


c

 Copy code

```
#include <stdio.h>

int findMax(int arr[], int size) {
    int max = arr[0];
    for (int i = 1; i < size; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}

int main() {
    int arr[] = {10, 5, 8, 15, 3};
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("El elemento máximo del arreglo es: %d\n", findMax(arr, size));
    return 0;
}
```

$$1 + \sum_{i=1}^n 1$$

$$O(n)$$

```

int binarySearch(int arr[], int left, int right, int target) {
    if (right >= left) { →  $O(1)$ 
        int mid = left + (right - left) / 2; →  $O(1)$ 

        if (arr[mid] == target) {  $O(1)$ 
            return mid;
        }

        if (arr[mid] > target) {
            return binarySearch(arr, left, mid - 1, target);
        }

        return binarySearch(arr, mid + 1, right, target);
    }

    return -1;
}

```

for $n (0, k)$

while

i
 h

k

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$T\left(\frac{n}{4}\right) + 1 + 1$$

$$T\left(\frac{n}{8}\right) + 3$$

$$T\left(\frac{n}{16}\right) + 4$$

$$T\left(\frac{n}{2^k}\right) + k$$

$$T\left(\frac{n}{2^k}\right) + k \quad k = \log(n)$$

$$T\left(\frac{n}{2^{\log(n)}}\right) + \log(n)$$

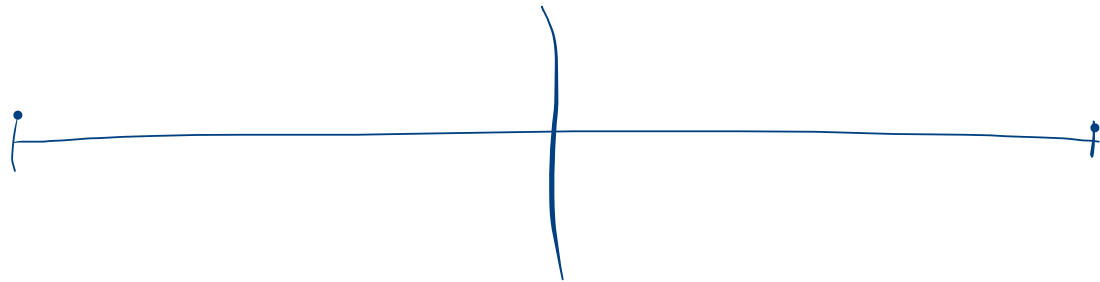
$$\sum_{i=0}^n \frac{n}{2^k}$$

0

$$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \quad / \log$$

$$k = \log_2 n$$

$$\cancel{T(1)} + T(\log_2 n)$$



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

$$n = 2^k / \log_2$$

$$\log_2 n = k$$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$T(n) = (T\left(\frac{n}{4}\right) + 1) + 1 \Rightarrow T\left(\frac{n}{4}\right) + 2$$

$$T\left(\frac{n}{8}\right) + 3 \Rightarrow T\left(\frac{n}{2^k}\right) + k$$

$$\hookrightarrow T(1) + T(\log_2 n)$$

$$O(\log_2 n) \text{ i.}$$

```

void selectionSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < size; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[minIndex];
        arr[minIndex] = temp;
    }
}

```

$$\sum_{i=0}^n \sum_{j=i+1}^n 1$$

$$\sum_{i=0}^n n = n^2 \Rightarrow O(n^2)$$

```
int fibonacci(int n) {
```

```
    if (n <= 1) {
```

```
        return n;
```

```
    } else {
```

```
        return fibonacci(n - 1) + fibonacci(n - 2);
```

```
    }
```

```
}
```

