Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 02: Análisis temporal

Gloria Oliva Olivares Ménez
Boleta: 2020630350

Análisis de Algoritmos

Profr. Edgardo Adrián Franco Martínez

3CM15

14 de abril, 2022

Índice.

Práctica 02: Análisis temporal.

Objetivo.

Realizar un análisis de algoritmos temporal a priori y comprobar mediante la experimentación el cumplimiento del caso medio de 5 algoritmos de búsqueda.

Planteamiento del problema.

Con base en el archivo de entrada de la práctica 01 que tiene 10,000,000 de números diferentes, realizar la búsqueda de elementos bajo 5 métodos. Realizar el análisis temporal a priori (análisis de casos) y a posteriori (empírico de las complejidades).

- Búsqueda lineal o secuencial
- Búsqueda en un árbol binario de búsqueda
- Búsqueda binaria o dicotómica [1]
- Búsqueda exponencial 1
- Búsqueda de Fibonnacci 1

Finalmente, realizar la adaptación de los 5 métodos de búsqueda empleando hilos (Threads) de manera que se mejore el tiempo de búsqueda de cada método.

Entorno de pruebas.

Hubo 2 entornos de pruebas, uno en Windows para probar los códigos en C y otro en Linux para correr con los tiempos. Para Linux, se hizo una máquina virtual a través de Vagrant. Las características del equipo de Windows son:

- Nombre del dispositivo: DESKTOP-SGH89EG
- Procesador: Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz 2.00 GHz
- RAM instalada: 8.00 GB
- Tipo de Sistema: Sistema operativo de 64 bits, procesador x64

De Linux:

- Distribución: Kali Linux
- Arquitectura: x86_64
- Fabricante: innotek GmbH
- Procesador: Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz
- Velocidad: 1995.384 MHz
- Tamaño del caché: 3072 KB

[1] Utilizan arreglos ordenados

Desarrollo.

Análisis teórico de casos.

Búsqueda lineal o secuencial

```
int busquedaLineal(int *A, int x, int n){
    int i; //se asigna variable para recorrer el arreglo
    for(i=0;i<n;i++){  //se recorre el arreglo
        if(x==A[i]){  //si el valor buscado corresponde con uno del arreglo
            return i;  //devuelve el índice de donde esta ubicado el numero
        }
    }
    return -1; //sino devuelve -1
}
```

Código 1. Algoritmo búsqueda lineal

Mejor Caso.

El mejor caso ocurre cuando el primer número del arreglo es el número buscado. En este caso, tendríamos la asignación de i, se hace el ciclo for 1 vez, una vez se hace la condicional y una operación de return, por lo tanto, la ecuación queda como:

$$f(n) = 4$$

Peor Caso.

Aquí tenemos 2 opciones, o el elemento buscado es el último del arreglo o no se encuentra en este.

Para cuando es el último, tendríamos una asignación de i, se repite el ciclo for n+1 veces, la condicional if (comparación) n veces y el return de i se hace 1 vez:

$$f(n) = 2n + 3$$

Si el número no es encontrado en el arreglo, entonces tenemos la asignación de i, la repetición del ciclo for n+1 veces, la repetición (comparación) del if n veces y el return de i se hace una vez:

$$f(n) = 2n + 3$$

Aquí podemos ver que, para ambas situaciones, la ecuación resultó ser la misma.

Caso Medio.

Aquí se toman los casos de:

- Caso 1: Se cumple la condición del if desde la primera iteración. = 4 operaciones
- Caso 2: Se encuentra el número hasta el final = 2n+3 operaciones
- Caso 3: No se encuentra el número = 2n+3 operaciones

Juntando lo anterior tenemos que:

$$\frac{1}{3}(4 + 2n + 3 + 2n + 3)$$

$$f(n) = \frac{1}{3}(10 + 4n)$$

Búsqueda en un árbol binario de búsqueda

```
bool encontrarNumero(arbolABB *raiz, int valor, int n){
    /*
        Estos 2 primeros if se refieren a la raiz del arbol, si la raiz
esta vacía, no se encuentra el valor
        Si el apuntador de la raíz a valor en la estructura del arbol
es igual al valor buscado, se encontró
    */
    if(raiz==NULL){
        printf("Valor NO encontrado\n");
        return false;
    }
    if(raiz->valor==valor){
        printf("Valor encontrado\n");
        return true;
    }
    /*
        Aquí nos vamos al lado izquierdo, que es cuando el valor a
buscar es menor que el valor en la raíz.
        Recorremos toda la parte del lado izquierdo, y de igual forma
si en el nodo no hay ningun valor (NULL)
        pues no se encuentra el valor, pero si el valor del nodo es
igual al buscado, pues regresa un true
    */
    for(int i=0; i<n; i++){
        if(valor<raiz->valor){
            raiz=raiz->izquierda;
            if(raiz==NULL) {
                printf("Valor NO encontrado\n");
                return false;
            }
            if(raiz->valor==valor){
                printf("Valor encontrado\n");
                return true;
            }
            //return encontrarNumero(raiz->izquierda, valor);
        }
        /*
            Lo mismo que se hizo en el lado izquierdo se hace en el
derecho. Se va a pasar al lado derecho
            cuando el valor sea mayor.
        */
        else{
            raiz=raiz->derecha;
            if(raiz==NULL) {
                printf("Valor NO encontrado\n");
                return false;
            }
            if(raiz->valor==valor){
                printf("Valor encontrado\n");
                return true;
            }
            //return encontrarNumero(raiz->derecha, valor);
        }
    }
}
```

Código 2. Algoritmo Árbol de Búsqueda Binaria

Mejor Caso.
Esta vez el mejor caso sería cuando el número está en la raíz del árbol.

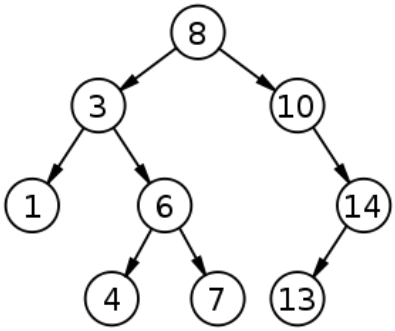


Figura 1. Ejemplo ABB

Tomando la figura 1 como ejemplo, el mejor caso sería que el número a buscar fuera el 8, que está inmediatamente en la raíz, así que no habría que hacer muchas operaciones.

De acuerdo con el código 2, haríamos 2 comparaciones de if y un return, diciendo que sí se encontró el valor, así que la complejidad está dada por:

$$f(n) = 3$$

Peor Caso.
Aquí puede haber 2 posibilidades, que el número buscado estén en el último nivel del árbol, o que no esté. Para cualquiera de las 2 situaciones, podemos ver que la complejidad estaría dada por la altura del árbol.

Cuando el árbol está balanceado, podemos darnos cuenta de que, en cada nivel que se baja o en cada paso que se da, se tiene que dividir el espacio de búsqueda entre 2, por lo que iría quedando de la siguiente forma:

- Paso 1: $n/2^1$
- Paso 2: $n/2^2$
- Paso 3: $n/2^3$
- Paso x: $n/2^x$

Por lo tanto, el tamaño del problema n, sería igual a:

$$n = 2^x$$

Donde x es la altura del árbol, y despejándola quedaría como:

$$x = \log(n)$$

Así que la función complejidad está dada por:

$$f(n) = \log(n)$$

Sin embargo, hay que recalcar que esto funciona solamente cuando el árbol está balanceado. En el caso de la figura 1 por ejemplo, sería diferente.

Ahora veremos la complejidad basada en el código. Tenemos 2 comparaciones if, un ciclo for que se repite n+1 veces, una comparación que se repite n veces, una asignación que

se hace n veces, otra comparación hecha n veces, el return de false se hará n-1 veces y el de true 1 vez:

$$2 + n + 1 + n + n + n + n - 1 + 1$$

$$f(n) = 3 + 5\mathrm{n}$$

En el caso de que no se encuentre el número, cambia que el return de false se hará n veces:

$$2 + n + 1 + n + n + n + n + 1$$

$$f(n) = 4 + 5\mathrm{n}$$

Aquí a diferencia de las veces anteriores, si cambia un poco.

Caso Medio.
Aquí podemos tomar el caso promedio, ya que es muy inexacto el cálculo del caso medio. Para esto podemos llegar a insertar muchos elementos y ni siquiera aseguramos que el árbol esté balanceado, lo que lo haría mucho más difícil, así que la fórmula queda:

$$f(n) = \frac{3 + \log(n)}{2}$$

Búsqueda binaria o dicotómica

```
int busquedaBinaria(int array[], int l, int r, int x){
    //l lado inferior y r lado superior
    while (l <= r) {
        int m = l + (r - l) / 2;

        // Checa que el valor a buscar esté en medio y retorna su posicion
        if (array[m] == x)
            return m;

        // Si el valor a buscar es mayor, se ignora el resto de la mitad
de la izquierda
        if (array[m] < x)
            l = m + 1;

        // Si el valor a buscar es menor, se ignora el resto de la mitad
de la derecha
        else
            r = m - 1;
    }

    return -1; // Si no se encuentran coincidencias se retorna -1
}
```

Mejor Caso.

El mejor caso ocurre cuando el elemento a buscar se encuentra en el punto medio de nuestro arreglo por lo que solo se haría la comparación de un if es decir solo se hará una iteración dándonos una complejidad de:

$$f(n) = 1$$

Peor Caso.

El peor caso en este método de búsqueda es cuando el elemento a buscar no se encuentra en el arreglo por lo que pasaría por todas las comparaciones, teniendo una complejidad:

$$f(n) = \log(n)$$

Caso Medio.

Su complejidad de caso medio es igual a la complejidad del peor caso, es decir se tiene una complejidad:

$$f(n) = \log(n)$$

Búsqueda exponencial

```
int busquedaExponencial(int array[], int n, int x){
    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    //uswtime(&utime0, &stime0, &wtime0);

    // Checa si el valor a buscar se encuentra en la primera posición
    if (array[0] == x)
        return 0;

    // Se calcula un rango para la busqueda binaria elevando 2^n su paso y
verificando
    //que el index sea menor o igual al valor a encontrar
    int i = 1;
    while (i < n && array[i] <= x)
        i = i*2;

    //Se hace una busqueda binaria con el rango calculado.
    return busquedaBinaria(array, i/2, min(i, n-1), x);
}
```

Mejor Caso.

El mejor de los casos ocurre cuando el elemento comparado es igual al elemento que se está buscando, por lo tanto, se devuelve en la primera iteración dándonos una función de complejidad de:

$$f(n) = 1$$

Peor Caso.

La complejidad del peor caso como en todos o casi todos los algoritmos de busqueda es cuando el elemento a buscar no se encuentra en el arreglo que se está buscando por lo tanto hace todo el número de iteraciones posibles hasta llegar a la conclusión de que el elemento buscado no está en el arreglo.

Para este algoritmo la complejidad de su peor caso es igual a:

$$f(n) = \log(i)$$

Donde i es el índice del elemento a buscar.

Caso Medio.

El caso promedio de este algoritmo esta dado por la misma función del peor caso es decir es:

$$f(n) = \log(i)$$

i como sabemos será el índice del elemento a buscar es decir si se encuentra en la posición n del arreglo su complejidad será log(n).

Búsqueda de Fibonnacci

```
int BusquedaDeFibonacci(int array[], int n, int x){
    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);

    /* Inicializa los primeros números de Fibonacci */
    int fibMMm2 = 0; // (m-2)ésimo numero de Fibonacci.
    int fibMMm1 = 1; // (m-1)ésimo numero de Fibonacci.
    int fibM = fibMMm2 + fibMMm1; // (m)ésimo numero de Fibonacci

    /* la variable fibM se encargará de guardar el número de Fibonacci más
    pequeño que sea menor o igual a n */
    while (fibM < n) {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }

    // Marca el rango eliminado del frente
    int compensacion = -1;

    /*
       El while se cumplira mientras haya elementos con los que se puede
operar
       Dentro del while se compara que el valor de arr[fibMm2] con el
valor a buscar
```

```
       Cuándo fibM toma el valor de 1, fibMm2 toma el valor de 0  */
    while (fibM > 1) {
        // Verifica que fibMm2 isea una valida locacion
        int i = min(compensacion + fibMMm2, n - 1);

        /* Si el número buscar es mayor que el valor en el index de
fibMm2,
        se corta el rango del arreglo desde la compensasion hasta i*/
        if (array[i] < x) {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            compensacion = i;
        }

        /* Si el número buscar es menor que el valor en el index de
fibMm2,
           se corta el rango después de i+1  */
        else if (array[i] > x) {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }

        /* El valor a buscar es encontrado y se retorna su posicion */
        else
            return i;
    }

    /* Compara el ultimo elemento con el numero a encontrar */
    if (fibMMm1 && array[compensacion + 1] == x)
        return compensacion + 1;

    // Si no se encuentran coincidencias se retorna -1
    return -1;
}

int min(int a, int b){
    return (a < b ? a : b);// retorna el número más pequeño a través de un
ternario
}
```

Mejor Caso.

El mejor caso ocurre como en los anteriores algoritmos que hemos analizado, este caso es cuando el elemento a buscar es el primer elemento que comparamos y se regresa en la primera iteración, por lo tanto, su complejidad es:

$$f(n) = 1$$

Peor Caso.

El peor caso ocurre cuando el elemento a buscar esta siempre en el subarreglo más grande por lo tanto la complejidad de tiempo en el peor de los casos es:

$$f(n) = \log(n)$$

Caso Medio.

La complejidad del caso medio está definida también por la complejidad en el peor de los casos, por lo tanto, su complejidad media es:

$$f(n) = \log(n)$$

Registro del tiempo.

Búsqueda lineal o secuencial

Tamaño de problema n	**Números por buscar**	**Tiempo (seg)**		
		Tiempo real	*Tiempo CPU*	*Tiempo E/S*
1000000	322486	0.0149328709	0.0149480000	0.0000000000
	14700764	0.0133609772	0.0133430000	0.0000340000
	3128036	0.0144948959	0.0144500000	0.0000690000
	6337399	0.0138199329	0.0138360000	0.0000000000
	61396	0.0151569843	0.0143790000	0.0000000000
	10393545	0.0145549774	0.0145400000	0.0000000000
	2147445644	0.0152728558	0.0152920000	0.0000000000
	1295390003	0.0143511295	0.0143110000	0.0000560000
	450057883	0.0142531395	0.0142710000	0.0000000000
	187645041	0.0148599148	0.0148810000	0.0000000000
	1980098116	0.0149371624	0.0149680000	0.0000000000
	152503	0.0133519173	0.0133450000	0.0000340000
	5000	0.0143489838	0.0142920000	0.0000810000
	1493283650	0.0088641644	0.0088850000	0.0000000000
	214826	0.0137591362	0.0137350000	0.0000440000
	1843349527	0.0137760639	0.0137350000	0.0000630000
	1360839354	0.0144128799	0.0143610000	0.0000670000
	2109248666	0.0141251087	0.0140880000	0.0000530000
	2147470852	0.0133578777	0.0133300000	0.0000500000

	0	0.0134210587	0.0134170000	0.0000250000
2000000	322486	0.0411889553	0.0411420000	0.0000620000
	14700764	0.0337429047	0.0336680000	0.0000340000
	3128036	0.0209610462	0.0209630000	0.0000180000
	6337399	0.0308859348	0.0307990000	0.0001020000
	61396	0.0269331932	0.0269480000	0.0000000000
	10393545	0.0276620388	0.0276780000	0.0000000000
	2147445644	0.0267250538	0.0266390000	0.0000620000
	1295390003	0.0317978859	0.0317600000	0.0000550000
	450057883	0.0277750492	0.0277940000	0.0000000000
	187645041	0.0294601917	0.0294800000	0.0000000000
	1980098116	0.0301711559	0.0301140000	0.0000510000
	152503	0.0290160179	0.0290120000	0.0000470000
	5000	0.0383558273	0.0384260000	0.0000000000
	1493283650	0.0074729919	0.0073800000	0.0001110000
	214826	0.0394158363	0.0327180000	0.0040220000
	1843349527	0.0346279144	0.0342530000	0.0000460000
	1360839354	0.0404989719	0.0403400000	0.0001830000
	2109248666	0.0267400742	0.0267590000	0.0000000000
	2147470852	0.0355370045	0.0355600000	0.0000000000
	0	0.0287828445	0.0287570000	0.0000130000
3000000	322486	0.0395450592	0.0395190000	0.0000000000
	14700764	0.0449519157	0.0449500000	0.0000190000
	3128036	0.0182149410	0.0142650000	0.0039680000
	6337399	0.0401670933	0.0400400000	0.0000040000
	61396	0.0434591770	0.0434280000	0.0000000000
	10393545	0.0381700993	0.0381470000	0.0000000000
	2147445644	0.0484030247	0.0482170000	0.0000000000
	1295390003	0.0402331352	0.0402530000	0.0000000000
	450057883	0.0402808189	0.0402970000	0.0000000000
	187645041	0.0449850559	0.0449830000	0.0000210000
	1980098116	0.0396587849	0.0396750000	0.0000000000
	152503	0.0402758121	0.0402500000	0.0000020000

	5000	0.0399119854	0.0399290000	0.0000000000
	1493283650	0.0076029301	0.0076210000	0.0000000000
	214826	0.0399630070	0.0399800000	0.0000000000
	1843349527	0.0349218845	0.0348690000	0.0000730000
	1360839354	0.0494039059	0.0493630000	0.0000590000
	2109248666	0.0505180359	0.0497380000	0.0000350000
	2147470852	0.0490119457	0.0489760000	0.0000080000
	0	0.0397670269	0.0397860000	0.0000000000
4000000	322486	0.0535399914	0.0535190000	0.0000330000
	14700764	0.0628278255	0.0627590000	0.0000940000
	3128036	0.0207600594	0.0207550000	0.0000310000
	6337399	0.0549540520	0.0549800000	0.0000000000
	61396	0.0524950027	0.0524010000	0.0000110000
	10393545	0.0603051186	0.0583290000	0.0000340000
	2147445644	0.0567610264	0.0567840000	0.0000230000
	1295390003	0.0542578697	0.0542150000	0.0000360000
	450057883	0.0549080372	0.0549250000	0.0000000000
	187645041	0.0538659096	0.0538320000	0.0000510000
	1980098116	0.0594069958	0.0591250000	0.0000000000
	152503	0.0593800545	0.0589300000	0.0000000000
	5000	0.0587558746	0.0586780000	0.0000000000
	1493283650	0.0075850487	0.0075890000	0.0000000000
	214826	0.0569069386	0.0566580000	0.0000140000
	1843349527	0.0343739986	0.0329700000	0.0000000000
	1360839354	0.0573368073	0.0564810000	0.0000000000
	2109248666	0.0563941002	0.0563860000	0.0000040000
	2147470852	0.0536339283	0.0536120000	0.0000460000
	0	0.0554099083	0.0554270000	0.0000000000
5000000	322486	0.0699429512	0.0697940000	0.0000000000
	14700764	0.0705358982	0.0705520000	0.0000000000
	3128036	0.0183918476	0.0184090000	0.0000000000
	6337399	0.0791389942	0.0788820000	0.0000540000
	61396	0.1598949432	0.1544750000	0.0053410000

	10393545	0.1158809662	0.1158240000	0.0000000000
	2147445644	0.1261029243	0.1246860000	0.0002720000
	1295390003	0.0665550232	0.0664740000	0.0000000000
	450057883	0.0681099892	0.0678620000	0.0000000000
	187645041	0.0652208328	0.0651520000	0.0000000000
	1980098116	0.0682430267	0.0682210000	0.0000000000
	152503	0.0673530102	0.0672440000	0.0000000000
	5000	0.0770130157	0.0769100000	0.0000320000
	1493283650	0.0072619915	0.0072790000	0.0000000000
	214826	0.0691430569	0.0689710000	0.0000310000
	1843349527	0.0419840813	0.0376680000	0.0041030000
	1360839354	0.0715920925	0.0716160000	0.0000000000
	2109248666	0.0711009502	0.0711250000	0.0000000000
	2147470852	0.0822458267	0.0812840000	0.0000670000
	0	0.0665938854	0.0665200000	0.0000430000
6000000	322486	0.0796549320	0.0794720000	0.0000000000
	14700764	0.0846910477	0.0846440000	0.0000090000
	3128036	0.0178220272	0.0178380000	0.0000000000
	6337399	0.0788121223	0.0787320000	0.0000000000
	61396	0.0801689625	0.0792590000	0.0000000000
	10393545	0.0785109997	0.0783790000	0.0000000000
	2147445644	0.0792410374	0.0792580000	0.0000000000
	1295390003	0.0813260078	0.0813240000	0.0000200000
	450057883	0.0839550495	0.0838870000	0.0000000000
	187645041	0.0797219276	0.0797700000	0.0000000000
	1980098116	0.0798549652	0.0798290000	0.0000000000
	152503	0.0789549351	0.0789280000	0.0000440000
	5000	0.0794827938	0.0794570000	0.0000000000
	1493283650	0.0071380138	0.0071540000	0.0000000000
	214826	0.0798120499	0.0797650000	0.0000000000
	1843349527	0.0367178917	0.0367200000	0.0000140000
	1360839354	0.0797770023	0.0797130000	0.0000000000
	2109248666	0.0790410042	0.0789470000	0.0000290000

