Pruebas

Éste documento muestra los cálculos para 6 arreglos diferentes con los que éste algoritmo (en su forma recursiva) funciona, y el propósito del documento es respaldar los resultados arrojados por el código compilado en RISC-V, mostrando que el proceso es el mismo descrito por el código de alto nivel.

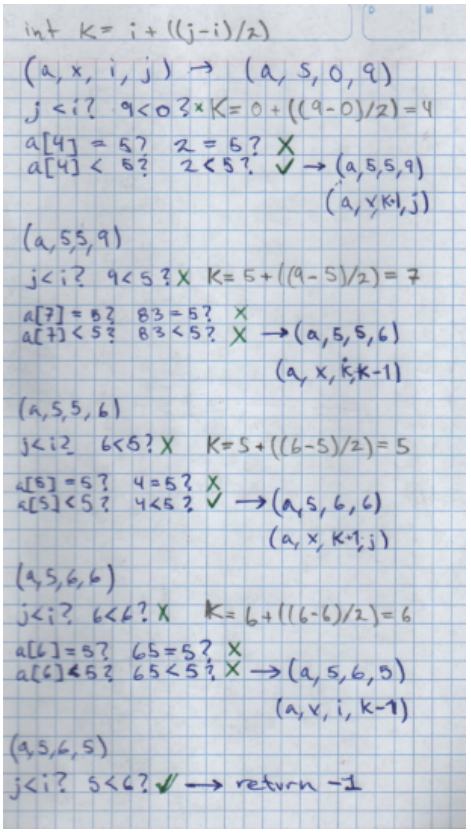
Contenido :
Comparación de resultados de una versión comentada del código original (éste código modificado está hasta abajo), con cálculos en hojas y un diagrama por arreglo.

El proceso para comprobar que funciona correctamente es el siguiente :

Ingresar valores de los ejemplos, en el código compilado y ejecutar el código analizando que los cálculos son los mismos por cada arreglo.

1 2 3 4 5 6 7 8 9 10

a = {-31, 0, 1, 2, 2, 4, 65, 83, 99, 782} - original



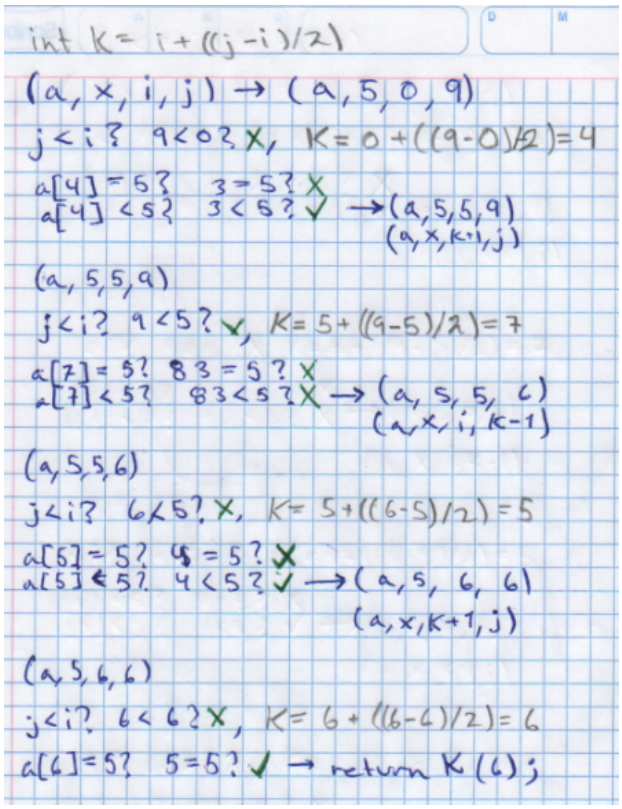
Objetivo (x) = 5, Número de elementos de arreglo = 10