$\rightarrow O(1)$

n

n

$$F_i(x) = \begin{cases} F_i(x-1) + F_i(x-2); & n > 1 \\ x & , n \leq 1 \end{cases}$$

$$T(n) = \begin{cases} T(n-1) + T(n-2) + 1; & n > 1 \\ 1 & ; n \leq 1 \end{cases}$$

$$T(n-1) + T(n-2)$$

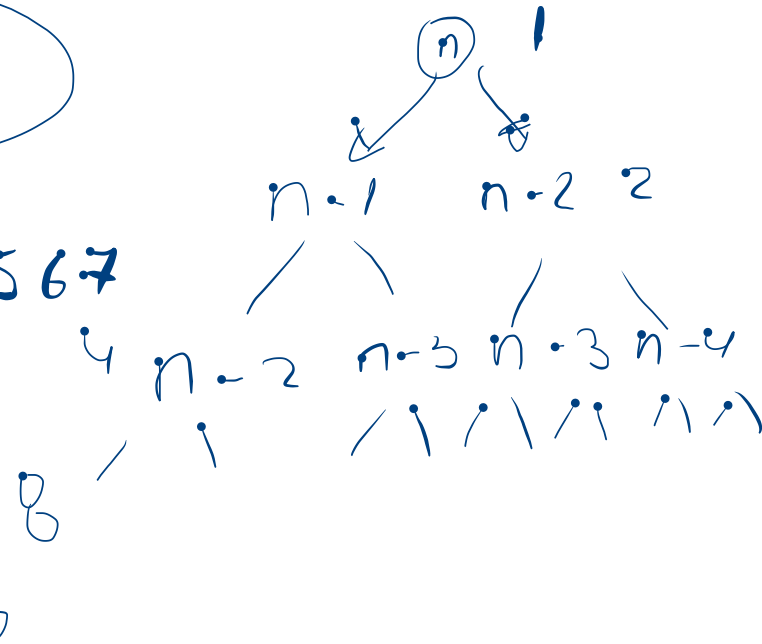
$$[T(n-2) + T(n-3) + 1] + [T(n-3) + T(n-4) + 1] + 1$$

$$T(n-2) + 2T(n-3) + T(n-4) + 3$$

$$f(n) \Rightarrow \underbrace{f(n-1)} + \underbrace{f(n-2)}$$

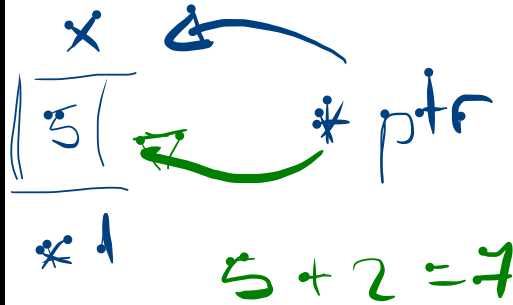
15
14 13 2^n

1534567



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

```
int main() {  
    int x = 5;  
    int *ptr = &x;  
    *ptr += 2;  
    printf("%d", *ptr);  
    return 0;  
}
```




```
void imprimirRecursivo(int *ptr, int n) {
    if (n > 0) {
        printf("%d ", *ptr);
        (*ptr)++;
        imprimirRecursivo(ptr, n - 1);
    }
}
```

Suponiendo que se llama a la función `imprimirRecursivo(&num, 5);` y `num` es inicialmente igual a 1, ¿cuál será la salida en la consola?

```
int main() {
    int num = 1;
}
```

ptr
→ num
(1)
2
3
4
5
6

n
5
4
3
2
1
0

1
2
3
4
5

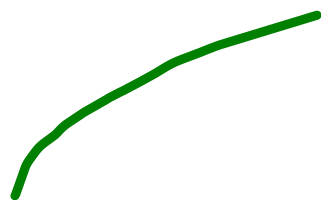
`int *num;`

`int num2;
&num2`


`int b = *num`

`↳ int nido`

`int **n`



c

 Copy code

```
void imprimirInversoRecursivo(int *arr, int n) {  
    if (n > 0) {  
        imprimirInversoRecursivo(arr + 1, n - 1);  
        printf("%d ", *arr);  
    }  
}
```

Suponiendo que se llama a la función `int numeros[] = {1, 2, 3, 4, 5};`
`imprimirInversoRecursivo(numeros, 5);`, ¿cuál será la salida en la consola?