	2147470852	0.0797441006	0.0797400000	0.0000000000
	0	0.0809109211	0.0800400000	0.0000000000
7000000	322486	0.1022489071	0.1020910000	0.0000000000
	14700764	0.1155378819	0.1153420000	0.0000000000
	3128036	0.0183329582	0.0183510000	0.0000000000
	6337399	0.1018571854	0.1018360000	0.0000420000
	61396	0.1001710892	0.1000880000	0.0000000000
	10393545	0.1417009830	0.1411220000	0.0004360000
	2147445644	0.1175808907	0.1173300000	0.0000000000
	1295390003	0.1084570885	0.1084280000	0.0000000000
	450057883	0.0932970047	0.0933000000	0.0000000000
	187645041	0.1050298214	0.1049040000	0.0000740000
	1980098116	0.0943458080	0.0943230000	0.0000000000
	152503	0.0982048512	0.0981650000	0.0000070000
	5000	0.1031489372	0.1031760000	0.0000000000
	1493283650	0.0092179775	0.0092260000	0.0000330000
	214826	0.1022119522	0.1022300000	0.0000000000
	1843349527	0.0319628716	0.0319800000	0.0000000000
	1360839354	0.1100239754	0.1097370000	0.0002810000
	2109248666	0.0886809826	0.0886390000	0.0000000000
	2147470852	0.0987529755	0.0986470000	0.0000670000
	0	0.1348509789	0.1045890000	0.0000000000
8000000	322486	0.1060318947	0.1060550000	0.0000000000
	14700764	0.1064350605	0.1063920000	0.0000280000
	3128036	0.0182819366	0.0182990000	0.0000000000
	6337399	0.0835459232	0.0834950000	0.0000670000
	61396	0.1074600220	0.1072810000	0.0000000000
	10393545	0.0922288895	0.0922060000	0.0000000000
	2147445644	0.1069178581	0.1069580000	0.0000000000
	1295390003	0.1111490726	0.1109780000	0.0000000000
	450057883	0.1053659916	0.1053610000	0.0000220000
	187645041	0.1059460640	0.1059640000	0.0000000000
	1980098116	0.1097791195	0.1097300000	0.0000260000

	152503	0.1063241959	0.1063150000	0.0000270000
	5000	0.1063911915	0.1063800000	0.0000280000
	1493283650	0.0071940422	0.0072120000	0.0000000000
	214826	0.1072499752	0.1072650000	0.0000000000
	1843349527	0.0327138901	0.0327170000	0.0000150000
	1360839354	0.1061770916	0.1060350000	0.0000310000
	2109248666	0.1075420380	0.1074480000	0.0001110000
	2147470852	0.1039161682	0.1037960000	0.0000000000
	0	0.1068170071	0.1068390000	0.0000000000
9000000	322486	0.1361889839	0.1360180000	0.0000000000
	14700764	0.1199879646	0.1196850000	0.0000950000
	3128036	0.0180289745	0.0180100000	0.0000370000
	6337399	0.0855100155	0.0854850000	0.0000000000
	61396	0.1385109425	0.1382880000	0.0000270000
	10393545	0.0914728642	0.0874730000	0.0039970000
	2147445644	0.1204149723	0.1204140000	0.0000000000
	1295390003	0.1095399857	0.1094870000	0.0000160000
	450057883	0.1179049015	0.1179210000	0.0000000000
	187645041	0.1187369823	0.1186340000	0.0000000000
	1980098116	0.1229290962	0.1227540000	0.0000000000
	152503	0.1309990883	0.1254310000	0.0000250000
	5000	0.1182689667	0.1182330000	0.0000380000
	1493283650	0.0071542263	0.0071160000	0.0000550000
	214826	0.1186180115	0.1185360000	0.0000420000
	1843349527	0.0314309597	0.0314470000	0.0000000000
	1360839354	0.1178429127	0.1177770000	0.0000240000
	2109248666	0.0873579979	0.0873190000	0.0000000000
	2147470852	0.1037628651	0.1037780000	0.0000000000
	0	0.1185030937	0.1184570000	0.0000000000
10000000	322486	0.1161210537	0.1120270000	0.0039020000
	14700764	0.1384668350	0.1383410000	0.0000000000
	3128036	0.0168550014	0.0168590000	0.0000130000
	6337399	0.0826520920	0.0826270000	0.0000410000

	61396	0.1325612068	0.1324280000	0.0000000000
	10393545	0.1118481159	0.1115750000	0.0000000000
	2147445644	0.1346800327	0.1346110000	0.0000440000
	1295390003	0.1129109859	0.1087550000	0.0039300000
	450057883	0.1325480938	0.1322780000	0.0000000000
	187645041	0.1332089901	0.1330850000	0.0000000000
	1980098116	0.1424579620	0.1422940000	0.0001090000
	152503	0.1305079460	0.1305240000	0.0000000000
	5000	0.1318838596	0.1319000000	0.0000000000
	1493283650	0.0072550774	0.0072730000	0.0000000000
	214826	0.1314780712	0.1313510000	0.0000000000
	1843349527	0.0315730572	0.0315760000	0.0000000000
	1360839354	0.1327869892	0.1327580000	0.0000000000
	2109248666	0.0876991749	0.0876440000	0.0000000000
	2147470852	0.1033968925	0.1032980000	0.0000000000
	0	0.1320908070	0.1318880000	0.0000000000

Búsqueda en un árbol binario de búsqueda

Tamaño de problema n	**Números por buscar**	**Tiempo (seg)**		
		Tiempo real	*Tiempo CPU*	*Tiempo E/S*
1000000	322486	0.0000069141	0.000023	0.000002
	14700764	0.0000050068	0.000023	0.000001
	3128036	0.0000050068	0.000025	0.000001
	6337399	0.0000059605	0.000023	0.000003
	61396	0.0000050068	0.000022	0.000001
	10393545	0.0000050068	0.000023	0.000002
	2147445644	0.0000059605	0.000026	0.000001
	1295390003	0.0000059605	0.000023	0.000001
	450057883	0.0000050068	0.000024	0.000001
	187645041	0.0000050068	0.000022	0.000001
	1980098116	0.0000050068	0.000022	0.000001
	152503	0.0000040531	0.000023	0.000002
	5000	0.0000050068	0.000023	0.000001

	1493283650	0.0000061989	0.000023	0.000001
	214826	0.0000069141	0.000034	0.000002
	1843349527	0.0000059605	0.000022	0.000001
	1360839354	0.0000059605	0.000023	0.000001
	2109248666	0.0000059605	0.000021	0.000001
	2147470852	0.0000040531	0.000021	0.000002
	0	0.0000050068	0.00002	0.000001
2000000	322486	0.0000069141	0.000024	0.000001
	14700764	0.0000040531	0.000021	0.000001
	3128036	0.0000059605	0.000024	0.000001
	6337399	0.0000059605	0.000021	0.000001
	61396	0.0000061989	0.000022	0.000001
	10393545	0.0000050068	0.000021	0.000001
	2147445644	0.0000038147	0.000023	0.000001
	1295390003	0.0000069141	0.000025	0.000002
	450057883	0.0000069141	0.000025	0.000001
	187645041	0.0000061989	0.000023	0.000001
	1980098116	0.0000050068	0.000026	0.000001
	152503	0.0000061989	0.000022	0.000001
	5000	0.0000050068	0.000024	0.000002
	1493283650	0.0000059605	0.000022	0.000001
	214826	0.0000069141	0.000026	0.000001
	1843349527	0.0000069141	0.000027	0.000001
	1360839354	0.0000071526	0.000025	0.000002
	2109248666	0.0000069141	0.000023	0.000001
	2147470852	0.0000038147	0.000023	0.000001
	0	0.0000059605	0.000026	0.000001
3000000	322486	0.0000090599	0.000036	0.000002
	14700764	0.0000047684	0.000024	0.000001
	3128036	0.0000059605	0.000025	0.000002
	6337399	0.0000059605	0.000023	0.000001
	61396	0.0000059605	0.000026	0.000001
	10393545	0.0000069141	0.000023	0.000001

	2147445644	0.0000050068	0.000023	0.000002
	1295390003	0.0000059605	0.000029	0.000002
	450057883	0.0000069141	0.000028	0.000002
	187645041	0.0000059605	0.000027	0.000001
	1980098116	0.0000059605	0.000037	0.000001
	152503	0.0000071526	0.000035	0.000002
	5000	0.0000071526	0.000025	0.000001
	1493283650	0.0000081062	0.000027	0.000002
	214826	0.0000071526	0.000027	0.000001
	1843349527	0.0000050068	0.000024	0.000002
	1360839354	0.0000069141	0.000029	0.000002
	2109248666	0.0000050068	0.000022	0.000002
	2147470852	0.0000050068	0.000024	0.000001
	0	0.0000071526	0.000026	0.000001
4000000	322486	0.0000071526	0.000026	0.000001
	14700764	0.0000050068	0.000024	0.000001
	3128036	0.0000059605	0.000022	0.000001
	6337399	0.0000059605	0.000024	0.000001
	61396	0.0000059605	0.000023	0.000001
	10393545	0.0000069141	0.000025	0.000002
	2147445644	0.0000040531	0.000023	0.000001
	1295390003	0.0000069141	0.000024	0.000001
	450057883	0.0000059605	0.000023	0.000001
	187645041	0.0000059605	0.000024	0.000001
	1980098116	0.0000059605	0.000029	0.000001
	152503	0.0000061989	0.000029	0.000002
	5000	0.0000059605	0.000022	0.000001
	1493283650	0.0000061989	0.000022	0.000001
	214826	0.0000059605	0.000022	0.000001
	1843349527	0.0000059605	0.000024	0.000001
	1360839354	0.0000071526	0.000024	0.000001
	2109248666	0.0000059605	0.000023	0.000001
	2147470852	0.0000061989	0	0.000056

	0	0.0000059605	0.000021	0.000001
5000000	322486	0.0000100136	0.000031	0.000001
	14700764	0.0000050068	0.000025	0.000001
	3128036	0.0000071526	0.000025	0.000001
	6337399	0.0000059605	0.000027	0.000001
	61396	0.0000059605	0.000024	0.000001
	10393545	0.0000061989	0.000026	0.000001
	2147445644	0.0000050068	0.000023	0.000001
	1295390003	0.0000069141	0.000026	0.000001
	450057883	0.0000059605	0.000028	0.000001
	187645041	0.0000059605	0.000023	0.000002
	1980098116	0.0000050068	0.000025	0.000001
	152503	0.0000078678	0.000027	0.000001
	5000	0.0000100136	0.000036	0.000001
	1493283650	0.0000061989	0.000024	0.000001
	214826	0.0000069141	0.000026	0.000002
	1843349527	0.0000059605	0.000022	0.000001
	1360839354	0.0000069141	0.000025	0.000001
	2109248666	0.0000059605	0.000025	0.000001
	2147470852	0.0000059605	0.000023	0.000001
	0	0.0000059605	0.000024	0.000001
6000000	322486	0.0000078678	0.000025	0.000001
	14700764	0.0000050068	0.000024	0.000001
	3128036	0.0000059605	0.000023	0.000001
	6337399	0.0000061989	0.000025	0.000001
	61396	0.0000071526	0.000026	0.000001
	10393545	0.0000059605	0.000025	0.000001
	2147445644	0.0000050068	0.000024	0.000001
	1295390003	0.0000059605	0.000024	0.000002
	450057883	0.0000059605	0.000025	0.000001
	187645041	0.0000059605	0.000022	0.000001
	1980098116	0.0000050068	0.000022	0.000001
	152503	0.0000059605	0.000024	0.000001

	5000	0.0000050068	0.000025	0.000002
	1493283650	0.0000059605	0.000024	0.000001
	214826	0.0000069141	0.000023	0.000001
	1843349527	0.0000071526	0.000025	0.000001
	1360839354	0.0000078678	0.000026	0.000001
	2109248666	0.0000071526	0.000024	0.000001
	2147470852	0.0000050068	0.000023	0.000001
	0	0.0000059605	0.000028	0.000001
7000000	322486	0.0000081062	0.000026	0.000001
	14700764	0.0000050068	0.000026	0.000001
	3128036	0.0000069141	0.000031	0.000001
	6337399	0.0000059605	0.000023	0.000001
	61396	0.0000059605	0.000023	0.000001
	10393545	0.0000061989	0.000025	0.000001
	2147445644	0.0000050068	0.000021	0.000001
	1295390003	0.0000059605	0.000022	0.000001
	450057883	0.0000069141	0.000029	0.000001
	187645041	0.0000061989	0.000025	0.000001
	1980098116	0.0000050068	0.000022	0.000001
	152503	0.0000059605	0.000024	0.000001
	5000	0.0000069141	0.000024	0.000001
	1493283650	0.0000059605	0.000023	0.000001
	214826	0.0000069141	0.000025	0.000001
	1843349527	0.0000059605	0.000023	0.000001
	1360839354	0.0000081062	0.000028	0.000001
	2109248666	0.0000059605	0.000025	0.000001
	2147470852	0.0000050068	0.000024	0.000001
	0	0.0000059605	0.000024	0
8000000	322486	0.0000081062	0.000024	0.000001
	14700764	0.0000050068	0.000023	0.000001
	3128036	0.0000069141	0.000026	0.000001
	6337399	0.0000069141	0.000025	0.000001
	61396	0.0000059605	0.000025	0.000001

	10393545	0.0000061989	0.000025	0.000002
	2147445644	0.0000040531	0.000021	0.000001
	1295390003	0.0000069141	0.000024	0.000001
	450057883	0.0000059605	0.000023	0.000001
	187645041	0.0000061989	0.000024	0.000001
	1980098116	0.0000050068	0.000024	0.000001
	152503	0.0000059605	0.000022	0.000001
	5000	0.0000059605	0.000025	0.000001
	1493283650	0.0000050068	0.000023	0.000001
	214826	0.0000140667	0.000045	0.000001
	1843349527	0.0000061989	0.000024	0.000001
	1360839354	0.0000069141	0.000024	0.000001
	2109248666	0.0000059605	0.000025	0.000001
	2147470852	0.0000059605	0.000022	0.000001
	0	0.0000069141	0.000022	0.000001
9000000	322486	0.0000090599	0.000026	0.000001
	14700764	0.0000059605	0.000023	0.000001
	3128036	0.0000059605	0.000022	0.000001
	6337399	0.0000069141	0.000025	0.000001
	61396	0.0000059605	0.000025	0.000001
	10393545	0.0000069141	0.000024	0.000001
	2147445644	0.0000040531	0.000023	0.000001
	1295390003	0.0000081062	0.000024	0.000001
	450057883	0.0000061989	0.000025	0.000001
	187645041	0.0000071526	0.000024	0.000001
	1980098116	0.0000050068	0.000022	0.000001
	152503	0.0000059605	0.000025	0.000001
	5000	0.0000071526	0.000023	0.000001
	1493283650	0.0000059605	0.000025	0.000001
	214826	0.0000071526	0.000026	0.000001
	1843349527	0.0000059605	0.000024	0
	1360839354	0.0000081062	0.000026	0.000001
	2109248666	0.0000059605	0.000026	0.000001

	2147470852	0.0000050068	0.000022	0.000001
	0	0.0000069141	0.000025	0.000001
10000000	322486	0.0000078678	0.000024	0.000002
	14700764	0.0000050068	0.000024	0.000001
	3128036	0.0000061989	0.000023	0.000001
	6337399	0.0000059605	0.000025	0.000001
	61396	0.0000059605	0.000026	0.000001
	10393545	0.0000059605	0.000026	0.000001
	2147445644	0.0000061989	0.000023	0
	1295390003	0.0000081062	0.000024	0.000001
	450057883	0.0000059605	0.000025	0.000001
	187645041	0.0000059605	0.000024	0.000001
	1980098116	0.0000050068	0.000021	0.000001
	152503	0.0000050068	0.000025	0.000001
	5000	0.0000059605	0.000022	0.000001
	1493283650	0.0000050068	0.000024	0
	214826	0.0000059605	0.000023	0.000001
	1843349527	0.0000059605	0.000026	0.000001
	1360839354	0.0000071526	0.000024	0
	2109248666	0.0000059605	0.000024	0.000001
	2147470852	0.0000059605	0.000023	0.000001
	0	0.0000059605	0.000022	0.000001

Búsqueda binaria o dicotómica

Tamaño de problema n	**Números por buscar**	**Tiempo (seg)**		
		Tiempo real	*Tiempo CPU*	*Tiempo E/S*
1000000	322486	0.0000028610	0.0000080000	0.0000010000
	14700764	0.0000028610	0.0000080000	0.0000000000
	3128036	0.0000030994	0.0000090000	0.0000010000
	6337399	0.0000028610	0.0000080000	0.0000000000
	61396	0.0000021458	0.0000090000	0.0000010000
	10393545	0.0000030994	0.0000080000	0.0000020000
	2147445644	0.0000021458	0.0000080000	0.0000010000

	1295390003	0.0000019073	0.0000070000	0.0000000000
	450057883	0.0000019073	0.0000070000	0.0000020000
	187645041	0.0000019073	0.0000070000	0.0000010000
	1980098116	0.0000009537	0.0000090000	0.0000000000
	152503	0.0000019073	0.0000090000	0.0000000000
	5000	0.0000019073	0.0000080000	0.0000000000
	1493283650	0.0000019073	0.0000070000	0.0000010000
	214826	0.0000019073	0.0000080000	0.0000000000
	1843349527	0.0000021458	0.0000070000	0.0000010000
	1360839354	0.0000021458	0.0000080000	0.0000000000
	2109248666	0.0000021458	0.0000080000	0.0000010000
	2147470852	0.0000019073	0.0000080000	0.0000010000
	0	0.0000019073	0.0000090000	0.0000000000
2000000	322486	0.0000030994	0.0000090000	0.0000000000
	14700764	0.0000019073	0.0000100000	0.0000000000
	3128036	0.0000019073	0.0000110000	0.0000010000
	6337399	0.0000030994	0.0000090000	0.0000010000
	61396	0.0000028610	0.0000100000	0.0000000000
	10393545	0.0000028610	0.0000090000	0.0000010000
	2147445644	0.0000019073	0.0000070000	0.0000010000
	1295390003	0.0000019073	0.0000080000	0.0000000000
	450057883	0.0000019073	0.0000080000	0.0000000000
	187645041	0.0000028610	0.0000100000	0.0000000000
	1980098116	0.0000019073	0.0000080000	0.0000000000
	152503	0.0000088215	0.0000440000	0.0000040000
	5000	0.0000028610	0.0000080000	0.0000000000
	1493283650	0.0000021458	0.0000090000	0.0000000000
	214826	0.0000019073	0.0000080000	0.0000010000
	1843349527	0.0000021458	0.0000080000	0.0000000000
	1360839354	0.0000019073	0.0000080000	0.0000010000
	2109248666	0.0000019073	0.0000080000	0.0000010000
	2147470852	0.0000021458	0.0000080000	0.0000000000
	0	0.0000021458	0.0000080000	0.0000010000

3000000	322486	0.0000021458	0.0000090000	0.0000000000
	14700764	0.0000030994	0.0000090000	0.0000000000
	3128036	0.0000028610	0.0000090000	0.0000000000
	6337399	0.0000019073	0.0000080000	0.0000010000
	61396	0.0000019073	0.0000090000	0.0000000000
	10393545	0.0000021458	0.0000090000	0.0000000000
	2147445644	0.0000019073	0.0000070000	0.0000010000
	1295390003	0.0000021458	0.0000080000	0.0000010000
	450057883	0.0000030994	0.0000090000	0.0000000000
	187645041	0.0000040531	0.0000090000	0.0000000000
	1980098116	0.0000019073	0.0000080000	0.0000000000
	152503	0.0000030994	0.0000140000	0.0000010000
	5000	0.0000030994	0.0000090000	0.0000010000
	1493283650	0.0000019073	0.0000080000	0.0000000000
	214826	0.0000030994	0.0000090000	0.0000000000
	1843349527	0.0000019073	0.0000070000	0.0000000000
	1360839354	0.0000021458	0.0000090000	0.0000000000
	2109248666	0.0000019073	0.0000090000	0.0000000000
	2147470852	0.0000019073	0.0000080000	0.0000010000
	0	0.0000019073	0.0000090000	0.0000010000
4000000	322486	0.0000028610	0.0000090000	0.0000000000
	14700764	0.0000019073	0.0000100000	0.0000000000
	3128036	0.0000030994	0.0000090000	0.0000000000
	6337399	0.0000030994	0.0000090000	0.0000010000
	61396	0.0000021458	0.0000080000	0.0000000000
	10393545	0.0000028610	0.0000090000	0.0000000000
	2147445644	0.0000021458	0.0000080000	0.0000000000
	1295390003	0.0000021458	0.0000080000	0.0000010000
	450057883	0.0000040531	0.0000110000	0.0000010000
	187645041	0.0000028610	0.0000090000	0.0000000000
	1980098116	0.0000021458	0.0000090000	0.0000000000
	152503	0.0000028610	0.0000090000	0.0000000000
	5000	0.0000030994	0.0000090000	0.0000000000

