```

```
                swapInt(arr+i, check);
```

```
}
```

```
    if(count > 1)
```

```
        quickSort_R(arr, count - 1); O( $\frac{n}{2} - 1$ )
```

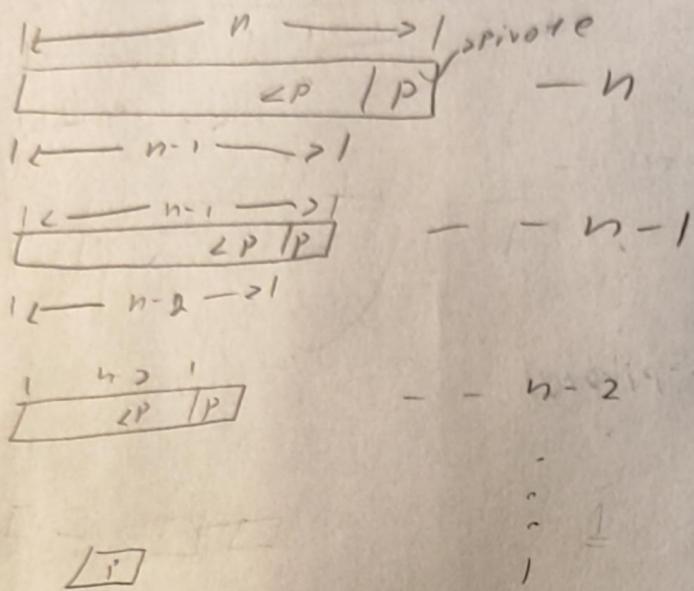
```
    if(pivotc - count > 1)
```

```
        quickSort_R(++check, pivotc - count - 1); O( $\frac{n}{2} - 1$ )
```

```
} // Fin de la función
```

Calculo de complejidad de quicksort-R

Peor de los casos



debido al for dentro de la función

$$T(n) = O(n) + O(n-1) + O(n-2) + \dots + O(1)$$

$$\approx O(n^2) //$$

Nota

implementando una mejor selección del pivote se puede llegar a una complejidad de $O(n \log n)$

similar a mergesort

4. Quicksort Iterativo

int detectExp2(int n) $O(\log_2 n)$ {
 int count = 0, x = 1;

if (n == 1)

return 1;

while (x != n)

{

x *= 2;

count++;

}

return count;

}

int detectPow2(int n) $O(\log_2 n)$ {
 int x = 2; while ($\gamma_x \neq \emptyset$ || $\gamma_x = \{1\}$) { if ($\gamma_x = \{1\}$ & $n \% x = \emptyset$)

return x;

else

x *= 2;

}

return x;

}

Void quicksort_I(int *arr, int n)

Eduardo Guerrero Joshua Duean
2BML 2024630163

{

int i, j, m, count, aux;
int *lims, *limi;

if(n <= 1)
 return;

m = detectExp2(detectPow2(n)); $O([\log_2 n]^2)$

lims = crearArreglo(m);

limi = crearArreglo(m);

*limi = \emptyset ;

*lims = n - 1;

j = 0;

while(j ≤ -1) $O(N)$

{ count = *(limit+j) - 1;

for(i = *(limit+j); i \leq *(lims+j); i++) $O(N)$

{ if(*arr+i) \leq *(arr+*(lims+j)))

{ count++;

if(i > count)

swapInt(arr+i, arr+count); $O(1)$

}

if((count-1) - *(limit+j) ≥ 1 || *(lims+j) - (count-1) ≥ 1) $O(1)$

{

aux = *(lims+j);

*lims+j = count - 1;

*limi+j+1 = count + 1;

*lims+j+1 = aux;

j++

}

else ->

Zavalete Guerrero Joshua Ivan 2B M2 2024610(63)

→ if ((count - 1) * (limit + j) >= 1)
 *(limst + j) = count - 1;
else
 if (*limst + j) - (count + i) >= 1
 *(limit + j) = count + 1;
 else
 j--;

} // fin de ~~quicksort_I~~ while

} // fin de ~~quicksort_I~~

calculo de complejidad

$$T = O(\cancel{\log N}) + O(N)[O(N)] = O(N^2) //$$

Zavaleta Guerrero Joshua Ivan 20112 2024630163

5. Código

```
int comportamientoOnda(int val, int base)
{
    int x;
    #ifdef NDEBUG
        srand(time(NULL));
    #endif
    x = rand() % 3; // 1/3 de probabilidad de que choque con algo
    if(x > 0)
        return val / base;
    else return 0;
}
```

```
void arbolOnda(int base, int lvl, int Lmax, int val)
{
    int i, j, x;
    if(lvl == 0)
    {
        printf("Nivel: 0, Nodo: 0, Valor: %d\n", val);
        arbolOnda(base, ++lvl, Lmax, val);
    }
    if(lvl <= max)
    {
        for(i = 0; i < base; i++)
        {
            x = comportamientoOnda(val, base);
            for(j = 0; j < lvl; j++)
                putchar('\t');
            printf("Nivel: %d, Nodo: %d, Valor: %d\n", lvl, i, x);
            if(x != 0)
                arbolOnda(base, lvl + 1, Lmax, x);
        }
    }
}
```