{ 1, 2, 3, 4, 5 }

* indice n

* i + 1 5

↘ 4

i + 2

↘ 3

i + 3

↘ 2

i + 4

↘ 1

i + 5

↘ 0

5

4

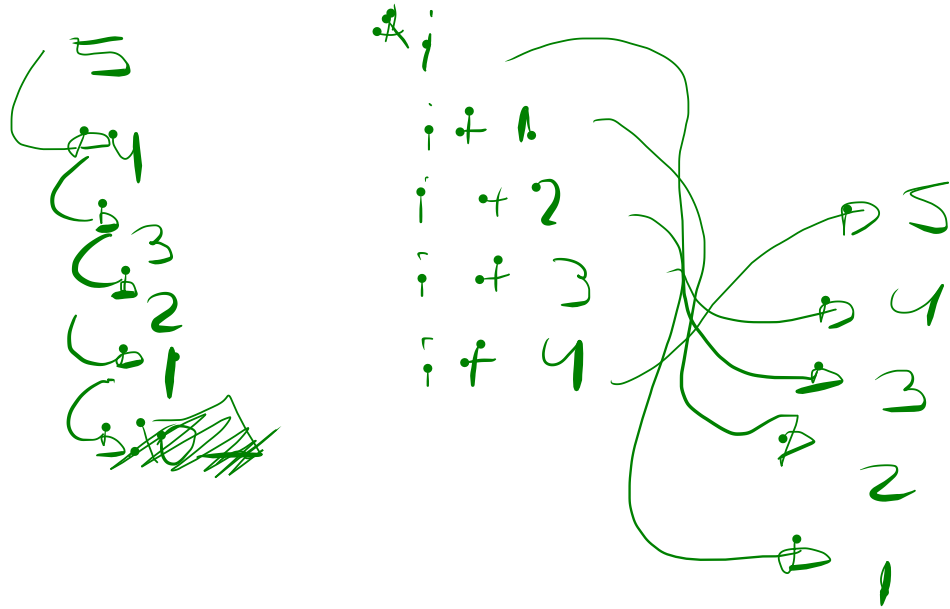
3

2

1

{1, 2, 3, 4, 5}

len(arr) = 5



```
int i = 1, j = 1;
```

```
while (i <= 5) {
```

```
    while (j <= i) {
```

```
        printf("%d ", j);
```

```
        j++;
```

```
    }
```

```
    i++;
```

```
    j = 1;
```

```
    printf("\n");
```

```
}
```

```
for (i = 5; i >= 1; i--) {
```

```
    int k = 1;
```

```
    while (k <= i) {
```

```
        printf("%d ", k);
```

```
        k++;
```

```
    }
```

```
    printf("\n");
```

```
}
```

i j

1	1
1	2
2	1
2	2
2	3
3	1
3	2
3	3
3	4

i j

4	1
4	2
4	3
4	4
4	5
5	1
5	2
5	3
5	4
5	5
5	6
6	1

1				
1	2			
1	2	3		
1	2	3	4	
1	2	3	4	5

```

int i = 1, j = 1;

while (i <= 5) {
    while (j <= i) {
        printf("%d ", j);
        j++;
    }
    i++;
    j = 1;
    printf("\n");
}

for (i = 5; i >= 1; i--) {
    int k = 1;
    while (k <= i) {
        printf("%d ", k);
        k++;
    }
    printf("\n");
}

```

```

1
5 1
5 2
5 3
5 4
5 5
5 6
4 1
4 2
4 3
4 4
4 5
3 1
3 2
3 3
3 4
2 1
2 2
2 3
1 1
1 2

```

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

```

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2
1