RESULTADOS :
no está 5
resultado esperado : índice = -1

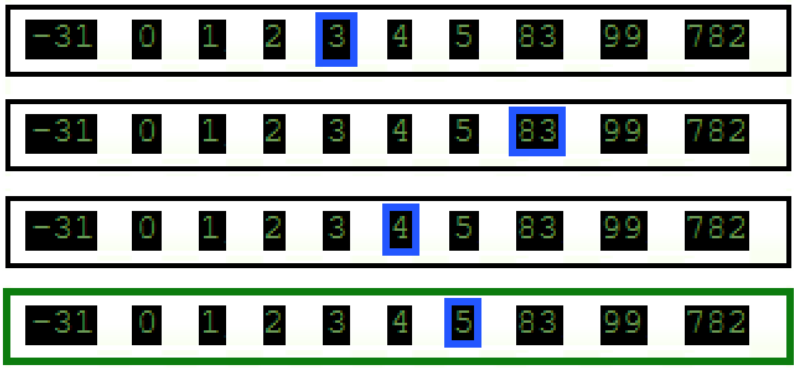
```
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# ./elef
{-31, 0, 1, 2, 2, 4, 65, 83, 99, 782};
 (0, 9)  k : 4

 (k+1,j) : (5, 9) porque 2 < 5

 (5, 9)  k : 7

 (i,k-1) : (5, 6) porque 83 !< 5

 (5, 6)  k : 5

 (k+1,j) : (6, 6) porque 4 < 5

 (6, 6)  k : 6

 (i,k-1) : (6, 5) porque 65 !< 5


 5 < 6 entonces 5 no está en el arreglo
el valor del índice es -1
```

1 2 3 4 5 6 7 8 9 10

a = {-31, 0, 1, 2, 3, 4, 5, 83, 99, 782};



Objetivo (x) = 5, Número de elementos de arreglo = 10

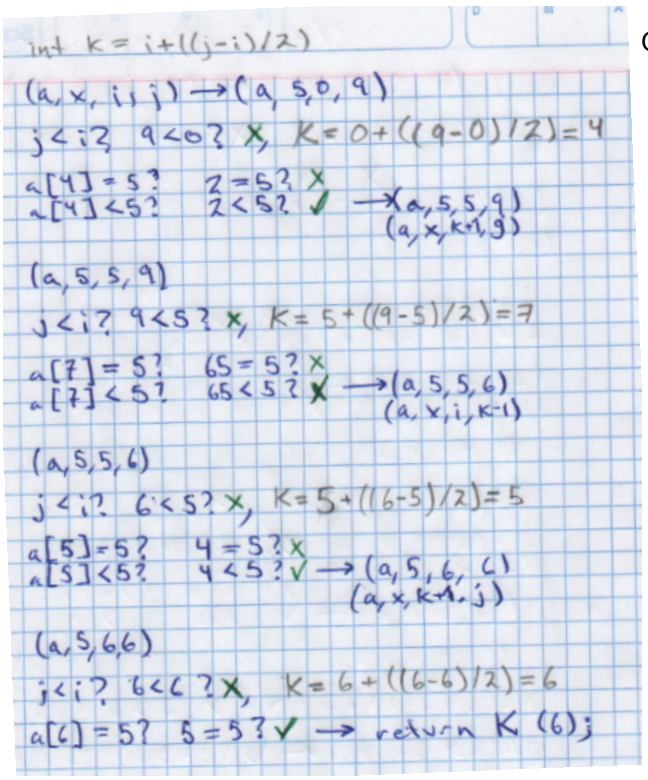


RESULTADOS :
está 5
resultado esperado : índice = 6

```
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# ./elef
{-31, 0, 1, 2, 3, 4, 5, 83, 99, 782}
 (0, 9)  k : 4

 (k+1,j) : (5, 9) porque 3 < 5

 (5, 9)  k : 7

 (i,k-1) : (5, 6) porque 83 !< 5

 (5, 6)  k : 5

 (k+1,j) : (6, 6) porque 4 < 5

 (6, 6)  k : 6

el valor del índice es 6
```