	1493283650	0.0000030994	0.0000080000	0.0000000000
	214826	0.0000028610	0.0000090000	0.0000000000
	1843349527	0.0000019073	0.0000070000	0.0000010000
	1360839354	0.0000021458	0.0000080000	0.0000000000
	2109248666	0.0000019073	0.0000070000	0.0000000000
	2147470852	0.0000021458	0.0000080000	0.0000010000
	0	0.0000040531	0.0000180000	0.0000010000
5000000	322486	0.0000019073	0.0000090000	0.0000010000
	14700764	0.0000028610	0.0000090000	0.0000010000
	3128036	0.0000040531	0.0000100000	0.0000000000
	6337399	0.0000028610	0.0000090000	0.0000010000
	61396	0.0000021458	0.0000090000	0.0000000000
	10393545	0.0000019073	0.0000080000	0.0000010000
	2147445644	0.0000019073	0.0000070000	0.0000010000
	1295390003	0.0000040531	0.0000110000	0.0000000000
	450057883	0.0000028610	0.0000090000	0.0000010000
	187645041	0.0000040531	0.0000110000	0.0000000000
	1980098116	0.0000019073	0.0000080000	0.0000000000
	152503	0.0000019073	0.0000090000	0.0000000000
	5000	0.0000019073	0.0000090000	0.0000010000
	1493283650	0.0000021458	0.0000080000	0.0000000000
	214826	0.0000028610	0.0000080000	0.0000000000
	1843349527	0.0000028610	0.0000080000	0.0000010000
	1360839354	0.0000019073	0.0000080000	0.0000000000
	2109248666	0.0000030994	0.0000080000	0.0000000000
	2147470852	0.0000019073	0.0000090000	0.0000000000
	0	0.0000021458	0.0000080000	0.0000000000
6000000	322486	0.0000030994	0.0000090000	0.0000000000
	14700764	0.0000030994	0.0000100000	0.0000000000
	3128036	0.0000030994	0.0000080000	0.0000000000
	6337399	0.0000030994	0.0000110000	0.0000010000
	61396	0.0000028610	0.0000090000	0.0000000000
	10393545	0.0000030994	0.0000090000	0.0000000000

	2147445644	0.0000019073	0.0000090000	0.0000010000
	1295390003	0.0000021458	0.0000080000	0.0000010000
	450057883	0.0000040531	0.0000110000	0.0000000000
	187645041	0.0000040531	0.0000100000	0.0000000000
	1980098116	0.0000030994	0.0000080000	0.0000000000
	152503	0.0000030994	0.0000090000	0.0000010000
	5000	0.0000028610	0.0000100000	0.0000010000
	1493283650	0.0000021458	0.0000080000	0.0000000000
	214826	0.0000030994	0.0000090000	0.0000010000
	1843349527	0.0000019073	0.0000080000	0.0000000000
	1360839354	0.0000019073	0.0000080000	0.0000000000
	2109248666	0.0000021458	0.0000080000	0.0000000000
	2147470852	0.0000019073	0.0000080000	0.0000000000
	0	0.0000030994	0.0000080000	0.0000000000
7000000	322486	0.0000028610	0.0000090000	0.0000000000
	14700764	0.0000030994	0.0000080000	0.0000000000
	3128036	0.0000028610	0.0000090000	0.0000010000
	6337399	0.0000028610	0.0000100000	0.0000010000
	61396	0.0000021458	0.0000080000	0.0000000000
	10393545	0.0000038147	0.0000090000	0.0000000000
	2147445644	0.0000019073	0.0000080000	0.0000000000
	1295390003	0.0000028610	0.0000090000	0.0000010000
	450057883	0.0000028610	0.0000090000	0.0000000000
	187645041	0.0000030994	0.0000090000	0.0000000000
	1980098116	0.0000021458	0.0000080000	0.0000000000
	152503	0.0000030994	0.0000100000	0.0000010000
	5000	0.0000030994	0.0000080000	0.0000010000
	1493283650	0.0000021458	0.0000080000	0.0000000000
	214826	0.0000028610	0.0000090000	0.0000028610
	1843349527	0.0000019073	0.0000090000	0.0000000000
	1360839354	0.0000030994	0.0000100000	0.0000000000
	2109248666	0.0000019073	0.0000080000	0.0000000000
	2147470852	0.0000021458	0.0000090000	0.0000000000

	0	0.0000021458	0.0000090000	0.0000000000
8000000	322486	0.0000030994	0.0000090000	0.0000000000
	14700764	0.0000019073	0.0000100000	0.0000000000
	3128036	0.0000030994	0.0000100000	0.0000010000
	6337399	0.0000030994	0.0000090000	0.0000010000
	61396	0.0000019073	0.0000090000	0.0000000000
	10393545	0.0000030994	0.0000080000	0.0000000000
	2147445644	0.0000019073	0.0000070000	0.0000010000
	1295390003	0.0000030994	0.0000090000	0.0000010000
	450057883	0.0000030994	0.0000090000	0.0000000000
	187645041	0.0000028610	0.0000090000	0.0000000000
	1980098116	0.0000019073	0.0000080000	0.0000000000
	152503	0.0000021458	0.0000090000	0.0000010000
	5000	0.0000028610	0.0000090000	0.0000000000
	1493283650	0.0000030994	0.0000090000	0.0000000000
	214826	0.0000028610	0.0000080000	0.0000010000
	1843349527	0.0000011921	0.0000080000	0.0000000000
	1360839354	0.0000028610	0.0000090000	0.0000010000
	2109248666	0.0000021458	0.0000080000	0.0000000000
	2147470852	0.0000019073	0.0000080000	0.0000010000
	0	0.0000021458	0.0000080000	0.0000010000
9000000	322486	0.0000028610	0.0000090000	0.0000000000
	14700764	0.0000059605	0.0000120000	0.0000000000
	3128036	0.0000028610	0.0000090000	0.0000000000
	6337399	0.0000028610	0.0000090000	0.0000010000
	61396	0.0000028610	0.0000080000	0.0000000000
	10393545	0.0000040531	0.0000100000	0.0000000000
	2147445644	0.0000019073	0.0000080000	0.0000000000
	1295390003	0.0000038147	0.0000090000	0.0000000000
	450057883	0.0000040531	0.0000090000	0.0000010000
	187645041	0.0000040531	0.0000090000	0.0000000000
	1980098116	0.0000021458	0.0000090000	0.0000010000
	152503	0.0000019073	0.0000090000	0.0000010000

	5000	0.0000030994	0.0000080000	0.0000000000
	1493283650	0.0000028610	0.0000100000	0.0000010000
	214826	0.0000019073	0.0000090000	0.0000000000
	1843349527	0.0000021458	0.0000100000	0.0000010000
	1360839354	0.0000030994	0.0000090000	0.0000000000
	2109248666	0.0000021458	0.0000090000	0.0000000000
	2147470852	0.0000019073	0.0000080000	0.0000010000
	0	0.0000030994	0.0000090000	0.0000000000
10000000	322486	0.0000021458	0.0000100000	0.0000010000
	14700764	0.0000040531	0.0000080000	0.0000010000
	3128036	0.0000119209	0.0000350000	0.0000020000
	6337399	0.0000040531	0.0000130000	0.0000010000
	61396	0.0000040531	0.0000100000	0.0000010000
	10393545	0.0000030994	0.0000090000	0.0000010000
	2147445644	0.0000021458	0.0000080000	0.0000000000
	1295390003	0.0000119209	0.0000330000	0.0000020000
	450057883	0.0000040531	0.0000100000	0.0000000000
	187645041	0.0000030994	0.0000090000	0.0000000000
	1980098116	0.0000038147	0.0000100000	0.0000000000
	152503	0.0000030994	0.0000090000	0.0000010000
	5000	0.0000030994	0.0000090000	0.0000000000
	1493283650	0.0000030994	0.0000100000	0.0000000000
	214826	0.0000030994	0.0000090000	0.0000000000
	1843349527	0.0000030994	0.0000100000	0.0000010000
	1360839354	0.0000030994	0.0000080000	0.0000000000
	2109248666	0.0000021458	0.0000090000	0.0000000000
	2147470852	0.0000019073	0.0000080000	0.0000000000
	0	0.0000030994	0.0000080000	0.0000010000

Búsqueda exponencial

Tamaño de problema n	**Números por buscar**	**Tiempo (seg)**		
		Tiempo real	*Tiempo CPU*	*Tiempo E/S*
1000000	322486	0.0000019073	0.0000100000	0.0000010000

	14700764	0.0000030994	0.0000100000	0.0000000000
	3128036	0.0000028610	0.0000080000	0.0000010000
	6337399	0.0000030994	0.0000080000	0.0000010000
	61396	0.0000019073	0.0000090000	0.0000010000
	10393545	0.0000030994	0.0000090000	0.0000010000
	2147445644	0.0000028610	0.0000090000	0.0000010000
	1295390003	0.0000019073	0.0000080000	0.0000010000
	450057883	0.0000019073	0.0000090000	0.0000000000
	187645041	0.0000030994	0.0000080000	0.0000010000
	1980098116	0.0000028610	0.0000090000	0.0000000000
	152503	0.0000019073	0.0000080000	0.0000000000
	5000	0.0000011921	0.0000080000	0.0000000000
	1493283650	0.0000019073	0.0000080000	0.0000010000
	214826	0.0000030994	0.0000070000	0.0000010000
	1843349527	0.0000021458	0.0000080000	0.0000000000
	1360839354	0.0000021458	0.0000080000	0.0000010000
	2109248666	0.0000028610	0.0000090000	0.0000000000
	2147470852	0.0000021458	0.0000090000	0.0000000000
	0	0.0000019073	0.0000080000	0.0000000000
2000000	322486	0.0000019073	0.0000090000	0.0000000000
	14700764	0.0000038147	0.0000130000	0.0000000000
	3128036	0.0000028610	0.0000090000	0.0000010000
	6337399	0.0000028610	0.0000080000	0.0000010000
	61396	0.0000019073	0.0000080000	0.0000010000
	10393545	0.0000030994	0.0000090000	0.0000000000
	2147445644	0.0000028610	0.0000090000	0.0000010000
	1295390003	0.0000030994	0.0000080000	0.0000000000
	450057883	0.0000028610	0.0000090000	0.0000000000
	187645041	0.0000038147	0.0000110000	0.0000010000
	1980098116	0.0000019073	0.0000080000	0.0000010000
	152503	0.0000019073	0.0000080000	0.0000000000
	5000	0.0000019073	0.0000080000	0.0000000000
	1493283650	0.0000030994	0.0000090000	0.0000010000

	214826	0.0000028610	0.0000090000	0.0000010000
	1843349527	0.0000019073	0.0000080000	0.0000010000
	1360839354	0.0000019073	0.0000080000	0.0000000000
	2109248666	0.0000019073	0.0000090000	0.0000010000
	2147470852	0.0000030994	0.0000080000	0.0000000000
	0	0.0000009537	0.0000080000	0.0000000000
3000000	322486	0.0000019073	0.0000090000	0.0000000000
	14700764	0.0000028610	0.0000090000	0.0000000000
	3128036	0.0000030994	0.0000080000	0.0000000000
	6337399	0.0000021458	0.0000090000	0.0000000000
	61396	0.0000021458	0.0000080000	0.0000000000
	10393545	0.0000028610	0.0000080000	0.0000010000
	2147445644	0.0000030994	0.0000090000	0.0000000000
	1295390003	0.0000019073	0.0000080000	0.0000010000
	450057883	0.0000040531	0.0000100000	0.0000010000
	187645041	0.0000040531	0.0000100000	0.0000000000
	1980098116	0.0000028610	0.0000090000	0.0000000000
	152503	0.0000021458	0.0000080000	0.0000010000
	5000	0.0000021458	0.0000070000	0.0000000000
	1493283650	0.0000030994	0.0000090000	0.0000010000
	214826	0.0000019073	0.0000080000	0.0000010000
	1843349527	0.0000021458	0.0000090000	0.0000000000
	1360839354	0.0000030994	0.0000080000	0.0000000000
	2109248666	0.0000030994	0.0000080000	0.0000010000
	2147470852	0.0000019073	0.0000090000	0.0000000000
	0	0.0000011921	0.0000070000	0.0000000000
4000000	322486	0.0000019073	0.0000080000	0.0000000000
	14700764	0.0000071526	0.0000130000	0.0000000000
	3128036	0.0000028610	0.0000090000	0.0000000000
	6337399	0.0000030994	0.0000090000	0.0000000000
	61396	0.0000028610	0.0000080000	0.0000000000
	10393545	0.0000030994	0.0000090000	0.0000010000
	2147445644	0.0000019073	0.0000080000	0.0000000000

	1295390003	0.0000021458	0.0000090000	0.0000010000
	450057883	0.0000028610	0.0000100000	0.0000000000
	187645041	0.0000050068	0.0000100000	0.0000010000
	1980098116	0.0000028610	0.0000080000	0.0000010000
	152503	0.0000021458	0.0000080000	0.0000000000
	5000	0.0000019073	0.0000090000	0.0000000000
	1493283650	0.0000030994	0.0000080000	0.0000000000
	214826	0.0000028610	0.0000080000	0.0000010000
	1843349527	0.0000028610	0.0000090000	0.0000000000
	1360839354	0.0000030994	0.0000080000	0.0000000000
	2109248666	0.0000028610	0.0000090000	0.0000000000
	2147470852	0.0000030994	0.0000080000	0.0000000000
	0	0.0000021458	0.0000090000	0.0000000000
5000000	322486	0.0000030994	0.0000090000	0.0000010000
	14700764	0.0000030994	0.0000090000	0.0000010000
	3128036	0.0000030994	0.0000090000	0.0000010000
	6337399	0.0000028610	0.0000090000	0.0000000000
	61396	0.0000021458	0.0000080000	0.0000010000
	10393545	0.0000028610	0.0000090000	0.0000000000
	2147445644	0.0000030994	0.0000090000	0.0000000000
	1295390003	0.0000030994	0.0000080000	0.0000000000
	450057883	0.0000040531	0.0000090000	0.0000010000
	187645041	0.0000038147	0.0000100000	0.0000010000
	1980098116	0.0000019073	0.0000090000	0.0000000000
	152503	0.0000021458	0.0000080000	0.0000010000
	5000	0.0000019073	0.0000080000	0.0000000000
	1493283650	0.0000030994	0.0000090000	0.0000000000
	214826	0.0000019073	0.0000080000	0.0000000000
	1843349527	0.0000019073	0.0000090000	0.0000010000
	1360839354	0.0000030994	0.0000090000	0.0000000000
	2109248666	0.0000021458	0.0000090000	0.0000000000
	2147470852	0.0000030994	0.0000100000	0.0000000000
	0	0.0000009537	0.0000090000	0.0000000000

6000000	322486	0.0000028610	0.0000080000	0.0000000000
	14700764	0.0000028610	0.0000090000	0.0000000000
	3128036	0.0000040531	0.0000080000	0.0000000000
	6337399	0.0000030994	0.0000090000	0.0000000000
	61396	0.0000019073	0.0000080000	0.0000000000
	10393545	0.0000028610	0.0000100000	0.0000010000
	2147445644	0.0000030994	0.0000090000	0.0000010000
	1295390003	0.0000028610	0.0000090000	0.0000000000
	450057883	0.0000030994	0.0000090000	0.0000000000
	187645041	0.0000040531	0.0000100000	0.0000000000
	1980098116	0.0000021458	0.0000090000	0.0000000000
	152503	0.0000021458	0.0000080000	0.0000000000
	5000	0.0000019073	0.0000080000	0.0000000000
	1493283650	0.0000021458	0.0000080000	0.0000000000
	214826	0.0000028610	0.0000080000	0.0000010000
	1843349527	0.0000030994	0.0000090000	0.0000000000
	1360839354	0.0000019073	0.0000090000	0.0000000000
	2109248666	0.0000030994	0.0000100000	0.0000000000
	2147470852	0.0000030994	0.0000100000	0.0000000000
	0	0.0000011921	0.0000080000	0.0000010000
7000000	322486	0.0000030994	0.0000090000	0.0000000000
	14700764	0.0000028610	0.0000090000	0.0000010000
	3128036	0.0000030994	0.0000080000	0.0000010000
	6337399	0.0000028610	0.0000100000	0.0000000000
	61396	0.0000028610	0.0000080000	0.0000000000
	10393545	0.0000028610	0.0000100000	0.0000000000
	2147445644	0.0000028610	0.0000090000	0.0000010000
	1295390003	0.0000028610	0.0000090000	0.0000000000
	450057883	0.0000030994	0.0000100000	0.0000010000
	187645041	0.0000040531	0.0000100000	0.0000000000
	1980098116	0.0000028610	0.0000090000	0.0000010000
	152503	0.0000028610	0.0000090000	0.0000010000
	5000	0.0000019073	0.0000090000	0.0000000000

	1493283650	0.0000030994	0.0000090000	0.0000000000
	214826	0.0000019073	0.0000090000	0.0000000000
	1843349527	0.0000028610	0.0000090000	0.0000010000
	1360839354	0.0000040531	0.0000090000	0.0000000000
	2109248666	0.0000028610	0.0000100000	0.0000010000
	2147470852	0.0000030994	0.0000090000	0.0000000000
	0	0.0000009537	0.0000090000	0.0000000000
8000000	322486	0.0000050068	0.0000290000	0.0000000000
	14700764	0.0000030994	0.0000080000	0.0000000000
	3128036	0.0000030994	0.0000090000	0.0000010000
	6337399	0.0000030994	0.0000100000	0.0000010000
	61396	0.0000030994	0.0000080000	0.0000000000
	10393545	0.0000030994	0.0000080000	0.0000010000
	2147445644	0.0000028610	0.0000090000	0.0000010000
	1295390003	0.0000040531	0.0000090000	0.0000000000
	450057883	0.0000040531	0.0000100000	0.0000000000
	187645041	0.0000030994	0.0000100000	0.0000010000
	1980098116	0.0000030994	0.0000090000	0.0000000000
	152503	0.0000028610	0.0000090000	0.0000010000
	5000	0.0000009537	0.0000080000	0.0000000000
	1493283650	0.0000050068	0.0000100000	0.0000000000
	214826	0.0000021458	0.0000080000	0.0000010000
	1843349527	0.0000019073	0.0000080000	0.0000000000
	1360839354	0.0000040531	0.0000100000	0.0000000000
	2109248666	0.0000028610	0.0000090000	0.0000000000
	2147470852	0.0000030994	0.0000090000	0.0000010000
	0	0.0000019073	0.0000080000	0.0000000000
9000000	322486	0.0000028610	0.0000110000	0.0000010000
	14700764	0.0000030994	0.0000100000	0.0000000000
	3128036	0.0000030994	0.0000090000	0.0000000000
	6337399	0.0000028610	0.0000090000	0.0000010000
	61396	0.0000030994	0.0000080000	0.0000000000
	10393545	0.0000030994	0.0000100000	0.0000000000

	2147445644	0.0000021458	0.0000080000	0.0000000000
	1295390003	0.0000040531	0.0000100000	0.0000000000
	450057883	0.0000028610	0.0000090000	0.0000000000
	187645041	0.0000040531	0.0000110000	0.0000000000
	1980098116	0.0000021458	0.0000090000	0.0000010000
	152503	0.0000019073	0.0000080000	0.0000010000
	5000	0.0000021458	0.0000080000	0.0000000000
	1493283650	0.0000040531	0.0000100000	0.0000010000
	214826	0.0000019073	0.0000090000	0.0000000000
	1843349527	0.0000040531	0.0000100000	0.0000000000
	1360839354	0.0000040531	0.0000100000	0.0000000000
	2109248666	0.0000030994	0.0000090000	0.0000000000
	2147470852	0.0000030994	0.0000090000	0.0000000000
	0	0.0000009537	0.0000080000	0.0000000000
10000000	322486	0.0000019073	0.0000100000	0.0000010000
	14700764	0.0000028610	0.0000100000	0.0000000000
	3128036	0.0000030994	0.0000090000	0.0000010000
	6337399	0.0000030994	0.0000090000	0.0000000000
	61396	0.0000028610	0.0000080000	0.0000000000
	10393545	0.0000030994	0.0000100000	0.0000010000
	2147445644	0.0000030994	0.0000100000	0.0000000000
	1295390003	0.0000040531	0.0000100000	0.0000010000
	450057883	0.0000040531	0.0000090000	0.0000000000
	187645041	0.0000040531	0.0000100000	0.0000000000
	1980098116	0.0000030994	0.0000100000	0.0000010000
	152503	0.0000019073	0.0000080000	0.0000010000
	5000	0.0000019073	0.0000080000	0.0000000000
	1493283650	0.0000040531	0.0000110000	0.0000000000
	214826	0.0000019073	0.0000090000	0.0000010000
	1843349527	0.0000040531	0.0000100000	0.0000000000
	1360839354	0.0000040531	0.0000110000	0.0000000000
	2109248666	0.0000030994	0.0000100000	0.0000000000
	2147470852	0.0000028610	0.0000090000	0.0000000000