```

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

```
void cambiarPosicion(char *str, int pos1, int pos2) {
    char temp = str[pos1];
    str[pos1] = str[pos2];
    str[pos2] = temp;
}
```

```
void permutarString(char *str, int l, int r) {
    if (l == r) {
        printf("%s\n", str);
    } else {
        for (int i = l; i <= r; i++) {
            cambiarPosicion(str, l, i);
            permutarString(str, l + 1, r);
            cambiarPosicion(str, l, i); // Restaurar el orden original para
        }
    }
}
```

```
int main() {
    char palabra[] = "ABC";
    int longitud = strlen(palabra);
    permutarString(palabra, 0, longitud - 1);
    return 0;
}
```

BAC

ABC

ACB

CAB

"ABC" len = 3
permutar (palabra
0, 2)

i	r	palabra
0	2	ABC ABC
0	2	BAC

1	2	BAC
1	2	BAC
		ABC

C:\User

ABC

ACB

BAC

BCA

CBA

CAB

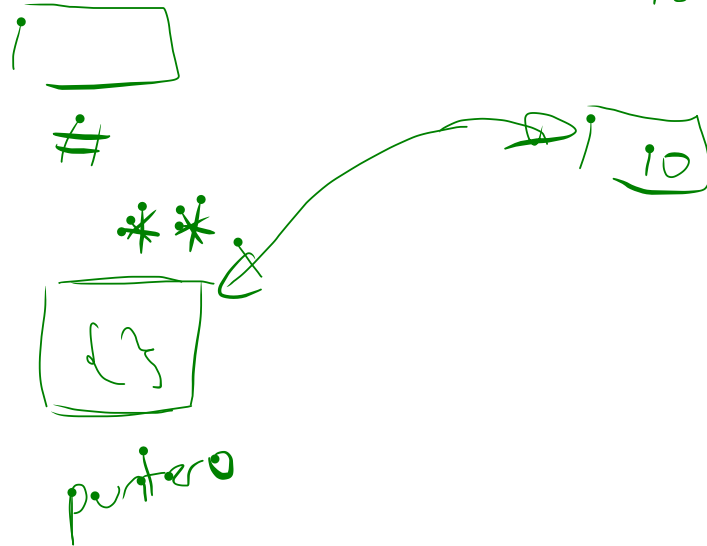
```
#include <stdio.h>
```

```
void modificarPuntero(int **ptr) {  
    int num = 10;  
    *ptr = &num;  
}
```

```
int main() {  
    int *puntero = NULL;  
    modificarPuntero(&puntero);  
    printf("%d\n", *puntero);  
    return 0;  
}
```

$\text{puntero} = \text{NULL} \quad \leadsto \quad f(x) \Rightarrow$

$\text{num} = 10$



A B C
0 1 2

C B A
0 1 2

18. ¿Cuál es la salida del siguiente programa en C?