1 2 3 4 5 6 7 8 9 10

a = {-31, 0, 1, 2, 2, 4, 5, 65, 99, 782}



Objetivo (x) = 5, Número de elementos de arreglo = 10

RESULTADOS :

está 5, y en el mismo índice aunque con algunos elementos diferentes,

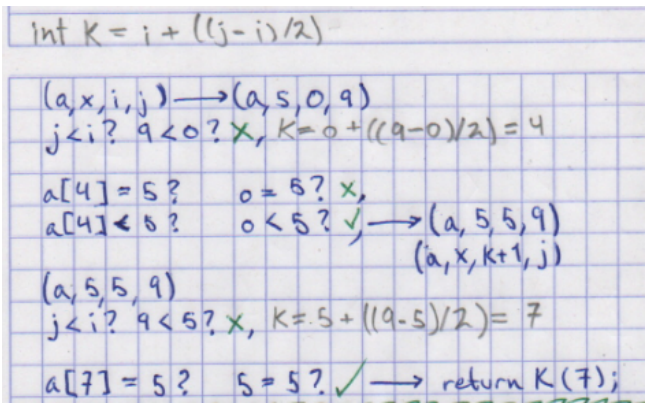
pero debería dar mismo el resultado

resultado esperado : índice = 6

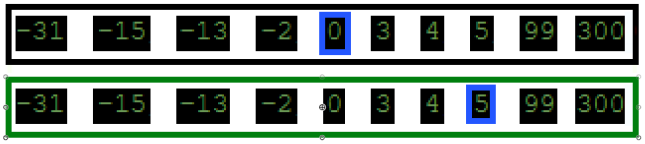
```
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# ./elef
{-31, 0, 1, 2, 2, 4, 5, 65, 99, 782}
 (0, 9)  k : 4

 (k+1,j) : (5, 9) porque 2 < 5

 (5, 9)  k : 7

 (i,k-1) : (5, 6) porque 65 !< 5

 (5, 6)  k : 5

 (k+1,j) : (6, 6) porque 4 < 5

 (6, 6)  k : 6

el valor del índice es 6
```

1 2 3 4 5 6 7 8 9 10

a = {-31, -15, -13, -2, 0, 3, 4, 5, 99, 300}



Objetivo (x)=5, Número de elementos de arreglo=10

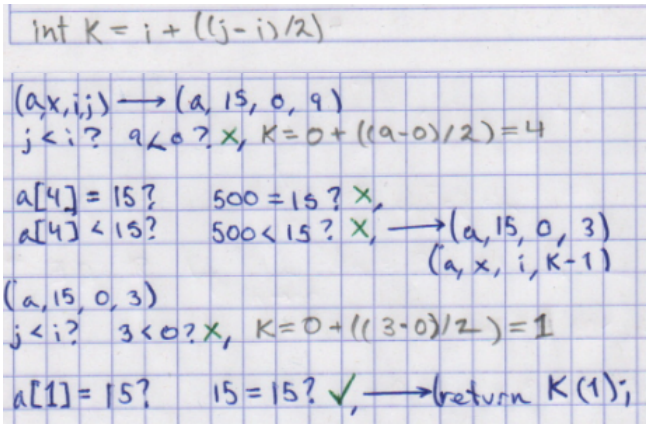


RESULTADOS:
está 5, en diferente índice con algunos elementos diferentes,
resultado esperado : índice = 7

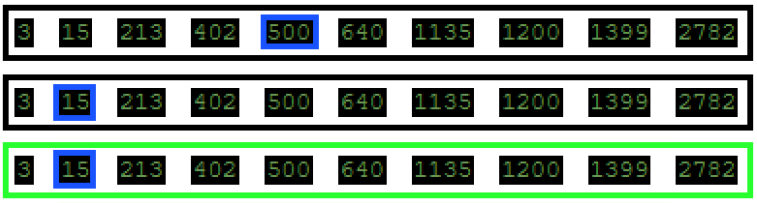
```
el valor del índice es 6
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# gcc binsearch_ro.c -o elef
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# ./elef
{-31, -15, -13, -2, 0, 3, 4, 5, 99, 300}
 (0, 9)  k : 4

 (k+1,j) : (5, 9) porque 0 < 5

 (5, 9)  k : 7
```