	0	0.0000009537	0.0000070000	0.0000000000

Búsqueda de Fibonacci

Tamaño de problema n	**Números por buscar**	**Tiempo (seg)**		
		Tiempo real	*Tiempo CPU*	*Tiempo E/S*
1000000	322486	0.0000019073	0.0000080000	0.0000000000
	14700764	0.0000028610	0.0000090000	0.0000000000
	3128036	0.0000021458	0.0000090000	0.0000000000
	6337399	0.0000030994	0.0000080000	0.0000000000
	61396	0.0000030994	0.0000090000	0.0000000000
	10393545	0.0000028610	0.0000100000	0.0000010000
	2147445644	0.0000019073	0.0000080000	0.0000010000
	1295390003	0.0000021458	0.0000090000	0.0000000000
	450057883	0.0000019073	0.0000070000	0.0000010000
	187645041	0.0000030994	0.0000080000	0.0000000000
	1980098116	0.0000019073	0.0000090000	0.0000010000
	152503	0.0000021458	0.0000080000	0.0000000000
	5000	0.0000030994	0.0000080000	0.0000010000
	1493283650	0.0000021458	0.0000080000	0.0000000000
	214826	0.0000019073	0.0000080000	0.0000010000
	1843349527	0.0000021458	0.0000090000	0.0000000000
	1360839354	0.0000021458	0.0000080000	0.0000010000
	2109248666	0.0000028610	0.0000080000	0.0000010000
	2147470852	0.0000019073	0.0000080000	0.0000010000
	0	0.0000019073	0.0000080000	0.0000000000
2000000	322486	0.0000030994	0.0000100000	0.0000010000
	14700764	0.0000028610	0.0000100000	0.0000000000
	3128036	0.0000028610	0.0000100000	0.0000010000
	6337399	0.0000030994	0.0000090000	0.0000000000
	61396	0.0000028610	0.0000090000	0.0000000000
	10393545	0.0000028610	0.0000090000	0.0000010000
	2147445644	0.0000019073	0.0000080000	0.0000000000
	1295390003	0.0000021458	0.0000080000	0.0000000000

	450057883	0.0000019073	0.0000080000	0.0000010000
	187645041	0.0000028610	0.0000110000	0.0000000000
	1980098116	0.0000019073	0.0000090000	0.0000000000
	152503	0.0000030994	0.0000090000	0.0000000000
	5000	0.0000030994	0.0000090000	0.0000000000
	1493283650	0.0000019073	0.0000080000	0.0000010000
	214826	0.0000030994	0.0000080000	0.0000010000
	1843349527	0.0000021458	0.0000090000	0.0000000000
	1360839354	0.0000021458	0.0000110000	0.0000010000
	2109248666	0.0000021458	0.0000090000	0.0000010000
	2147470852	0.0000019073	0.0000080000	0.0000000000
	0	0.0000028610	0.0000080000	0.0000000000
3000000	322486	0.0000021458	0.0000080000	0.0000000000
	14700764	0.0000030994	0.0000080000	0.0000010000
	3128036	0.0000069141	0.0000090000	0.0000000000
	6337399	0.0000021458	0.0000090000	0.0000000000
	61396	0.0000030994	0.0000090000	0.0000010000
	10393545	0.0000030994	0.0000090000	0.0000010000
	2147445644	0.0000030994	0.0000080000	0.0000010000
	1295390003	0.0000019073	0.0000080000	0.0000000000
	450057883	0.0000028610	0.0000090000	0.0000010000
	187645041	0.0000040531	0.0000090000	0.0000000000
	1980098116	0.0000019073	0.0000090000	0.0000000000
	152503	0.0000030994	0.0000090000	0.0000010000
	5000	0.0000019073	0.0000120000	0.0000000000
	1493283650	0.0000021458	0.0000080000	0.0000010000
	214826	0.0000028610	0.0000100000	0.0000000000
	1843349527	0.0000019073	0.0000080000	0.0000000000
	1360839354	0.0000019073	0.0000080000	0.0000000000
	2109248666	0.0000021458	0.0000080000	0.0000010000
	2147470852	0.0000019073	0.0000090000	0.0000000000
	0	0.0000030994	0.0000090000	0.0000010000
4000000	322486	0.0000030994	0.0000080000	0.0000000000

	14700764	0.0000028610	0.0000090000	0.0000000000
	3128036	0.0000028610	0.0000090000	0.0000000000
	6337399	0.0000028610	0.0000090000	0.0000010000
	61396	0.0000030994	0.0000090000	0.0000010000
	10393545	0.0000030994	0.0000090000	0.0000010000
	2147445644	0.0000019073	0.0000080000	0.0000010000
	1295390003	0.0000019073	0.0000080000	0.0000010000
	450057883	0.0000038147	0.0000100000	0.0000000000
	187645041	0.0000038147	0.0000100000	0.0000000000
	1980098116	0.0000019073	0.0000090000	0.0000000000
	152503	0.0000030994	0.0000080000	0.0000010000
	5000	0.0000028610	0.0000090000	0.0000000000
	1493283650	0.0000028610	0.0000070000	0.0000000000
	214826	0.0000030994	0.0000090000	0.0000000000
	1843349527	0.0000028610	0.0000080000	0.0000000000
	1360839354	0.0000019073	0.0000080000	0.0000000000
	2109248666	0.0000050068	0.0000370000	0.0000010000
	2147470852	0.0000021458	0.0000080000	0.0000010000
	0	0.0000021458	0.0000080000	0.0000000000
5000000	322486	0.0000028610	0.0000100000	0.0000000000
	14700764	0.0000028610	0.0000100000	0.0000000000
	3128036	0.0000030994	0.0000090000	0.0000000000
	6337399	0.0000030994	0.0000100000	0.0000000000
	61396	0.0000030994	0.0000090000	0.0000000000
	10393545	0.0000028610	0.0000090000	0.0000000000
	2147445644	0.0000019073	0.0000090000	0.0000000000
	1295390003	0.0000028610	0.0000090000	0.0000000000
	450057883	0.0000040531	0.0000110000	0.0000010000
	187645041	0.0000030994	0.0000100000	0.0000000000
	1980098116	0.0000019073	0.0000080000	0.0000000000
	152503	0.0000028610	0.0000080000	0.0000010000
	5000	0.0000021458	0.0000090000	0.0000000000
	1493283650	0.0000030994	0.0000080000	0.0000010000

	214826	0.0000028610	0.0000100000	0.0000000000
	1843349527	0.0000021458	0.0000080000	0.0000000000
	1360839354	0.0000019073	0.0000090000	0.0000000000
	2109248666	0.0000021458	0.0000080000	0.0000010000
	2147470852	0.0000028610	0.0000090000	0.0000010000
	0	0.0000019073	0.0000080000	0.0000000000
6000000	322486	0.0000019073	0.0000080000	0.0000000000
	14700764	0.0000028610	0.0000090000	0.0000000000
	3128036	0.0000028610	0.0000100000	0.0000000000
	6337399	0.0000030994	0.0000090000	0.0000010000
	61396	0.0000030994	0.0000090000	0.0000000000
	10393545	0.0000028610	0.0000100000	0.0000000000
	2147445644	0.0000021458	0.0000090000	0.0000000000
	1295390003	0.0000030994	0.0000080000	0.0000000000
	450057883	0.0000038147	0.0000100000	0.0000000000
	187645041	0.0000030994	0.0000100000	0.0000000000
	1980098116	0.0000019073	0.0000080000	0.0000000000
	152503	0.0000030994	0.0000090000	0.0000010000
	5000	0.0000019073	0.0000090000	0.0000000000
	1493283650	0.0000019073	0.0000070000	0.0000000000
	214826	0.0000030994	0.0000090000	0.0000000000
	1843349527	0.0000019073	0.0000080000	0.0000010000
	1360839354	0.0000021458	0.0000080000	0.0000010000
	2109248666	0.0000021458	0.0000080000	0.0000010000
	2147470852	0.0000021458	0.0000080000	0.0000000000
	0	0.0000028610	0.0000090000	0.0000000000
7000000	322486	0.0000021458	0.0000090000	0.0000010000
	14700764	0.0000030994	0.0000100000	0.0000000000
	3128036	0.0000030994	0.0000090000	0.0000010000
	6337399	0.0000040531	0.0000100000	0.0000010000
	61396	0.0000028610	0.0000000000	0.0000130000
	10393545	0.0000030994	0.0000090000	0.0000000000
	2147445644	0.0000030994	0.0000080000	0.0000010000

	1295390003	0.0000030994	0.0000090000	0.0000010000
	450057883	0.0000040531	0.0000110000	0.0000000000
	187645041	0.0000038147	0.0000100000	0.0000000000
	1980098116	0.0000019073	0.0000090000	0.0000010000
	152503	0.0000019073	0.0000090000	0.0000000000
	5000	0.0000019073	0.0000100000	0.0000000000
	1493283650	0.0000030994	0.0000080000	0.0000000000
	214826	0.0000030994	0.0000090000	0.0000000000
	1843349527	0.0000019073	0.0000090000	0.0000000000
	1360839354	0.0000028610	0.0000090000	0.0000000000
	2109248666	0.0000019073	0.0000080000	0.0000000000
	2147470852	0.0000021458	0.0000080000	0.0000000000
	0	0.0000030994	0.0000090000	0.0000010000
8000000	322486	0.0000030994	0.0000090000	0.0000000000
	14700764	0.0000030994	0.0000090000	0.0000000000
	3128036	0.0000040531	0.0000110000	0.0000010000
	6337399	0.0000028610	0.0000090000	0.0000000000
	61396	0.0000028610	0.0000080000	0.0000010000
	10393545	0.0000028610	0.0000090000	0.0000000000
	2147445644	0.0000030994	0.0000090000	0.0000010000
	1295390003	0.0000040531	0.0000090000	0.0000000000
	450057883	0.0000030994	0.0000100000	0.0000000000
	187645041	0.0000030994	0.0000090000	0.0000010000
	1980098116	0.0000028610	0.0000090000	0.0000010000
	152503	0.0000028610	0.0000110000	0.0000000000
	5000	0.0000030994	0.0000080000	0.0000010000
	1493283650	0.0000038147	0.0000090000	0.0000000000
	214826	0.0000028610	0.0000090000	0.0000010000
	1843349527	0.0000021458	0.0000080000	0.0000000000
	1360839354	0.0000040531	0.0000100000	0.0000000000
	2109248666	0.0000019073	0.0000090000	0.0000010000
	2147470852	0.0000030994	0.0000100000	0.0000000000
	0	0.0000040531	0.0000100000	0.0000010000

9000000	322486	0.0000030994	0.0000100000	0.0000010000
	14700764	0.0000071526	0.0000090000	0.0000000000
	3128036	0.0000040531	0.0000100000	0.0000010000
	6337399	0.0000040531	0.0000100000	0.0000010000
	61396	0.0000030994	0.0000100000	0.0000010000
	10393545	0.0000028610	0.0000100000	0.0000000000
	2147445644	0.0000021458	0.0000080000	0.0000010000
	1295390003	0.0000040531	0.0000090000	0.0000000000
	450057883	0.0000030994	0.0000090000	0.0000000000
	187645041	0.0000040531	0.0000100000	0.0000010000
	1980098116	0.0000019073	0.0000090000	0.0000010000
	152503	0.0000030994	0.0000100000	0.0000000000
	5000	0.0000030994	0.0000080000	0.0000000000
	1493283650	0.0000028610	0.0000090000	0.0000000000
	214826	0.0000028610	0.0000090000	0.0000000000
	1843349527	0.0000030994	0.0000090000	0.0000010000
	1360839354	0.0000040531	0.0000100000	0.0000010000
	2109248666	0.0000030994	0.0000080000	0.0000000000
	2147470852	0.0000019073	0.0000090000	0.0000000000
	0	0.0000028610	0.0000090000	0.0000010000
10000000	322486	0.0000028610	0.0000090000	0.0000000000
	14700764	0.0000040531	0.0000100000	0.0000010000
	3128036	0.0000040531	0.0000090000	0.0000000000
	6337399	0.0000038147	0.0000100000	0.0000000000
	61396	0.0000030994	0.0000090000	0.0000000000
	10393545	0.0000028610	0.0000100000	0.0000000000
	2147445644	0.0000028610	0.0000090000	0.0000000000
	1295390003	0.0000040531	0.0000100000	0.0000000000
	450057883	0.0000040531	0.0000100000	0.0000010000
	187645041	0.0000040531	0.0000110000	0.0000000000
	1980098116	0.0000028610	0.0000100000	0.0000000000
	152503	0.0000040531	0.0000100000	0.0000000000
	5000	0.0000030994	0.0000090000	0.0000000000

	1493283650	0.0000030994	0.0000100000	0.0000000000
	214826	0.0000030994	0.0000100000	0.0000010000
	1843349527	0.0000038147	0.0000090000	0.0000000000
	1360839354	0.0000040531	0.0000110000	0.0000010000
	2109248666	0.0000030994	0.0000090000	0.0000000000
	2147470852	0.0000030994	0.0000080000	0.0000000000
	0	0.0000028610	0.0000090000	0.0000000000

Tiempos de búsqueda promedio.

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	1000000	0.013223958
Búsqueda en un árbol binario de búsqueda	1000000	0.0000054479
Búsqueda binaria o dicotómica	1000000	0.00000229633
Búsqueda exponencial	1000000	0.0000023961
Búsqueda de Fibonnacci	1000000	0.0000023603

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	2000000	0.030387545
Búsqueda en un árbol binario de búsqueda	2000000	0.0000058889
Búsqueda binaria o dicotómica	2000000	0.00000274806
Búsqueda exponencial	2000000	0.0000025272
Búsqueda de Fibonnacci	2000000	0.0000025391

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	3000000	0.03947228
Búsqueda en un árbol binario de búsqueda	3000000	0.0000063539
Búsqueda binaria o dicotómica	3000000	0.00000253474
Búsqueda exponencial	3000000	0.0000025868
Búsqueda de Fibonnacci	3000000	0.0000027656

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	4000000	0.05119293
Búsqueda en un árbol binario de búsqueda	4000000	0.0000060678
Búsqueda binaria o dicotómica	4000000	2.81082E-06
Búsqueda exponencial	4000000	0.0000029921
Búsqueda de Fibonnacci	4000000	0.0000028610

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	5000000	0.07311527
Búsqueda en un árbol binario de búsqueda	5000000	0.0000065446
Búsqueda binaria o dicotómica	5000000	2.69787E-06

Búsqueda exponencial	5000000	0.0000026703
Búsqueda de Fibonnacci	5000000	0.0000026822

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	6000000	0.07126689
Búsqueda en un árbol binario de búsqueda	6000000	0.0000061512
Búsqueda binaria o dicotómica	6000000	2.93628E-06
Búsqueda exponencial	6000000	0.0000027180
Búsqueda de Fibonnacci	6000000	0.0000025987

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	7000000	0.09378076
Búsqueda en un árbol binario de búsqueda	7000000	0.0000061989
Búsqueda binaria o dicotómica	7000000	2.78572E-06
Búsqueda exponencial	7000000	0.0000028491
Búsqueda de Fibonnacci	7000000	0.0000028133

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	8000000	0.09187337

Búsqueda en un árbol binario de búsqueda	8000000	0.0000065088
Búsqueda binaria o dicotómica	8000000	2.64767E-06
Búsqueda exponencial	8000000	0.0000031233
Búsqueda de Fibonnacci	8000000	0.0000031471

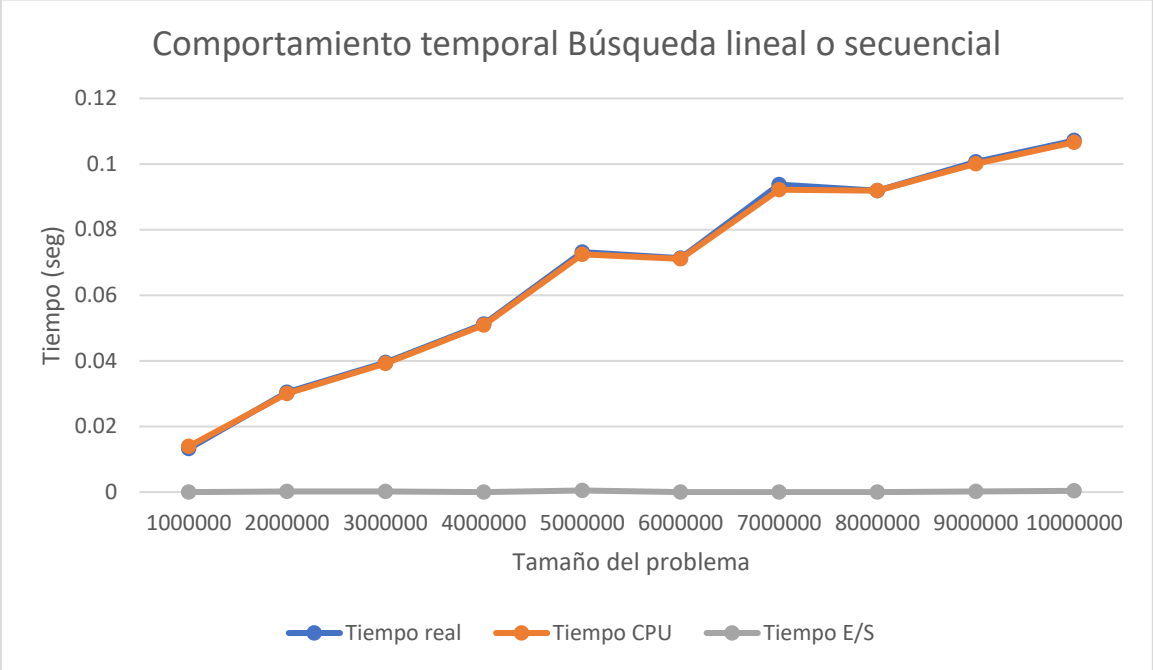
Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	9000000	0.10065819
Búsqueda en un árbol binario de búsqueda	9000000	0.0000064731
Búsqueda binaria o dicotómica	9000000	3.13707E-06
Búsqueda exponencial	9000000	0.0000029802
Búsqueda de Fibonnacci	9000000	0.0000033259

Algoritmos de búsqueda		
Algoritmo	**Tamaño de n**	**Tiempo promedio de todas las búsquedas**
Búsqueda lineal o secuencial	10000000	0.10714911
Búsqueda en un árbol binario de búsqueda	10000000	0.0000060559
Búsqueda binaria o dicotómica	10000000	4.21622E-06
Búsqueda exponencial	10000000	0.0000030040
Búsqueda de Fibonnacci	10000000	0.0000034451

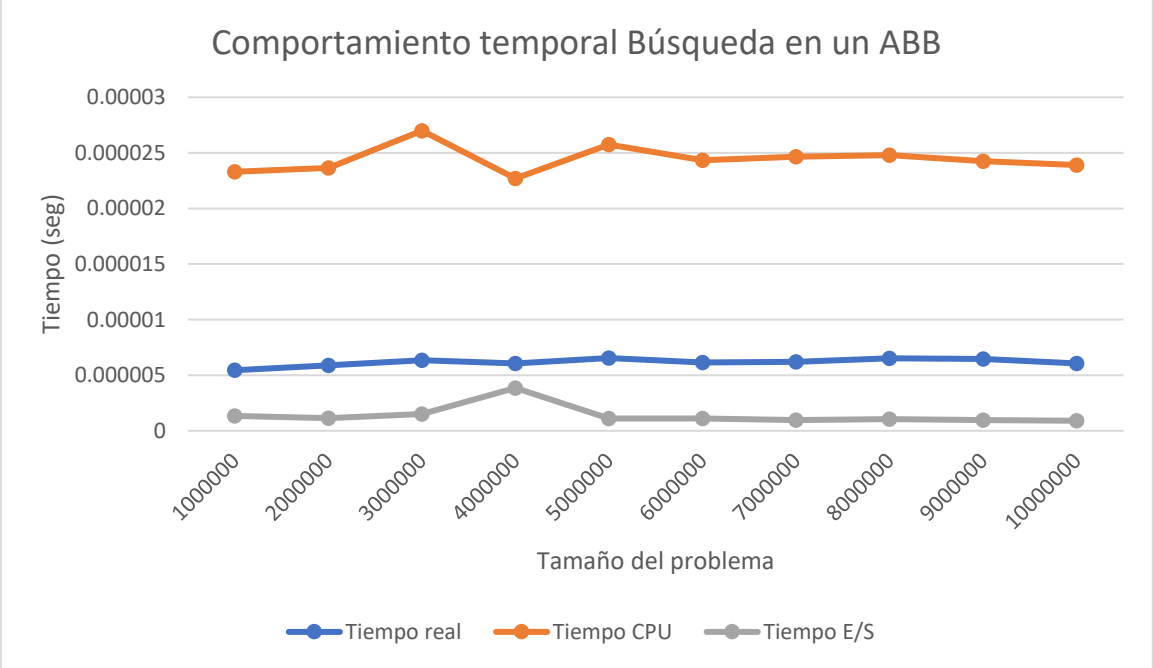
Gráfica del comportamiento temporal.

Cabe destacar que cada una de estas gráficas se hicieron con el tiempo promedio en segundos de todas las búsquedas, como las tablas de la sección anterior, solamente que en este caso no sólo se sacó tiempo promedio del tiempo real, sino también del CPU y del E/S.