C

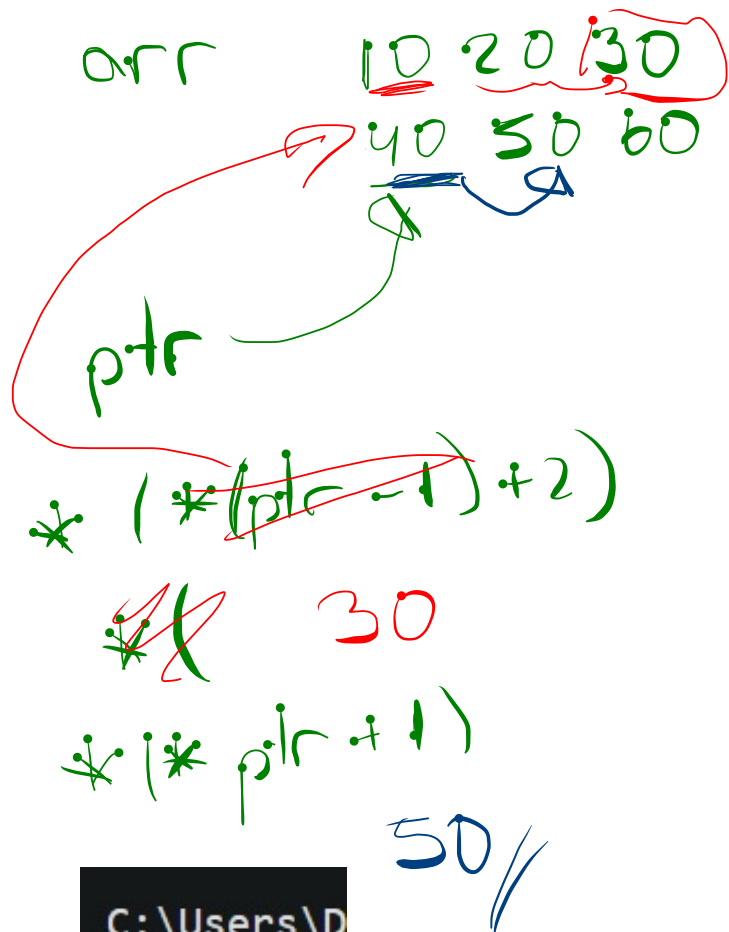
Copy code

```
#include <stdio.h>

int main() {
    int arr[2][3] = {{10, 20, 30}, {40, 50, 60}};
    int (*ptr)[3] = arr + 1;
    printf("%d %d", (*(ptr - 1) + 2), *(*ptr + 1));
    return 0;
}
```

- a) 20 50
- b) 30 50
- c) 60 20
- d) 60 50

Nota: "(*ptr)[3]" significa que "ptr" es un puntero a un arreglo de 3 enteros.



C:\Users\D
30 50
C:\Users\D

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

```
void reordenarPunteros(int *ptr1, int *ptr2, int *ptr3) {
```

```
    int *temp;
```

```
    temp = ptr1;
```

```
    ptr1 = ptr3;
```

```
    ptr3 = ptr2;
```

```
    ptr2 = temp;
```

```
}
```

```
int main() {
```

```
    int num1 = 1, num2 = 2, num3 = 3;
```

```
    int *p1 = &num1, *p2 = &num2, *p3 = &num3;
```

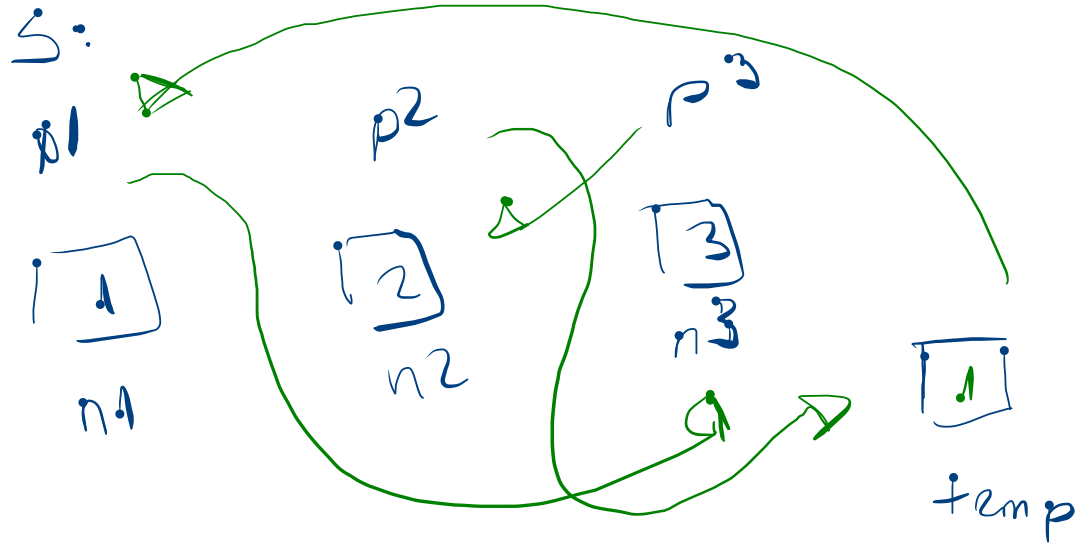
```
    reordenarPunteros(p1, p2, p3);
```

```
    printf("%d %d %d\n", *p1, *p2, *p3);
```

```
    return 0;
```

```
}
```