1 2 3 4 5 6 7 8 9 10

a = {3, 15, 213, 402, 500, 640, 1135, 1200, 1399, 2782}



Objetivo (x)=15, Número de elementos de arreglo=10

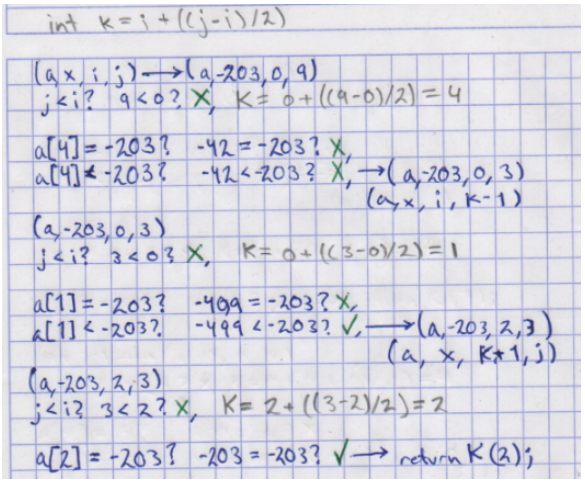


RESULTADOS:
se busca un número diferente + (15) con elementos diferentes,
resultados esperados : índice = 1

```
el valor del índice es 6
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# gcc binsearch_ro.c -o elef
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# ./elef
{-31, -15, -13, -2, 0, 3, 4, 5, 99, 300}
 (0, 9)  k : 4

 (k+1,j) : (5, 9) porque 0 < 5

 (5, 9)  k : 7
```

1 2 3 4 5 6 7 8 9 10

a = {-782, -499, -203, -120, -42, -24, -15, -7, -2, 0}



Objetivo (x) = -203, Número de elementos de arreglo = 10

RESULTADOS:
se busca un número diferente - (-203)
elementos diferentes,
resultados esperados : índice = 2

```
root@DESKTOP-REBRR6L:/mnt/c/Users/Santi/Desktop/PP/RISC-V/R# ./elef
{-782, -499, -203, -120, -42, -24, -15, -7, -2,  0}
 (0, 9)  k : 4

 (i,k-1) : (0, 3) porque -42 !< -203

 (0, 3)  k : 1

 (k+1,j) : (2, 3) porque -499 < -203

 (2, 3)  k : 2

el valor del índice es 2
```

Código modificado
nombre del archivo : binsearch_ro.c

```
#include <stdio.h>

int bsearch_r (int *a, int x, int i, int j) {
    if (j < i) {
    printf("\n\n %d < %d entonces %d no está en el arreglo", j, i, x);
        return -1;
    }
    int k = i + ((j - i) / 2);
    printf("\n (%d, %d)  k : %d\n", i, j, k);
    if (a[k] == x) {
        return k;
    }
    else if (a[k] < x) {
        printf("\n (k+1,j) : (%d, %d) porque %d < %d\n", k+1, j,
a[k], x);
        return bsearch_r(a, x, k + 1, j);
    }
    else {
        printf("\n (i,k-1) : (%d, %d) porque %d !< %d\n", i, k-1,
a[k], x);
        return bsearch_r(a, x, i, k - 1);
    }
}

int main () {
    int a[] = {-782, -499, -203, -120, -42, -24, -15, -7, -2,  0};
    int n = sizeof a / sizeof a[0];
    int x = -203;
    printf("{-782, -499, -203, -120, -42, -24, -15, -7, -2,  0}");
    int i = bsearch_r(a, x, 0, n - 1);
    printf("\nel valor del índice es %d\n", i);
    return 0;
}
```