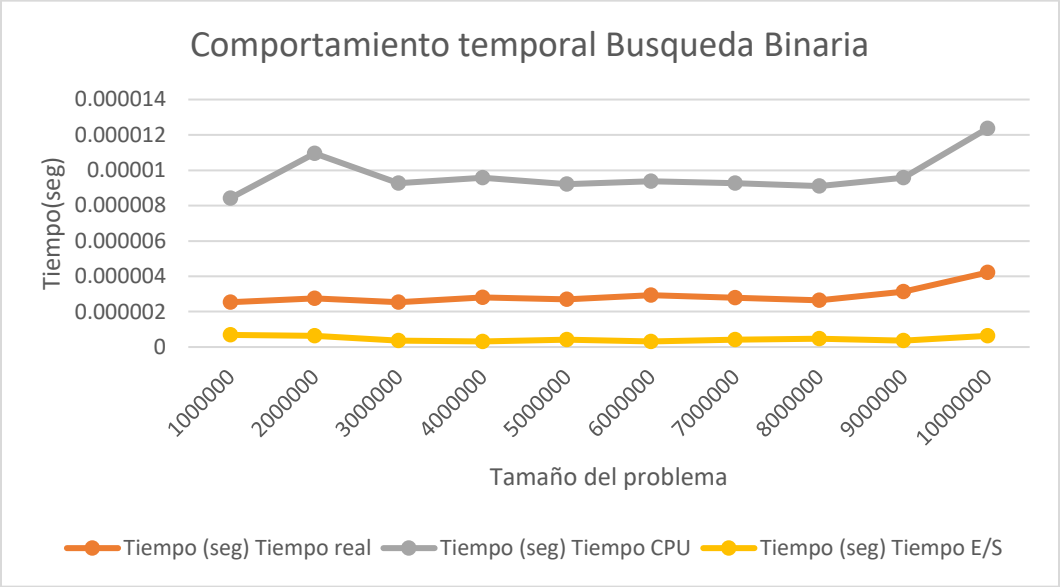
Búsqueda lineal o secuencial



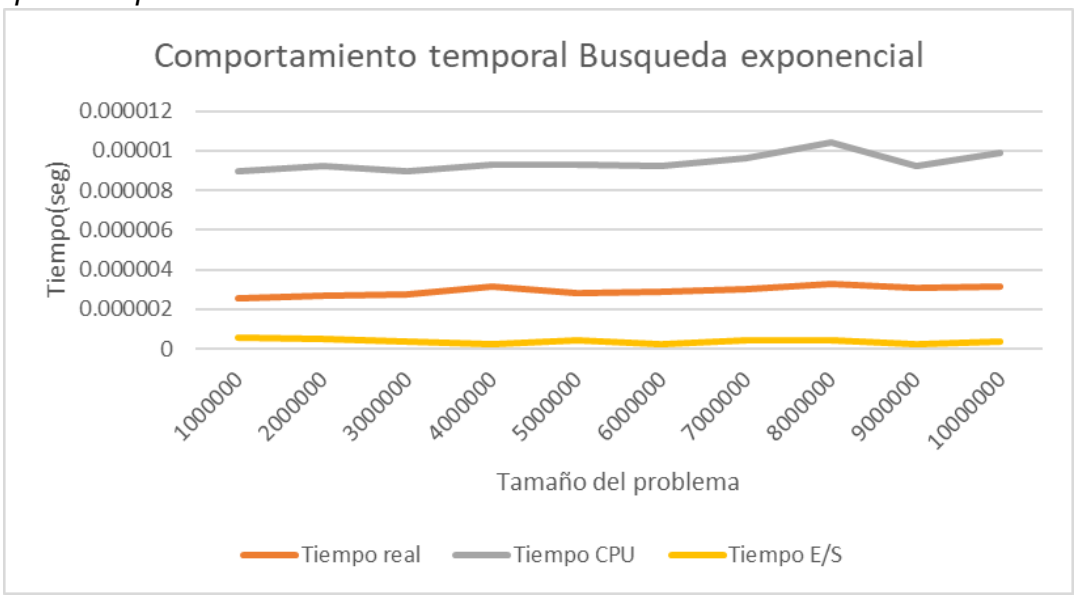
Búsqueda en un árbol binario de búsqueda



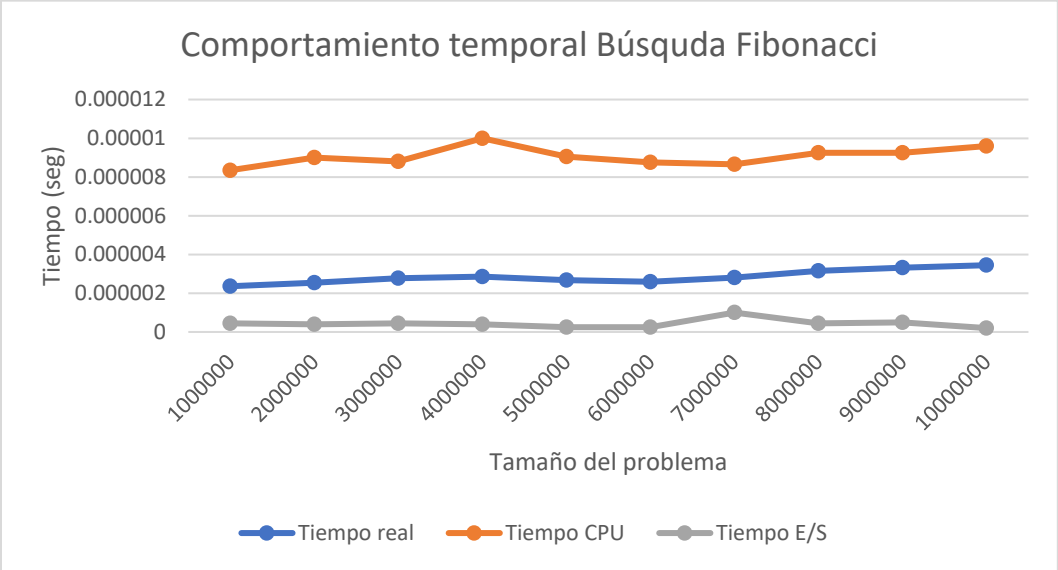
Búsqueda binaria o dicotómica



Búsqueda exponencial

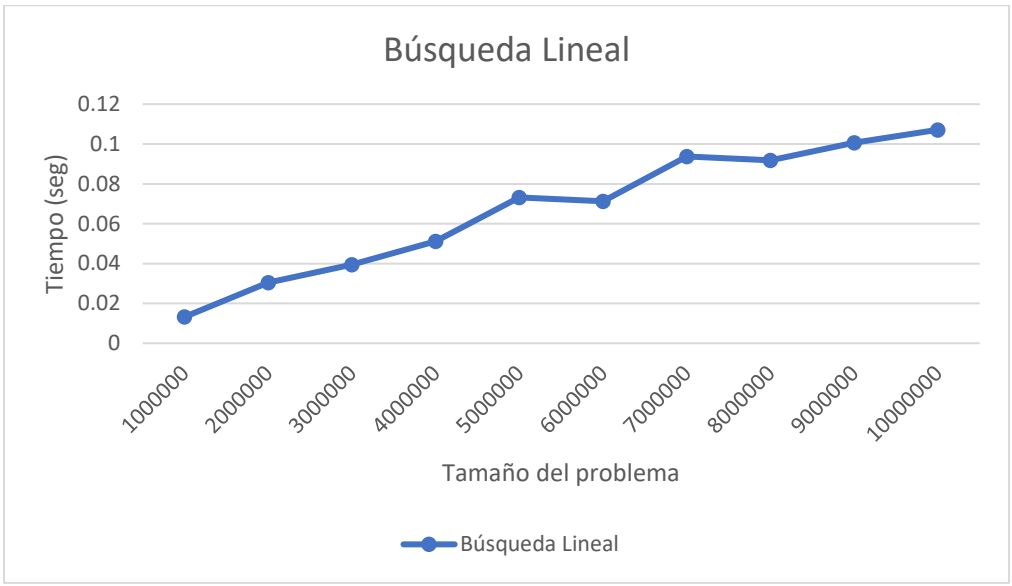


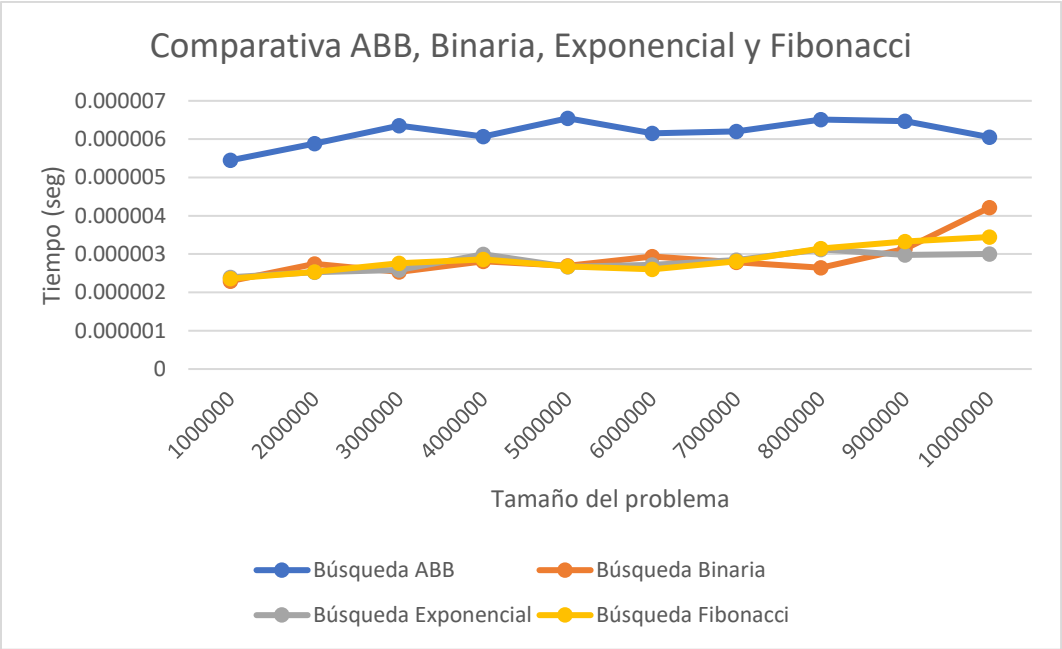
Búsqueda de Fibonnacci



Gráfica comparativa del comportamiento temporal.

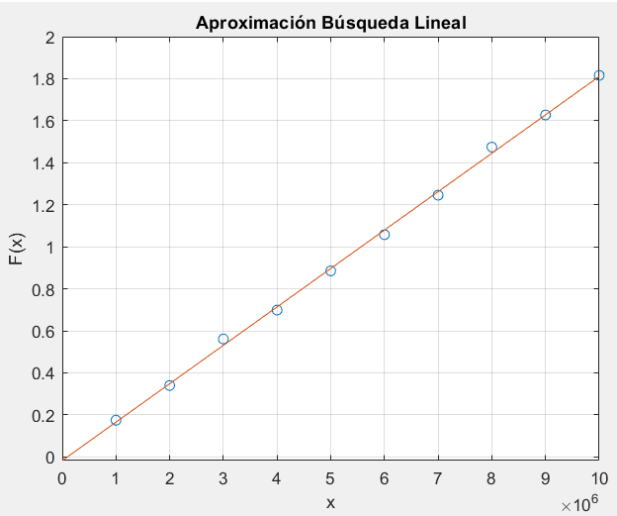
Aquí tenemos que hacerlo por partes, porque si lo hacemos todo junto la búsqueda lineal se “come” a los demás:



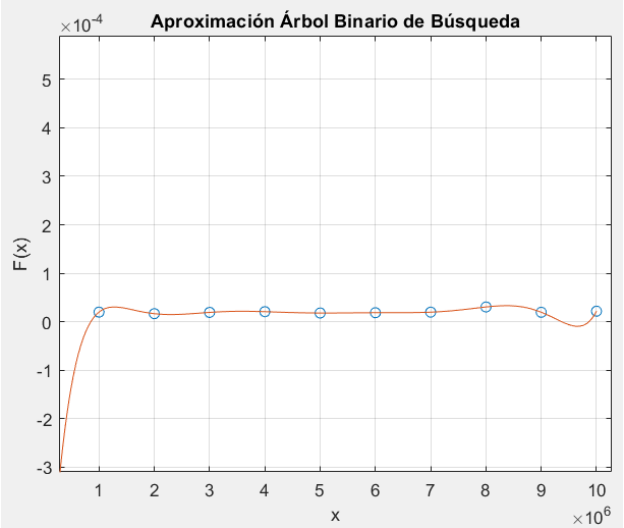


Aproximación del comportamiento temporal.

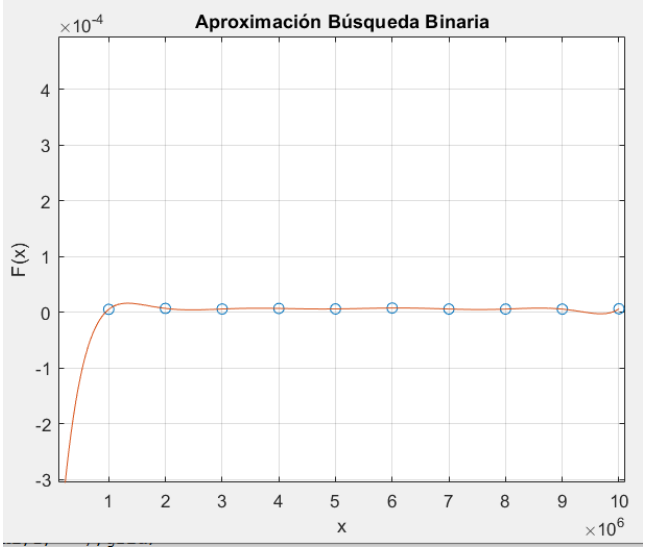
Búsqueda lineal o secuencial



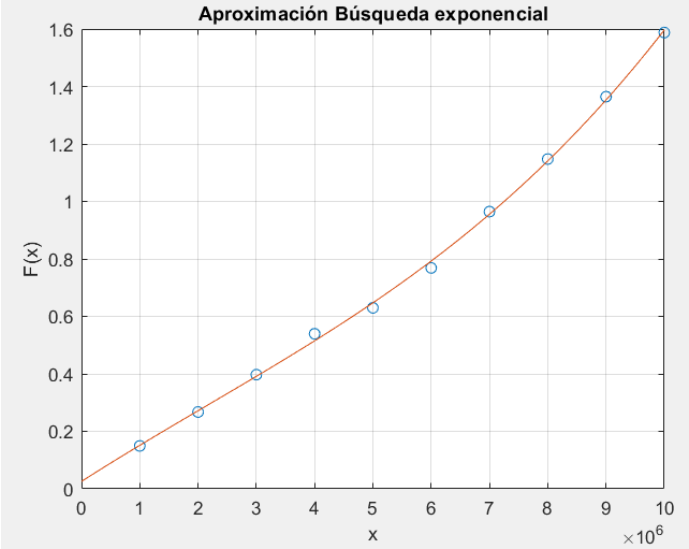
Búsqueda en un árbol binario de búsqueda



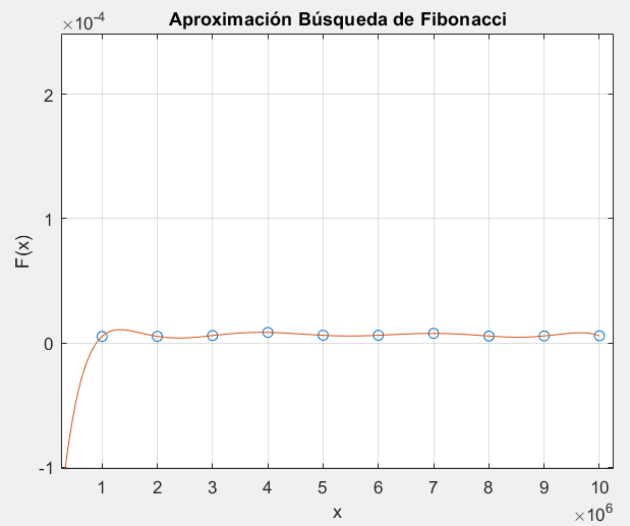
Búsqueda binaria o dicotómica



Búsqueda exponencial



Búsqueda de Fibonnacci



Aproximación a la función complejidad (peor caso).

Búsqueda lineal o secuencial

$$y = 0.0105n + 0.0097$$

Tamaño del problema (n)	Tiempo real
50000000	525000.0097
100000000	1050000.01
500000000	5250000.01
1000000000	10500000.01
5000000000	52500000.01

Búsqueda en un árbol binario de búsqueda

$$y = 6E-08n + 6E-06$$

Tamaño del problema (n)	Tiempo real
50000000	3.000006
100000000	6.000006
500000000	30.000006
1000000000	60.000006
5000000000	300.000006

Búsqueda binaria o dicotómica

$$y = 1E-07x + 2E-06$$

Tamaño del problema (n)	Tiempo real
50000000	5.000002
100000000	10.000002
500000000	50.000002
1000000000	100.000002
5000000000	500.000002

Búsqueda exponencial

$$y = 7E-08x + 3E-06$$

Tamaño del problema (n)	Tiempo real
50000000	3.500003
100000000	7.000003
500000000	35.000003
1000000000	70.000003
5000000000	350.000003

Búsqueda de Fibonnacci

$$y = 3E-10x6 - 9E-09x5 + 1E-07x4 - 8E-07x3 + 3E-06x2 - 4E-06x + 4E-06$$

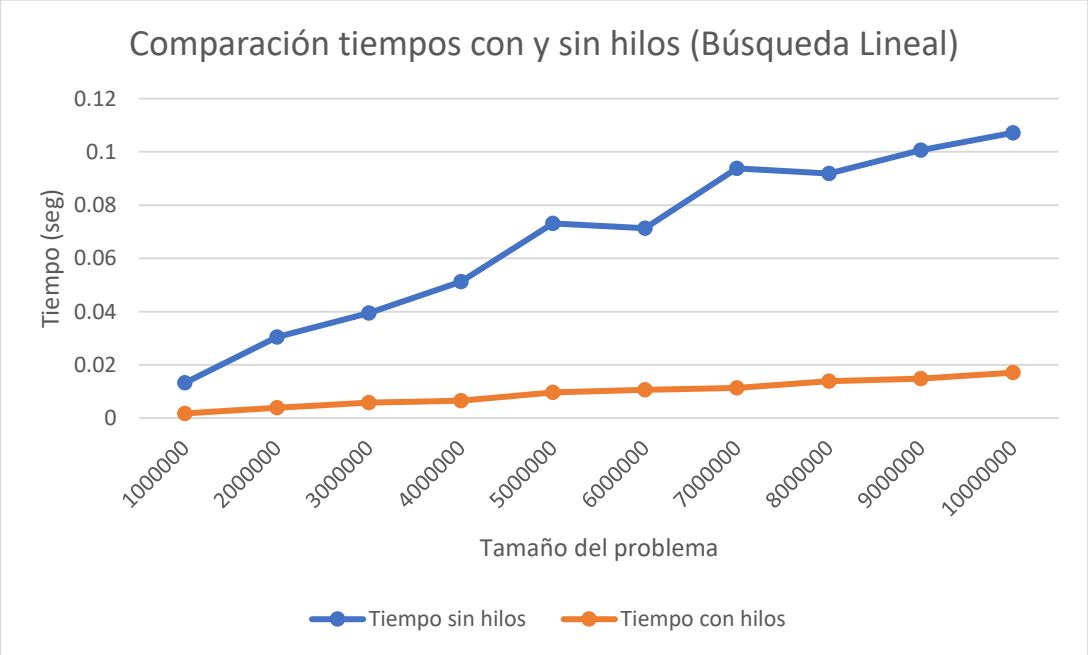
Tamaño del problema (n)	Tiempo real
50000000	4.6875E+36
100000000	3E+38
500000000	4.6875E+42
1000000000	3E+44
5000000000	4.6875E+48

Algoritmos con hilos de procesamiento (Threads).

Búsqueda lineal o secuencial

El arreglo de números que se tiene se divide a la mitad, se hacen 2 hilos y en cada uno se mete una mitad. Posteriormente, se procede a buscar en cada mitad de forma lineal.