1 2 3



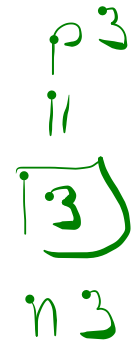
3 1 2
Solo local!!!

C:\Users\
1 2 3

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

```
void reordenarPunteros(int **ptr1, int **ptr2, int **ptr3) {  
    int *temp;  
    temp = *ptr1;  
    *ptr1 = *ptr2;  
    *ptr2 = *ptr3;  
    *ptr3 = temp;  
}
```

```
int main() {  
    int num1 = 1, num2 = 2, num3 = 3;  
    int *p1 = &num1, *p2 = &num2, *p3 = &num3;  
  
    reordenarPunteros(&p1, &p2, &p3);  
  
    printf("%d %d %d\n", *p1, *p2, *p3);  
  
    return 0;  
}
```



```
C:\Users\O  
2 3 1
```

Enunciado:

Escribe dos funciones en lenguaje C para resolver el siguiente problema: dada una lista de números enteros, encontrar el máximo valor presente en la lista. Debes proporcionar una solución iterativa y otra solución recursiva.

1. Escribe una función llamada ``maximoIterativo`` que reciba como parámetro un arreglo de enteros y su tamaño, y retorne el máximo valor encontrado en el arreglo utilizando una solución iterativa.
2. Escribe una función llamada ``maximoRecursivo`` que reciba como parámetro un arreglo de enteros y su tamaño, y retorne el máximo valor encontrado en el arreglo utilizando una solución recursiva.

Después de escribir las funciones, responde a las siguientes preguntas:

- a) ¿Cuál es la complejidad algorítmica de la función ``maximoIterativo`` en términos de tiempo? Explica tu respuesta.
- b) ¿Cuál es la complejidad algorítmica de la función ``maximoRecursivo`` en términos de tiempo? Explica tu respuesta.

```

int maximoIterativo(int arr[], int size){
    int max = arr[0];
    int i;
    for(i = 0; i < size; i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }
    return max;
}

int maximoRecurso(int *arr, int size){
    if (size == 0){
        return 0;
    } else if (size == 1){
        return arr[0];
    } else{
        int max = maximoRecurso(arr + 1, size - 1);
        if(arr[0] > max){
            return arr[0];
        }
        return max;
    }
}

```

$O(n)$

$$T(n) = \begin{cases} 1 & \text{si } n \leq 1 \\ T(n-1) + 1 & \text{si } n > 1 \end{cases}$$

$$T(n-2) + 1 + 1$$

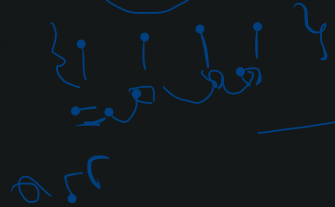
$$T(n-3) + 1 + 1 + 1$$

$$T(n-4) + 4$$

$$T(n-k) + k$$

$$n - k = 1$$

$$n = k$$



$$T(n-2) + 1 + 1$$

$$T(n-3) + 1 + 1 + 1$$

$$T(n-4) + 4$$

$$T(n-k) + k$$

$$n = k$$

$$\Rightarrow T(\cancel{n-n}) + n$$

$$T(0)$$

$$T(1) + n$$

$$\therefore \boxed{T(n) = O(n)}$$

Conclusión

Temporal: Ambos tienen la misma complejidad

pero...

Espacial: El más eficiente en memoria
(Memoria) es el algoritmo iterativo