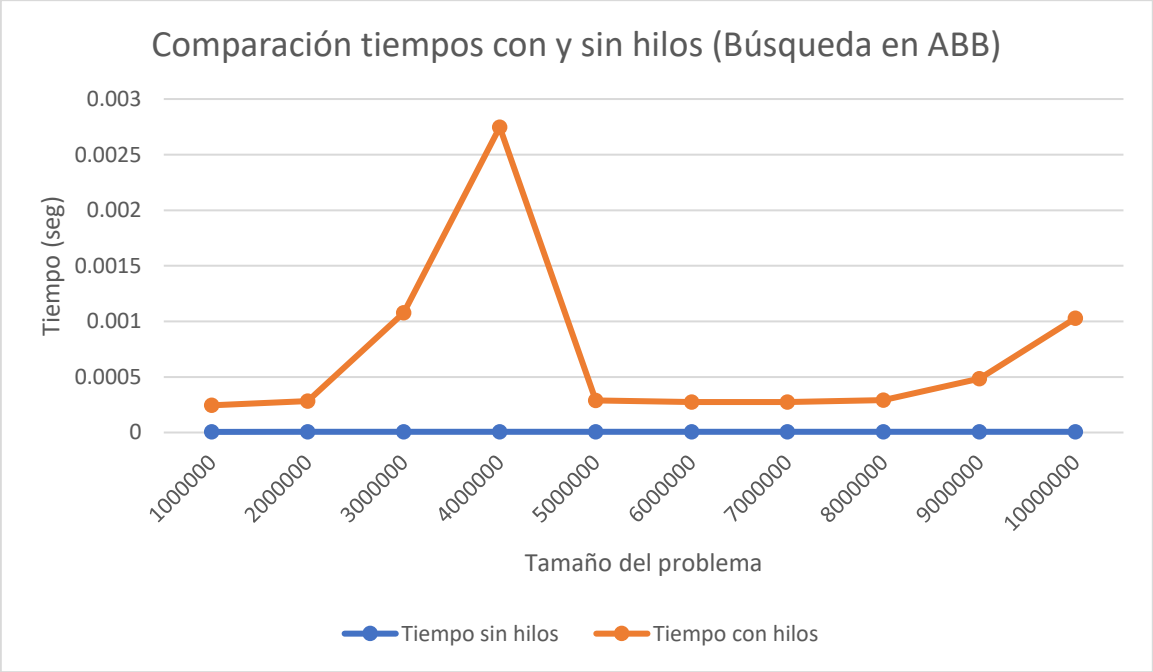
Tamaño de n	Tiempo real (sin hilos)	Tiempo real (con hilos)	Mejora (%)
1000000	0.013223958	0.001756417755	752.893665
2000000	0.030387545	0.003934752945	772.285971
3000000	0.03947228	0.005810546870	679.321256
4000000	0.05119293	0.006472778310	790.895772
5000000	0.07311527	0.009654319280	757.332215
6000000	0.07126689	0.010644197460	669.537466
7000000	0.09378076	0.011369991320	824.809425
8000000	0.09187337	0.013864457610	662.653907
9000000	0.10065819	0.014831984040	678.656272
10000000	0.10714911	0.017073869700	627.561952



Búsqueda en un árbol binario de búsqueda

Para este algoritmo se pensó en guardar en un hilo cada subárbol. Es decir, el árbol completo se va a segmentar en subárboles y éstos se guardarán en diferentes hilos, luego se buscará en cada subárbol el número deseado.

Tamaño de n	Tiempo real (sin hilos)	Tiempo real (con hilos)	Mejora (%)
1000000	0.0000054479	0.00024464	2.22689292
2000000	0.0000058889	0.00028249	2.08463828
3000000	0.0000063539	0.00107619	0.59040926
4000000	0.0000060678	0.00274701	0.22088734
5000000	0.0000065446	0.00028876	2.26644447
6000000	0.0000061512	0.00027511	2.23589589
7000000	0.0000061989	0.00027523	2.25225839
8000000	0.0000065088	0.00029204	2.22874399
9000000	0.0000064731	0.00048382	1.33790712
10000000	0.0000060559	0.00102798	0.58910824

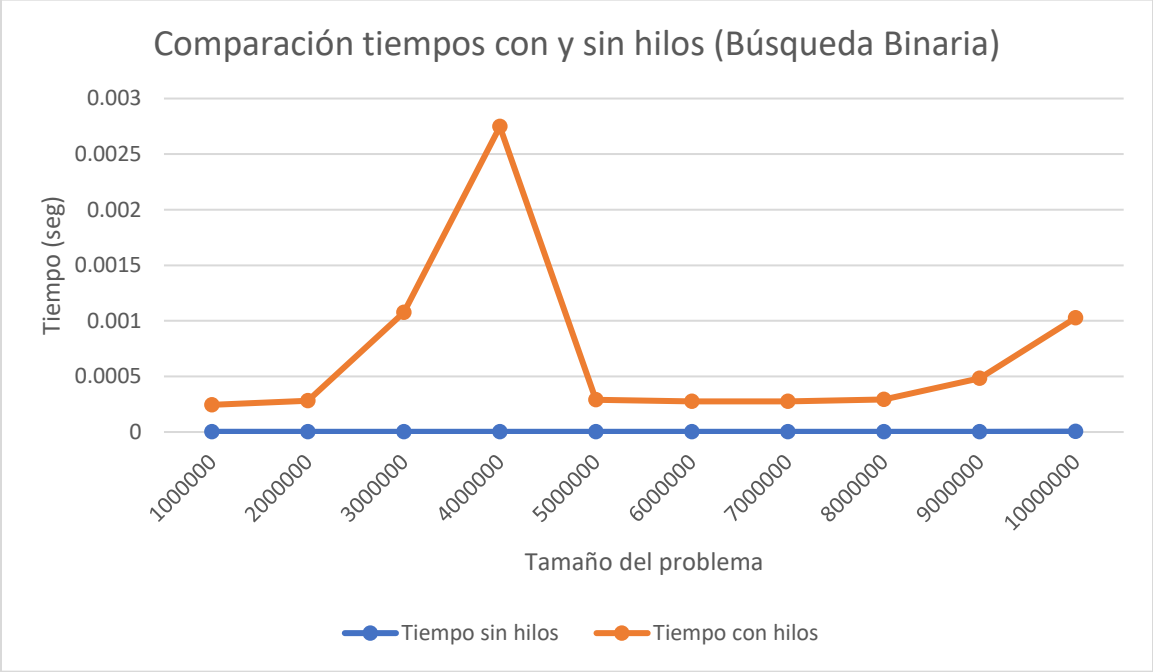


Búsqueda binaria o dicotómica

Se pensó en partir el arreglo en 4 diferentes “subarreglos” y cada uno meterlo en un hilo, posteriormente, se busca el número deseado.

Tamaño de n	Tiempo real (sin hilos)	Tiempo real (con hilos)	Mejora (%)
1000000	0.00000229633	0.000244641310	0.93865178
2000000	0.00000274806	0.000282490255	0.97279816

3000000	0.00000253474	0.001076185690	0.23552999
4000000	2.81082E-06	0.002747011180	0.10232285
5000000	2.69787E-06	0.000288760660	0.93429278
6000000	2.93628E-06	0.000275111200	1.0673066
7000000	2.78572E-06	0.000275230410	1.01214106
8000000	2.64767E-06	0.000292038925	0.90661545
9000000	3.13707E-06	0.000483822825	0.64839231
10000000	4.21622E-06	0.001027977470	0.41014712



Búsqueda exponencial

Se hace similar a como se hizo la búsqueda binaria, de hecho, se toma del código de la búsqueda binaria para poder realizar este. De igual forma dividimos en 4 el arreglo total y cada parte la metemos en un hilo.

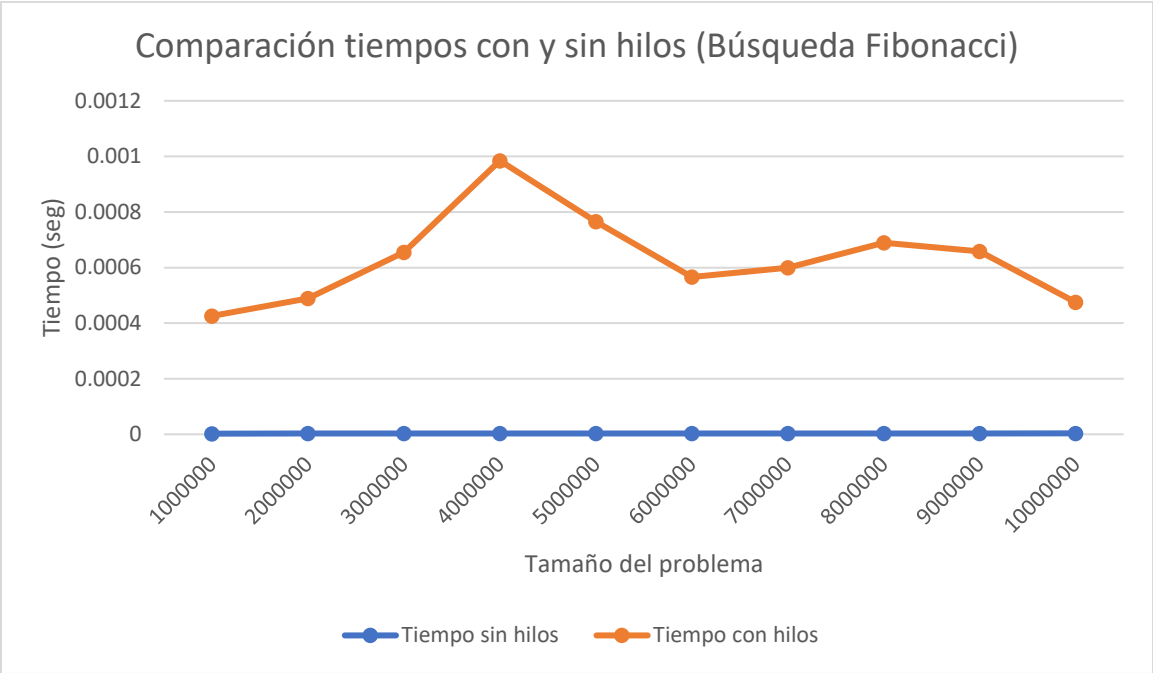
Tamaño de n	**Tiempo real (sin hilos)**	**Tiempo real (con hilos)**	**Mejora (%)**
1000000	0.0000023961	0.121994626525	0.0019641
2000000	0.0000025272	0.267581725120	0.00094446
3000000	0.0000025868	0.394236063965	0.00065616
4000000	0.0000029921	0.508967828745	0.00058788
5000000	0.0000026703	0.701796758170	0.00038049
6000000	0.0000027180	0.844309616095	0.00032192

7000000	0.0000028491	1.062385749785	0.00026818
8000000	0.0000031233	1.288877737520	0.00024233
9000000	0.0000029802	1.541325871470	0.00019335
10000000	0.0000030040	1.723877110900	0.00017426

Búsqueda de Fibonnacci

En esta ocasión, se mete en un hilo el arreglo y se empiezan a hacer las operaciones necesarias para poder encontrar el número deseado.

Tamaño de n	Tiempo real (sin hilos)	Tiempo real (con hilos)	Mejora (%)
1000000	0.0000023603	0.000425869000	0.55423147
2000000	0.0000025391	0.000488796540	0.51945949
3000000	0.0000027656	0.000655248260	0.42206903
4000000	0.0000028610	0.000984653280	0.29055913
5000000	0.0000026822	0.000765569536	0.35035354
6000000	0.0000025987	0.000565988650	0.45914348
7000000	0.0000028133	0.000599784955	0.46905145
8000000	0.0000031471	0.000689796422	0.45623606
9000000	0.0000033259	0.000658472100	0.50509353
10000000	0.0000034451	0.000475186640	0.72499934



Cuestionario.

1. ¿Cuál de los 5 algoritmos es más fácil de implementar?

R= El de búsqueda lineal. Realmente no se requirió mucho para poder implementarlo.

2. ¿Cuál de los 5 algoritmos es más difícil de implementar?

R= El de la búsqueda Fibonacci.

3. ¿Cuál de los 5 algoritmos fue el más difícil de modelar en su variante con hilos?

R= El de búsqueda Fibonacci, porque costó un poco más entenderlo para poder implementarlo de la mejor manera.

4. ¿Cuál de los 5 algoritmos en su variante con hilos resultó ser más rápido? ¿Por qué?

R= La búsqueda binaria porque al hacer la división del arreglo y meter cada parte en un hilo, se puede buscar en cada parte al mismo tiempo, lo que acelera mucho el proceso.

5. ¿Cuál de los 5 algoritmos en su variante con hilos no representó alguna ventaja? ¿Por qué?

R= La búsqueda exponencial, como vemos en las tablas no hubo mucho porcentaje de mejora. Tal vez se pudo implementar mejor.

6. ¿Cuál algoritmo tiene menor complejidad temporal?

R= La búsqueda binaria.

7. ¿Cuál algoritmo tiene mayor complejidad temporal?

R= La búsqueda lineal

8. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué?

R= Si, pero sinceramente en algunos casos, como se ve en las gráficas, salieron resultados no tan esperado. Probablemente hubo algún error en el análisis teórico.

9. ¿Sus resultados experimentales difieren mucho de los análisis teóricos que realizó? ¿A qué se debe?

R= No mucho, solo en algunos casos, tal vez no se tomó en cuenta algo en el análisis teórico.

10. En la versión con hilos, ¿usar hilos dividió el tiempo en C? ¿lo hizo c veces más rápido?

R= Si, en unos algoritmos si se notó el tiempo de mejora por mucho, pero en otros como en la búsqueda exponencial, realmente no varió mucho.

11. ¿Cuál es el % de mejora que tiene cada uno de los algoritmos en su variante con hilos? ¿Es lo que esperabas? ¿Por qué?

R= En promedio el % de mejora para las búsquedas lineal, en árbol binario de búsqueda, binaria, exponencial y Fibonacci son, 721.5947901, 1.60331859, 0.72281981, 0.000573313, 0.475119652, respectivamente. Si es lo que se esperaba, ya que hubo cierta mejora.

12. ¿Existió un entorno controlado para realizar las pruebas experimentales? ¿cuál fue?

R= Si, las especificaciones del equipo usado están en la sección de “Entorno de prueba”

13. Si sólo se realizara el análisis teórico de un algoritmo antes de implementarlo, ¿podrías asegurar cuál es el mejor?

R= No como tal, sí se tendría una aproximación buena, pero realmente en la teoría todo es ideal, ya cuando se lleva a la práctica tiene que ver mucho los recursos de las computadoras.

14. ¿Qué tan difícil fue realizar el análisis teórico de cada algoritmo?

R= Un poco, sobre todo en la parte de identificación de los casos.

15. ¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica?

R= Hacerla con bastante tiempo de anticipación, el uso de scripts y código para automatizar la formación de las tablas en archivos CSV y así solo copiar y pegar, y sea más rápido el llenado.

Anexos.

Anexo A. Códigos en ANSI C (sin hilos).

Búsqueda lineal o secuencial

```
/*
    ******************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Búsqueda Lineal o Secuencial
    Compilación: "gcc main.c tiempo.x  -o main (tiempo.c si se tiene la
implementación de la libreria o tiempo.o si solo se tiene el codigo
objeto)"
    Ejecución: "./main n" (Linux y MAC OS)
    NOTA: Si se hace desde un script.sh, solo se tienen que cambiar los
permisos para que pueda ejecutarse como programa dentro de la
computadora y en la terminal poner "./myscript.sh"
    ******************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tiempo.h"

int busquedaLineal(int *A, int x, int n);

//PROGRAMA PRINCIPAL
int main(int argc, char **argv){
      //Variables del programa
    double utime0, stime0, wtime0,utime1, stime1, wtime1; //para medir
el tiempo
    int index, x, n, *A; //del algoritmo

    //Si no se introducen exactamente 2 argumentos (Cadena de ejecución
y cadena=n)
      if (argc!=3){
            printf("\nIndique el tamaño del arreglo y el número a
buscar- Ejemplo: [user@equipo]$ %s 100 7\n",argv[0]);
            exit(1);
      }
      else{
            n=atoi(argv[1]); //tamanio del arreglo
        x=atoi(argv[2]);  //elemento a buscar
      }

    //Creacion del arreglo
      A = malloc(sizeof(int)*n);

    //Guardar números en el arreglo
      for(int i=0; i<n; i++){
        scanf("%d",&A[i]);
    }
```

```
    //Iniciar conteo para evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);

    //Algoritmo
    index=busquedaLineal(A,x,n);
    printf("%d\n", index);

    //Evaluar tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

      //Cálculo del tiempo de ejecución del programa
      printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");


printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++");

    exit(0);
}

int busquedaLineal(int *A, int x, int n){
    int i; //se asigna variable para recorrer el arreglo
    for(i=0;i<n;i++){  //se recorre el arreglo
        if(x==A[i]){  //si el valor buscado corresponde con uno del
arreglo
            return i;  //devuelve el índice de donde esta ubicado el
numero
        }
    }
    return -1; //sino devuelve -1
}
```

Búsqueda en un árbol binario de búsqueda

```
/*
    ************************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Búsqueda en ABB
```

```
    Compilación: "gcc main.c tiempo.x  -o main (tiempo.c si se tiene la
implementación de la libreria o tiempo.o si solo se tiene el codigo
objeto)"
    Ejecución: "./main n" (Linux y MAC OS)
    NOTA: Si se hace desde un script.sh, solo se tienen que cambiar los
permisos para que pueda ejecutarse como programa dentro de la
computadora y en la terminal poner "./myscript.sh"
    ************************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "tiempo.h"

//se crea la estructura del árbol con un valor y 2 apuntadores a la
izquierda y derecha
typedef struct arbolABB{
    int valor;
    struct arbolABB *izquierda;
    struct arbolABB *derecha;
}arbolABB;

arbolABB *crearNodo(int valor);
void imprimirTabulacion(int numTabs);
void imprimirArbol_recursivo(arbolABB *raiz, int nivel);
void imprimirArbol(arbolABB *raiz);
bool insertar(arbolABB **raizptr, int valor, int n);
bool encontrarNumero(arbolABB *raiz, int valor, int n);

int main(int argc, char **argv){
    //Variables del programa
    double utime0, stime0, wtime0,utime1, stime1, wtime1; //para medir
el tiempo
    //del algoritmo
    arbolABB *miRaiz = NULL;
    int *A, x, n;

    //Si no se introducen exactamente 2 argumentos (Cadena de ejecución
y cadena=n)
      if (argc!=3){
            printf("\nIndique el tamaño del arreglo y el número a
buscar- Ejemplo: [user@equipo]$ %s 100 7\n",argv[0]);
            exit(1);
      }
      else{
            n=atoi(argv[1]); //tamanio del arreglo
        x=atoi(argv[2]);  //elemento a buscar
      }

    //creacion arreglo
    A = malloc(sizeof(int)*n);

    //Guardar números en el arreglo
    for(int i=0; i<n; i++){
        scanf("%d",&A[i]);
        //printf("%d\n", A[i]);
        insertar(&miRaiz, A[i], n);
```

```
    }

    imprimirArbol(miRaiz);

    //Iniciar conteo para evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);

    //Algoritmo de busqueda
    printf("%d (%d)\n", x, encontrarNumero(miRaiz, x, n));

    //Evaluar tiempos de ejecución
    uswtime(&utime1, &stime1, &wtime1);

      //Cálculo del tiempo de ejecución del programa
      printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");


printf("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++");
    exit(0);
}

//función para crear un nodo del arbo, recibe el valor que ira dentro
arbolABB *crearNodo(int valor){
    arbolABB* nodo = malloc(sizeof(arbolABB)); //se crea el espacio en
la memoria
    if(nodo != NULL){
        //se igualan los 2 aputnadores de izquiera y derecha a nulo, y
el valor de la estructura al valor entrante
        nodo->izquierda=NULL;
        nodo->derecha=NULL;
        nodo->valor=valor;
    }
    return nodo;
}

//función para que al imprimir el arbol se hagan las tabulaciones
necesarias
void imprimirTabulacion(int numTabs){
    for(int i=0; i<numTabs; i++){
```

```
        printf("\t");
    }
}

//función para imprimir el arbol
void imprimirArbol_recursivo(arbolABB *raiz, int nivel){
    //si la raiz esta vacia, se hace una tabulacion y se imprime vacio
    if(raiz==NULL){
        imprimirTabulacion(nivel);
        printf("VACIO \n");
        return;
    }
    //sino esta vacio igual imprime una tabulacion y el valor, y de
nuevo imprime otra tabulacion
    imprimirTabulacion(nivel);
    printf("Valor: %d\n", raiz->valor);
    imprimirTabulacion(nivel);
    printf("izq: \n");

    //vamos por el lado izquierdo y hacemos lo mismo, solo aumentamos
el nivel
    imprimirArbol_recursivo(raiz->izquierda, nivel+1);

    imprimirTabulacion(nivel);
    printf("der: \n");

    //vamos hacia el lado derecho e imprimimos también
    imprimirArbol_recursivo(raiz->derecha, nivel+1);

    imprimirTabulacion(nivel);
    printf("listo\n");
}

void imprimirArbol(arbolABB *raiz){
    imprimirArbol_recursivo(raiz,0);
}

//funcion para insertar
bool insertar(arbolABB **raizptr, int valor, int n){
    arbolABB *raiz = *raizptr;
    if(raiz==NULL){
        //arbol vacío, se crea un nuevo nodo
        (*raizptr)=crearNodo(valor);
        return true;
    }
    //si el valor a insertar ya esta
    if(valor==raiz->valor){
        //hacer nada
        return false;
    }

    //vamos desde 0 hasta la altura del arbol
    //se hace lo mismo que vimos anteriormente, solo que del lado
izquierdo y derecho
    for(int i=0;i<n;i++){
        if(valor<raiz->valor){
            raizptr=&(raiz->izquierda);
            raiz=*raizptr;
            if(raiz==NULL){
```

```
                (*raizptr)=crearNodo(valor);
                return true;
            }
            if(valor==raiz->valor){
                //hacer nada
                return false;
            }
            //return insertar(&(raiz->izquierda), valor);
        }else{
            raizptr=&(raiz->derecha);
            raiz=*raizptr;
            if(raiz==NULL){
                (*raizptr)=crearNodo(valor);
                return true;
            }
            if(valor==raiz->valor){
                //hacer nada
                return false;
            }
            //return insertar(&(raiz->derecha), valor);
        }
    }
}

bool encontrarNumero(arbolABB *raiz, int valor, int n){
    /*
        Estos 2 primeros if se refieren a la raiz del arbol, si la raiz
esta vacía, no se encuentra el valor
        Si el apuntador de la raíz a valor en la estructura del arbol
es igual al valor buscado, se encontró
    */
    if(raiz==NULL){
        printf("Valor NO encontrado\n");
        return false;
    }
    if(raiz->valor==valor){
        printf("Valor encontrado\n");
        return true;
    }
    /*
        Aquí nos vamos al lado izquierdo, que es cuando el valor a
buscar es menor que el valor en la raíz.
        Recorremos toda la parte del lado izquierdo, y de igual forma
si en el nodo no hay ningun valor (NULL)
        pues no se encuentra el valor, pero si el valor del nodo es
igual al buscado, pues regresa un true
    */
    for(int i=0; i<n; i++){
        if(valor<raiz->valor){
            raiz=raiz->izquierda;
            if(raiz==NULL) {
                printf("Valor NO encontrado\n");
                return false;
            }
            if(raiz->valor==valor){
                printf("Valor encontrado\n");
                return true;
            }
            //return encontrarNumero(raiz->izquierda, valor);
```

```
        }
        /*
            Lo mismo que se hizo en el lado izquierdo se hace en el
derecho. Se va a pasar al lado derecho
            cuando el valor sea mayor.
        */
        else{
            raiz=raiz->derecha;
            if(raiz==NULL) {
                printf("Valor NO encontrado\n");
                return false;
            }
            if(raiz->valor==valor){
                printf("Valor encontrado\n");
                return true;
            }
            //return encontrarNumero(raiz->derecha, valor);
        }
    }
}
```

Búsqueda binaria o dicotómica

```
/*
    ************************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Binaria
    Compilación: "gcc main.c tiempo.x  -o main (tiempo.c si se tiene la
implementación de la libreria o tiempo.o si solo se tiene el codigo
objeto)"
    Ejecución: "./main n" (Linux y MAC OS)
    ************************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "tiempo.h"

int busquedaBinariaMini(int array[], int n, int x);
int busquedaBinaria(int array[], int l, int r, int x);
int busquedaExponencial(int arr[], int n, int x);
int BusquedaDeFibonacci(int array[], int n, int x);
int *array;
int min(int a, int b);


#define TRUE 1
#define FALSE 0
```

```
#define TAMANO_A 20


int num; //elementos en el arreglo, indicará el número de enteros a
buscar
double utime0, stime0, wtime0, utime1, stime1, wtime1; //Variables para
medición de tiempos


int main(int argc, char *argv[]){


      if (argc > 2) {
        int algoritmoBusqueda; //parámetro que representa que algoritmo
de busqueda que se utilizará
        int valorABuscar, valorEncontrado;
        //se obtienen los primeros 3 parametros escritos en consola
(los volvemos enteros)
        //---------- se ingresa algoritmo -> tamaño -> valorabuscar----
----------//
        algoritmoBusqueda = atoi(argv[1]);
        num = atoi(argv[2]);
        valorABuscar = atoi(argv[3]);


        array = malloc(sizeof(int) * num);
        int i;


        for (i = 0; i < num; ++i) { //recoge los valores del .txt
gracias al redireccionamiento en la consola (<)
            scanf("%d", &array[i]);
        }




printf("==========================================================
=======\n");
        switch (algoritmoBusqueda) {
case 3: //busqueda binaria
                printf("Busqueda binaria\n");
                valorEncontrado = busquedaBinariaMini(array, num,
valorABuscar);
```

```
                uswtime(&utime1, &stime1, &wtime1);//Evaluar los
tiempos de ejecución
                break;
  default:
                exit(0);
                break;
        }
//printf("\nContenido del arreglo:\n");
        //ImprimeArr(array);
        //printf("Num[%d]: %d.\n", num, array[num-1]);
        printf("\nTamaño del arreglo (n): %d\n", num);
        printf("\nElemento a buscar (x): %d\n", valorABuscar);
        printf("Coincidencias: %s\n", valorEncontrado != -1 ? "Si" :
"No");


        //Mostrar los tiempos con 10 decimas
        printf("\n");
        printf("real (Tiempo total)  %.10f s\n", wtime1 - wtime0);
        printf("user (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
        printf("sys (Tiempo en acciónes de E/S)  %.10f s\n", stime1 -
stime0);
        printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
        printf("\n");


        //Mostrar los tiempos en formato exponecial
        printf("\n");
        printf("real (Tiempo total)  %.10e s\n", wtime1 - wtime0);
        printf("user (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
        printf("sys (Tiempo en acciónes de E/S)  %.10e s\n", stime1 -
stime0);
        printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
        printf("\n");
    }


    return 0;
}
```

```
int busquedaBinariaMini(int array[], int n, int x){
    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);


    int l = 0;//izquierda
    int r = n-1;//derecha
    return busquedaBinaria(array, l, r, x);


}


int busquedaBinaria(int array[], int l, int r, int x){
    //l lado inferior y r lado superior
    while (l <= r) {
        int m = l + (r - l) / 2;

        // Checa que el valor a buscar esté en medio y retorna su
posicion
        if (array[m] == x)
            return m;

        // Si el valor a buscar es mayor, se ignora el resto de la
mitad de la izquierda
        if (array[m] < x)
            l = m + 1;

        // Si el valor a buscar es menor, se ignora el resto de la
mitad de la derecha
        else
            r = m - 1;
    }

    return -1; // Si no se encuentran coincidencias se retorna -1
}
```

Búsqueda exponencial

```
/*
    ************************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Exponencial
    Compilación: "gcc main.c tiempo.x  -o main (tiempo.c si se tiene la
implementación de la libreria o tiempo.o si solo se tiene el codigo
objeto)"
```

```
    Ejecución: "./main n" (Linux y MAC OS)
    ************************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "tiempo.h"


int busquedaBinariaMini(int array[], int n, int x);
int busquedaBinaria(int array[], int l, int r, int x);
int busquedaExponencial(int arr[], int n, int x);
int BusquedaDeFibonacci(int array[], int n, int x);
int *array;
int min(int a, int b);


#define TRUE 1
#define FALSE 0
#define TAMANO_A 20


int num; //elementos en el arreglo, indicará el número de enteros a
buscar
double utime0, stime0, wtime0, utime1, stime1, wtime1; //Variables para
medición de tiempos


int main(int argc, char *argv[]){


      if (argc > 2) {
        int algoritmoBusqueda; //parámetro que representa que algoritmo
de busqueda que se utilizará
        int valorABuscar, valorEncontrado;
        //se obtienen los primeros 3 parametros escritos en consola
(los volvemos enteros)
        //---------- se ingresa algoritmo -> tamaño -> valorabuscar----
-----------//
        algoritmoBusqueda = atoi(argv[1]);
        num = atoi(argv[2]);
        valorABuscar = atoi(argv[3]);
```

```
        array = malloc(sizeof(int) * num);
        int i;


        for (i = 0; i < num; ++i) { //recoge los valores del .txt
gracias al redireccionamiento en la consola (<)
            scanf("%d", &array[i]);
        }




printf("================================================================
=======\n");
        switch (algoritmoBusqueda) {
case 4: //busqueda exponencial
                printf("Busqueda exponencial\n");
                valorEncontrado = busquedaExponencial(array, num,
valorABuscar);
                uswtime(&utime1, &stime1, &wtime1);//Evaluar los
tiempos de ejecución
                break;
default:
                exit(0);
                break;
        }
//printf("\nContenido del arreglo:\n");
        //ImprimeArr(array);
        //printf("Num[%d]: %d.\n", num, array[num-1]);
        printf("\nTamaño del arreglo (n): %d\n", num);
        printf("\nElemento a buscar (x): %d\n", valorABuscar);
        printf("Coincidencias: %s\n", valorEncontrado != -1 ? "Si" :
"No");


        //Mostrar los tiempos con 10 decimas
        printf("\n");
        printf("real (Tiempo total)  %.10f s\n", wtime1 - wtime0);
        printf("user (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
        printf("sys (Tiempo en acciónes de E/S)  %.10f s\n", stime1 -
stime0);
        printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
        printf("\n");
```

```
        //Mostrar los tiempos en formato exponecial
        printf("\n");
        printf("real (Tiempo total)  %.10e s\n", wtime1 - wtime0);
        printf("user (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
        printf("sys (Tiempo en acciónes de E/S)  %.10e s\n", stime1 -
stime0);
        printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
        printf("\n");
    }


    return 0;
}
int busquedaExponencial(int array[], int n, int x){
    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);


    // Checa si el valor a buscar se encuentra en la primera posición
    if (array[0] == x)
        return 0;

    // Se calcula un rango para la busqueda binaria elevando 2^n su
paso y verificando
    //que el index sea menor o igual al valor a encontrar
    int i = 1;
    while (i < n && array[i] <= x)
        i = i*2;

    //Se hace una busqueda binaria con el rango calculado.
    return busquedaBinaria(array, i/2, min(i, n-1), x);
}
```

Búsqueda de Fibonnacci

```
/*
    ***************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda de Fibonacci
```

```
    Compilación: "gcc main.c tiempo.x  -o main (tiempo.c si se tiene la
implementación de la libreria o tiempo.o si solo se tiene el codigo
objeto)"
    Ejecución: "./main n" (Linux y MAC OS)
    ************************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "tiempo.h"


int busquedaBinariaMini(int array[], int n, int x);
int busquedaBinaria(int array[], int l, int r, int x);
int busquedaExponencial(int arr[], int n, int x);
int BusquedaDeFibonacci(int array[], int n, int x);
int *array;
int min(int a, int b);


#define TRUE 1
#define FALSE 0
#define TAMANO_A 20


int num; //elementos en el arreglo, indicará el número de enteros a
buscar
double utime0, stime0, wtime0, utime1, stime1, wtime1; //Variables para
medición de tiempos


int main(int argc, char *argv[]){


      if (argc > 2) {
        int algoritmoBusqueda; //parámetro que representa que algoritmo
de busqueda que se utilizará
        int valorABuscar, valorEncontrado;
        //se obtienen los primeros 3 parametros escritos en consola
(los volvemos enteros)
        //---------- se ingresa algoritmo -> tamaño -> valorabuscar----
----------//
        algoritmoBusqueda = atoi(argv[1]);
        num = atoi(argv[2]);
```

```
        valorABuscar = atoi(argv[3]);


        array = malloc(sizeof(int) * num);
        int i;


        for (i = 0; i < num; ++i) { //recoge los valores del .txt
gracias al redireccionamiento en la consola (<)
            scanf("%d", &array[i]);
        }




printf("================================================================
=======\n");
        switch (algoritmoBusqueda) {
  case 5: //busqueda de Fibonacci
                printf("Busqueda de Fibonacci\n");
                valorEncontrado = BusquedaDeFibonacci(array, num,
valorABuscar);
                uswtime(&utime1, &stime1, &wtime1);//Evaluar los
tiempos de ejecución
                break;
            default:
                exit(0);
                break;
        }


        //printf("\nContenido del arreglo:\n");
        //ImprimeArr(array);
        //printf("Num[%d]: %d.\n", num, array[num-1]);
        printf("\nTamaño del arreglo (n): %d\n", num);
        printf("\nElemento a buscar (x): %d\n", valorABuscar);
        printf("Coincidencias: %s\n", valorEncontrado != -1 ? "Si" :
"No");


        //Mostrar los tiempos con 10 decimas
        printf("\n");
        printf("real (Tiempo total)  %.10f s\n", wtime1 - wtime0);
        printf("user (Tiempo de procesamiento en CPU) %.10f s\n",
utime1 - utime0);
```

```
        printf("sys (Tiempo en acciónes de E/S)  %.10f s\n", stime1 -
stime0);
        printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
        printf("\n");


        //Mostrar los tiempos en formato exponecial
        printf("\n");
        printf("real (Tiempo total)  %.10e s\n", wtime1 - wtime0);
        printf("user (Tiempo de procesamiento en CPU) %.10e s\n",
utime1 - utime0);
        printf("sys (Tiempo en acciónes de E/S)  %.10e s\n", stime1 -
stime0);
        printf("CPU/Wall   %.10f %% \n", 100.0 * (utime1 - utime0 +
stime1 - stime0) / (wtime1 - wtime0));
        printf("\n");
    }


    return 0;
}


int BusquedaDeFibonacci(int array[], int n, int x){
    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento
    uswtime(&utime0, &stime0, &wtime0);


    /* Inicializa los primeros números de Fibonacci */
    int fibMMm2 = 0; // (m-2)ésimo numero de Fibonacci.
    int fibMMm1 = 1; // (m-1)ésimo numero de Fibonacci.
    int fibM = fibMMm2 + fibMMm1; // (m)ésimo numero de Fibonacci

    /* la variable fibM se encargará de guardar el número de Fibonacci
más
    pequeño que sea menor o igual a n */
    while (fibM < n) {
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }

    // Marca el rango eliminado del frente
    int compensacion = -1;
```

```
    /*
       El while se cumplira mientras haya elementos con los que se
puede operar
       Dentro del while se compara que el valor de arr[fibMm2] con el
valor a buscar
       Cuándo fibM toma el valor de 1, fibMm2 toma el valor de 0  */
    while (fibM > 1) {
        // Verifica que fibMm2 isea una valida locacion
        int i = min(compensacion + fibMMm2, n - 1);

        /* Si el número buscar es mayor que el valor en el index de
fibMm2,
        se corta el rango del arreglo desde la compensasion hasta i*/
        if (array[i] < x) {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            compensacion = i;
        }

        /* Si el número buscar es menor que el valor en el index de
fibMm2,
           se corta el rango después de i+1  */
        else if (array[i] > x) {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }

        /* El valor a buscar es encontrado y se retorna su posicion */
        else
            return i;
    }

    /* Compara el ultimo elemento con el numero a encontrar */
    if (fibMMm1 && array[compensacion + 1] == x)
        return compensacion + 1;

    // Si no se encuentran coincidencias se retorna -1
    return -1;
}


int min(int a, int b){
```

```
    return (a < b ? a : b);// retorna el número más pequeño a través de
un ternario
}
```

Anexo B. Códigos en ANSI C (con hilos).

Búsqueda lineal o secuencial

```
/*
    ************************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Búsqueda Lineal o Secuencial
    Compilación: "gcc main.c tiempo.x  -o main (tiempo.c si se tiene la
implementación de la libreria o tiempo.o si solo se tiene el codigo
objeto)"
    Ejecución: "./main n" (Linux y MAC OS)
    ************************************************************************
*/

//*************************************************************************
//LIBRERIAS INCLUIDAS
//*************************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>//libreria de hilos
#include "tiempo.h"
//*************************************************************************
//DECLARACIONES
//*************************************************************************
#define min(X, Y) (((X) < (Y)) ? (X) : (Y))
//*************************************************************************
***********
//DECLARACION DE ESTRUCTURAS
//*************************************************************************
***********
struct parametros{// Nuestro nodo del árbol contendrá hijo izquierdo,
derecho y un número entero
    int iz;
    int de;
    int x;
};
//*************************************************************************
//DECLARACIÓN DE FUNCIONES
//*************************************************************************
void* linealSearch(void *todo);
//*************************************************************************
//VARIABLES GLOBALES
//*************************************************************************
int *A, encontrado=0;//Apuntador a int para hacer arreglo
//*************************************************************************
//PROGRAMA PRINCIPAL
//*************************************************************************
int main(int argc, char *argv[])
{
      //*************************************************************************
***
      //Variables del main
```

```
    //*****************************************************************
***
    double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables
para medición de tiempos
    //Variables globales
    int n, i, x, result, medio;
    //Hilos
    pthread_t h[2];

//*****************************************************************
    //Recepción y decodificación de argumentos
    //Si no se recibe el argumento, se cierra el programa
    if(argc!=3)
        exit(1);
        //Numero elementos en el arreglo
    n=atoi(argv[1]);// Identifica el número de datos sobre los que se
va a trabajar
    x=atoi(argv[2]);// Asigna el número que se va a buscar
    printf("\nx = %i\nn = %i\n",x,n);
    //Se crea el arreglo
    A = malloc(n*sizeof(int));
    //Guardando los numeros
    for(i=0;i<n;i++)
        scanf("%i",&A[i]);
    //Para sacar la mitad
    medio=n/2;
    //Creacón deStructs para los hilos
    struct parametros *todo1 = (struct parametros
*)malloc(sizeof(struct parametros));
    todo1 -> iz = 0;
    todo1-> de = medio;
    todo1-> x = x;
    struct parametros *todo2 = (struct parametros
*)malloc(sizeof(struct parametros));
    todo2-> iz = medio-1;
    todo2-> de = n;
    todo2-> x = x;

//*****************************************************************
    //Iniciar el conteo del tiempo para las evaluaciones de rendimiento

//*****************************************************************
    uswtime(&utime0, &stime0, &wtime0);

//*****************************************************************
    pthread_create(&h[0],NULL,linealSearch,(void *)todo1);
    pthread_create(&h[1],NULL,linealSearch,(void *)todo2);
    //Se espera a que terminen
    for(int i = 0;i < 2;i++)
        pthread_join(h[i],NULL);
    //Se imprime si se encontró o no
    (encontrado == 1)? printf("Y\n") : printf("N\n");

//*****************************************************************
    //Evaluar los tiempos de ejecución

//*****************************************************************
    uswtime(&utime1, &stime1, &wtime1);
    //Cálculo del tiempo de ejecución del programa
```

```
    printf("\nTiempo = %.10f s\n\n",  wtime1 - wtime0);

//*********************************************************************
    for(i = 0;i < 27;i++)
        printf("*");// Adjuntamos una division para el resultado de la
búsqueda de cada número

    printf("\n");

    return 0;
}
/*
Descripción: Función encargada de buscar nuestro número
Recibe: Todos los argumentos que se comprimieron con anterioridad
*/
void* linealSearch(void *todo)
{
    //Se descomprime lo que se mando
    struct parametros *recibido =  (struct parametros
*)malloc(sizeof(struct parametros));
    recibido = (struct parametros *)todo;
    int iz = recibido -> iz;
    int de = recibido -> de;
    int x = recibido->x;
    for(int i = iz; i < de ; i++)//creamos un for que recora el arreglo
    {
        if(A[i] == x)//identifica si el número buscado se encuentra en
el arreglo
        {
            encontrado=1;//si el valor fue encontrado nuestra varible
"encontrado" imprimira el valor de 1
            return;
        }
    }
    return;
}
```

Búsqueda en un árbol binario de búsqueda

```
/*
    *********************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Búsqueda Lineal o Secuencial
    Compilación: "gcc main.c tiempo.x  -o main (tiempo.c si se tiene la
implementación de la libreria o tiempo.o si solo se tiene el codigo
objeto)"
    Ejecución: "./main n" (Linux y MAC OS)
    *********************************************************************
*/

//*********************************************************************
//LIBRERIAS INCLUIDAS
//*********************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <pthread.h>
```

```
#include <string.h>
#include "tiempo.h"
//*****************************************************************
//DEFINICION DE CONSTANTES DEL PROGRAMA
//*****************************************************************

//*****************************************************************
***********
//DECLARACION DE ESTRUCTURAS
//*****************************************************************
***********
typedef struct node
                                        // Nuestro nodo del árbol contendrá
hijo izquierdo, derecho y un número entero
{
      struct node *left, *right;
      int number;
}node;
//DEFINICIONES DE SINÓNIMOS
typedef node* position;
                                  // La posición será la dirección hacia
un nodo específico
typedef position arbol_bin;
arbol_bin mainTree;
//*****************************************************************
//DECLARACIÓN DE FUNCIONES
//*****************************************************************
void * search(void *root);
void initialize(arbol_bin *BinaryTree);
void insert(arbol_bin * BinaryTree, int newNumber);
//void subTrees(position aux[], arbol_bin * BinaryTree, int n);
//void imprimeTiempos(bool found, int keyNumber, int nSize, double
RealTime, double UserTime,double SysTime);
int Vacio(arbol_bin *a);
//*****************************************************************
//VARIABLES GLOBALES
//*****************************************************************
bool found = false; //variable que indicará si se encontró el número
en la búsqueda
int* Data; //apuntador de entero que será inicializado como arreglo
para los datos donde se buscará
int nSize = 0; //variable que tomará el tamaño de la línea de comando
int nThreads = 0;  //variable que tomará el número de hilos
int keyNumber = 0; //variable que indicará el número a buscar
//*****************************************************************
//PROGRAMA PRINCIPAL
//*****************************************************************
int main (int argc, char* argv[])
{
      //*************************************************************
***
      //Variables del main
      //*************************************************************
***
      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables
para medición de tiempos
```

```
	int n;	//n determina el tamaño del algorito dado por
argumento al ejecutar
	int i; //Variables para loops

	//*****************************************************************
***
	//Recepción y decodificación de argumentos
	//*****************************************************************
***

	if (argc < 4) exit(0);  // Verificacion sencilla

	nSize = atoi(argv[1]);  // Identifica el numero de datos sobre los
que se va a trabajar
	nThreads = atoi(argv[2]);	// Toma el numero de hilos a
trabajar
	keyNumber = atoi(argv[3]);	// Asigna el numero que se va a
buscar

	Data = (int*)calloc(nSize,sizeof(int));  // Inicializacion del
arreglo para los numeros

	for (i = 0; i < nSize; ++i){
		scanf("%d", Data+i);		// Insertamos los
numeros en el arreglo
	}

	initialize(&mainTree);  // Iniciamos nuestro ABB para usarlo

	for(i = 0; i < nSize; ++i){
		insert(&mainTree, Data[i]);  // Insertamos los datos en el
ABB
	}

	//*****************************************************************
***
	//Iniciar el conteo del tiempo para las evaluaciones de
rendimiento
	//*****************************************************************
***
	uswtime(&utime0, &stime0, &wtime0);
	//*****************************************************************
***

	//*****************************************************************
***
	position aux[nThreads]; // Arreglo de posiciones a partir de las
cuales se realizar	la b	squeda
	for (i = 0; i < nThreads; ++i)
	{
		aux[i] = NULL;
	}

	//subTrees(aux, &mainTree, nThreads);

	/*switch(nThreads){	// Switch-Case con base al n	mero de
hilos

		case 2:	// En caso de 2 hilos
```

```
            // Iniciaremos la b□squeda en los dos sub□rboles de la
ra□z
                  aux[0] = mainTree->left;
                  aux[1] = mainTree->right;
                  break;

            case 3:     // En caso de 3 hilos
            // Los dos primeros hilos iniciar□n en los dos sub□rboles
del sub□rbol izquierdo
                  aux[0] = (mainTree->left)->left;
                  aux[1] = (mainTree->left)->right;
            // El □ltimo hilo iniciar□ en el sub□rbol derecho
                  aux[2] = mainTree->right;
                  break;

            case 4:     // En caso de 4 hilos
            // Iniciaremos la b□squeda en los sub□rboles de los dos
hijos de la ra□z
                  aux[0] = (mainTree->left)->left;
                  aux[1] = (mainTree->left)->right;
                  aux[2] = (mainTree->right)->left;
                  aux[3] = (mainTree->right)->right;
                  break;

      }*/

      // En caso de que el n□mero se encuentre en la ra□z o en sus dos
hijos, no iniciamos las b□squedas
      if(!found){
            pthread_t *aThreads;    // Declaramos un arreglo de hilos
            aThreads = (pthread_t*) malloc(nThreads*sizeof(pthread_t));
      // Inicializaci□n del arreglo de hilos

            for (i = 0; i < nThreads; ++i)
            {
                  pthread_create(&aThreads[i], NULL, search,
(void*)aux[i]);   // Crear los hilos con el comportamiento "segmentar"
            }

            for (i = 0; i < nThreads; ++i)
            {
                  pthread_join(aThreads[i], NULL);    // Se verifica la
finalizaci□n de todos los hilos
            }

            free(aThreads);   // Liberamos el arreglo de hilos
      }

      //*****************************************************************
***
      //Evaluar los tiempos de ejecución
      //*****************************************************************
***
      uswtime(&utime1, &stime1, &wtime1);

      //Cálculo del tiempo de ejecución del programa
      printf("\n");
```

```
      printf("real  (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user  (Tiempo de procesamiento en CPU) %.10f s\n",  utime1
- utime0);
      printf("sys  (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real  (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user  (Tiempo de procesamiento en CPU) %.10e s\n",  utime1
- utime0);
      printf("sys  (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");
      //*********************************************************************
***

      //imprimeTiempos(found, keyNumber, nSize, RealTime, UserTime,
SysTime); // Funci□n que mostrar□ los resultados
      free(Data); // Liberamos el arreglo de n□meros
      //Terminar programa normalmente
      exit (0);
}

//*************************************************************************
***
//DEFINICIÓN DE FUNCIONES
//*************************************************************************
***
/*
Descripci□n: Funci□n encargada de buscar en los n□meros por medio de
un □rbol Binario de B□squeda
Recibe: void * root (Ser□ el apuntador a la ra□z sobre la que vamos a
buscar)
*/
void * search(void *root){
position aux = (position)root;      // Posici□n auxiliar que nos
permitir□ movernos en el ABB

      while(aux != NULL && !found){ // Iteraci□n que durar□ hasta que
ya no haya elementos donde buscar o se haya encontrado el n□mero en
otro hilo

            if(aux->number == keyNumber){ // En caso de encontrar el
n□mero
                  found = true;     // Indicamos que fue encontrado
            }

            if(aux->number > keyNumber){  // En caso de que el n□mero
en el que estamos sea mayor que el buscado
                  aux = aux->left;  // Nos moveremos al sub□rbol
izquierdo
```

```
            }
            else{ // En caso contrario
                  aux = aux->right; // Nos movemos al sub□rbol derecho
            }
      }
      pthread_exit(NULL);     // Salimos del hilo
}
/*
Descripci□n: Funci□n encargada de inicializar la estructura del ABB
Recibe: arbol_bin *BinaryTree (apuntador al ABB declarado por el
usuario)
*/
void initialize(arbol_bin *BinaryTree){
      *BinaryTree = NULL;                     // El apuntador enviado
por el usuario se coloca en un valor NULL
      return;
}

/*
Descripci□n: Funci□n encargada de insertar un nuevo elemento en el ABB
Recibe: arbol_bin * BinaryTree (apuntador al ABB utilizado por el
usuario),
      int newNumber (nuevo elemento que se va a incluir en el ABB)
*/
void insert(arbol_bin * BinaryTree, int newNumber){
      arbol_bin * aux = BinaryTree;                  // Declaramos un
apuntador para recorrer el □rbol
      while(*aux != NULL){                          // Recorremos el
□rbol hasta encontrar el espacio libre donde ir□ el nuevo elemento
            if (newNumber > ((*aux)->number))   // En caso de que el
valor sea mayor, iremos a la parte derecha del □rbol
            {
                  aux = &((*aux)->right);
            }
            else{                                       // Caso contrario,
viajaremos a la parte izquierda del □rbol
                  aux = &((*aux)->left);
            }
      }
      *aux = (node *)malloc(sizeof(node));  // Una vez ubicados en su
lugar, le haremos espacio en memoria al nuevo nodo
      (*aux)->number = newNumber;                // En el nodo
colocaremos el n□mero que desea introducir el usuario al □rbol
      (*aux)->left = NULL;                      // Nos aseguramos de que
ambos hijos est□n apuntando a un valor NULL para evitar errores
      (*aux)->right = NULL;
      return;
}

/*
Descripci□n: Funci□n encargada de imprimir los tiempos de ejecuci□n
de los programas
Recibe:
      bool found (indica si se encontr□ el n□mero en la b□squeda)
      int keyNumber (n□mero que se busc□)
      int nSize (n□mero de datos sobre los que se realiz□ la b□squeda)
      double SysTime (tiempo del sistema)
```

```
	double UserTime (tiempo del usuario)
	double RealTime (tiempo real)
*/
/*void imprimeTiempos(bool found, int keyNumber, int nSize, double
RealTime, double UserTime,double SysTime){
	if (found)
	{
		printf("\nKey Number = %i Encontrado\n", keyNumber);
	}else{
		printf("\nKey Number = %i No Encontrado\n", keyNumber);
	}
	printf(" n = %i\n", nSize);
	printf("Real Time: \t%.10f\n", RealTime);
	printf("User Time: \t%.10f \n", UserTime);
	printf("System Time: \t%.10f\n", SysTime);
}*/
/*
Descripci□n: Funci□n que devolver□ los sub□rboles necesarios para
iniciar los hilos
Recibe: position aux[] (arreglo de posiciones donde a□adiremos los
apuntadores a los sub□rboles),
	arbol_bin *BinaryTree (apuntador al ABB utilizado por el
usuario),
	int n (n□mero de hilos inicial que se ir□ reduciendo)
*/
void subTrees(position aux[], arbol_bin * BinaryTree, int n){

	while(Vacio(BinaryTree) != 1)
	{
		// Si el nodo en el que estamos contiene el valor
		// significa que fue encontrado
		if((*BinaryTree)->number == keyNumber){
		found = true;
		break;
		}
		else if((*BinaryTree)->number < keyNumber)
		{
			// Si el valor que buscamos es mayor
			// al del nodo en el que nos encontramos
			// nos movemos a la derecha
			BinaryTree = &((*BinaryTree)->right);
		}
		else if((*BinaryTree)->number > keyNumber)
		{
			// Si el valor que buscamos es menor
			// al del nodo en el que nos encontramos
			// nos movemos a la izquierda
			BinaryTree = &((*BinaryTree)->left);
		}
	}
}

/*
  Descripcion: Funci□n que determina si un arbol est□ vac□o
  Regresa 1 en caso de que est□ vac□o y 0 en otro caso
 */
int Vacio(arbol_bin *a){
	if(*a == NULL)
```

```
	{
		return 1;
	}
	else
	{
		return 0;
	}
}
```

Búsqueda binaria o dicotómica

```
/*
    ************************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Búsqueda Lineal o Secuencial
    Compilación: "gcc binariahilo.c tiempo.x  -o binariahilo (tiempo.c
si se tiene la implementación de la libreria o tiempo.o si solo se
tiene el codigo objeto)"
    Ejecución: "./binariahilo n" (Linux y MAC OS)
    ************************************************************************
*/


//************************************************************************
//LIBRERIAS INCLUIDAS
//************************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tiempo.h"
#include <pthread.h>
#include <math.h>

//************************************************************************
//Declaracion de funciones
//************************************************************************
int binaria(void *arguments);
//************************************************************************
//Declaracion de estructuras
struct bin_args
{
    int *arr;
    int l;
    int r;
    int x;
};

//************************************************************************
//PROGRAMA PRINCIPAL
//************************************************************************
int main (int argc, char* argv[])
{
	//************************************************************************
***
	//Variables del main
	//************************************************************************
***
```

```
	//Variables para medición de tiempos
	double utime0, stime0, wtime0,utime1, stime1, wtime1;

	//Variables del algoritmo
	int n,x,i;

	//Apuntador para el arreglo
	int *arr;

	//*********************************************************************
***
	//Recepción y decodificación de argumentos
	//*********************************************************************
***

	//Si no se introducen exactamente 2 argumentos (Cadena de
ejecución y cadena=n)
	if (argc!=3)
	{
		printf("\nIndique el tamanio del algoritmo - Ejemplo:
[user@equipo]$ %s 100\n",argv[0]);
		exit(1);
	}
	//Tomar el segundo argumento como tamaño del algoritmo
	else
	{
		n=atoi(argv[1]);
		x=atoi(argv[2]);
	}

	//Creacion del arreglo
	arr=malloc(n*sizeof(int));

	printf("\n El valor a buscar es %d en arreglo tamaño %d\n",x,n);

	//Guardado de numeros
	for(i=0;i<n;i++){
		scanf("%i",&arr[i]);
	}

	//*********************************************************************
***
	//Iniciar el conteo del tiempo para las evaluaciones de
rendimiento
	//*********************************************************************
***
	uswtime(&utime0, &stime0, &wtime0);
	//*********************************************************************
***

	//*********************************************************************
***
	//Algoritmo
	//*********************************************************************
***
	//Se llama a la funcion binaria
	pthread_t threads[4];
    int res[4], mod = n % 4, part = n / 4;
```

```
    for (i = 0; i < 4; i++)
    {
        struct bin_args arg;
        pthread_t thread;
        threads[i] = thread;
        arg.arr = arr;
        arg.x = x;
        arg.l = part * i;
        if (mod != 0 && i == 3)
            arg.r = part * (i + 1) - 1 + mod;
        else
            arg.r = part * (i + 1) - 1;


        res[i] = pthread_create(&threads[i], NULL, &binaria, (void
*)&arg);
        pthread_join(threads[i], NULL);
    }
      //*****************************************************************
***
      //*****************************************************************
***
      //Evaluar los tiempos de ejecución
      //*****************************************************************
***
      uswtime(&utime1, &stime1, &wtime1);

      //Cálculo del tiempo de ejecución del programa
      printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");
      //*****************************************************************
***
      //Terminar programa normalmente
      return 0;
}

int binaria(void *arguments){
      pthread_detach(pthread_self());
    struct bin_args *args = arguments;
    int *arr = args->arr;
    int l = args->l;
```

```
    int r = args->r;
    int x = args->x;
    int times = log(r + 1) / log(2);
    int mid, valor= -1;
      for (int i = 0; i <= times; i++)
    {
        if (r >= l)
        {
            mid = l + (r - l) / 2;
            if (arr[mid] == x)
            {
                valor = mid;
                break;
            }
            else if (arr[mid] > x)
                r = mid - 1;
            else
                l = mid + 1;
        }
    }
      (valor == -1) ? printf("El valor %d no esta en esta parte del
arreglo\n",x)
                    : printf("El valor esta en la posicion %d\n", valor);
}
```

Búsqueda exponencial

```
/*
    ************************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Búsqueda Lineal o Secuencial
    Compilación: "gcc exponencialhilos.c tiempo.x  -o exponencialhilos
(tiempo.c si se tiene la implementación de la libreria o tiempo.o si
solo se tiene el codigo objeto)"
    Ejecución: "./exponencialhilos n" (Linux y MAC OS)
    ************************************************************************
*/

//**********************************************************************
//LIBRERIAS INCLUIDAS
//**********************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <pthread.h>
#include "tiempo.h"
//**********************************************************************
//DEFINICION DE CONSTANTES DEL PROGRAMA
//**********************************************************************
#define min //Definimos la constante para encontrar el minimo de dos
numeros
#define MAX_THREAD 4 //Numero máximo de hilos que se utilizarán
//**********************************************************************
***********
//DECLARACION DE ESTRUCTURAS
```

```
//*************************************************************************
***********
/*
La estructura siguiente contiene 4 elementos, el arreglo de números
ordenados, el numero
a buscar, el numero donde inicia el arreglo y donde termina
*/
struct bin_args{
    int *arr;
    int x;
    int inicio;
    int fin;
};

//*********************************************************************
//DECLARACIÓN DE FUNCIONES
//*********************************************************************
void *exponentialSearch(int *arr, int n, int x); //Se declara como un
apuntador debido a que se utilizará para los hilos
void *binarySearch(void *arguments); //Se declara como apuntador y
recibirá una estructura de argumentos, la estructura
                                                    //declarada
anteriormente
//*********************************************************************
//VARIABLES GLOBALES
//*********************************************************************
bool found = false;
//*********************************************************************
//PROGRAMA PRINCIPAL
//*********************************************************************
int main (int argc, char* argv[]){
      //*****************************************************************
***
      //Variables del main
      //*****************************************************************
***
      double utime0, stime0, wtime0,utime1, stime1, wtime1; //Variables
para medición de tiempos
      int n;      //n determina el tamaño del algorito dado por
argumento al ejecutar
      int i; //Variables para loops
      int *a,x;
      //*****************************************************************
***
      //Recepción y decodificación de argumentos
      //*****************************************************************
***

      //Si no se introducen exactamente 2 argumentos (Cadena de
ejecución y cadena=n)
      if (argc!=3){
            exit(1);
      }
      //Tomar el segundo argumento como tamaño del algoritmo
      else{
            n=atoi(argv[1]);
      }
```

```
	//*********************************************************************
***
	//Iniciar el conteo del tiempo para las evaluaciones de
rendimiento
	//*********************************************************************
***
	uswtime(&utime0, &stime0, &wtime0);
	//*********************************************************************
***

	//*********************************************************************
***
	//Algoritmo
	//*********************************************************************
***
	printf("\n\nBusqueda exponencial para buscar x=%d con %d
numeros",x,n);
	i=0;
	a=malloc(n*sizeof(int));

	do{
		scanf("%d",&a[i++]);
	}while(i<n);

	x=atoi(argv[2]);

	exponentialSearch(a, n, x);

	//*********************************************************************
***

	//*********************************************************************
***
	//Evaluar los tiempos de ejecución
	//*********************************************************************
***
	uswtime(&utime1, &stime1, &wtime1);

	//Cálculo del tiempo de ejecución del programa
	printf("\n");
	printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
	printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1
- utime0);
	printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
	printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
	printf("\n");

	//Mostrar los tiempos en formato exponecial
	printf("\n");
	printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
	printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1
- utime0);
	printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
	printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
	printf("\n");
```

```
      //*********************************************************************
***

      //Terminar programa normalmente
      exit (0);
}

//*********************************************************************
***
//DEFINICIÓN DE FUNCIONES
//*********************************************************************
***

void *binarySearch(void *arguments){
      /*struct bin_args *args=arguments;
      int *a=args->arr;
      int key=args->x;
      int n=args->size;
      int part=0;
      // Each thread checks 1/4 of the array for the key
    int thread_part=part++;
    int mid;

    int low=thread_part*(n/4);
    int high=(thread_part+1)*(n/4);

    while (low<high&&!found) {
        mid=(high - low)/2+low;

        if (a[mid]==key)  {
            found = true;
            break;
        }

        else if (a[mid] > key)
            high = mid - 1;
        else
            low = mid + 1;
    }*/
    pthread_detach(pthread_self());
    struct bin_args *args=arguments;
    int *arr=args->arr;
    int l=args->inicio;
    int r=args->fin;
    int x=args->x;
    int times=log(r+1)/log(2);
    int mid,value=-1;

    for (int i=0;i<=times;i++){
        if (r>=l){
            mid =l+(r-l)/2;
            if (arr[mid]==x){
                value=mid;
                break;
            }
            else if(arr[mid]>x)
                r=mid-1;
            else
                l=mid+1;
```

```
        }
    }
    if(value==-1)
        printf("El numero a buscar no se encuentra en el arreglo\n");
    else
        printf("El numero a buscar esta en la posicion %d del
arreglo\n", value);
}

void *exponentialSearch(int *arr, int n, int x){
      /*pthread_t threads[MAX_THREAD];
      struct bin_args arg;

    if (arr[0]==x){
        printf("El elemento a buscar esta en la posicion 0 del
arreglo\n");
        return 0;
    }else{
      arg.arr=arr;
      arg.x=x;
      arg.size=n;

      for (int i = 0; i < MAX_THREAD; i++)
        pthread_create(&threads[i], NULL, &binarySearch, (void*)&arg);

          for (int i = 0; i < MAX_THREAD; i++)
              pthread_join(threads[i], NULL);

          if(found)
            printf("El numero %d se encuentra en el arreglo",x);
          else
            printf("El numero %d no se encuentra en el arreglo",x);

          // key found in array
          /*if (found)
              cout << key << " found in array" << endl;

          // key not found in array
          else
              cout << key << " not found in array" << endl;

          return 0;
    }*/

    pthread_t threads[4];
    int i = 1, res[4];

    if (arr[0]==x){
        printf("El elemento a buscar esta en la posicion 0 del
arreglo\n");
        return 0;
    }

    while (i<n&&arr[i]<=x)
        i=i*2;

    int mod=n%4;

    for (int j=0,part=min(i,n-1)/4;j<4;j++){
```

```
        struct bin_args arg;
        pthread_t thread;
        threads[j]=thread;
        arg.arr=arr;
        arg.x=x;
        arg.inicio=part*j;
        if (mod!=0&&j==3)
            arg.fin=part*(j+1)-1+mod;
        else
            arg.fin=part*(j+1)-1;

        res[j]=pthread_create(&threads[j], NULL, &binarySearch, (void
*)&arg);
        pthread_join(threads[j],NULL);
    }
}
```

Búsqueda de Fibonnacci

```
/*
    ************************************************************************
    Curso: Análisis de algoritmos
    ESCOM-IPN
    Algoritmo de búsqueda Búsqueda Lineal o Secuencial
    Compilación: "gcc fibohilos.c tiempo.x  -o fibohilos (tiempo.c si
se tiene la implementación de la libreria o tiempo.o si solo se tiene
el codigo objeto)"
    Ejecución: "./fibohilos n" (Linux y MAC OS)
    ************************************************************************
*/

//*************************************************************************
//LIBRERIAS INCLUIDAS
//*************************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "tiempo.h"
#include <pthread.h>

//*************************************************************************
//Declaracion de funciones
//*************************************************************************
void sub(int A[], int i, int n, int B[]);
void* procesar(void* id);
int min(int x, int y){
    return (x <= y) ? x : y;
}
int fibo(int arr[], int x, int n);

//*************************************************************************
//PROGRAMA PRINCIPAL
//*************************************************************************
int main (int argc, char* argv[])
{
      //*************************************************************************
***
      //Variables del main
```

```
	//*********************************************************************
***
	//Variables para medición de tiempos
	double utime0, stime0, wtime0,utime1, stime1, wtime1;

	//Variables del algoritmo
	int n,x,i,nt;
	pthread_t *thread;
	//Apuntador para el arreglo
	int *arr;

	//*********************************************************************
***
	//Recepción y decodificación de argumentos
	//*********************************************************************
***

	//Si no se introducen exactamente 2 argumentos (Cadena de
ejecución y cadena=n)
	if (argc!=3)
	{
		printf("\nIndique el tamanio del algoritmo - Ejemplo:
[user@equipo]$ %s 100\n",argv[0]);
		exit(1);
	}
	//Tomar el segundo argumento como tamaño del algoritmo
	else
	{
		n=atoi(argv[1]);
		x=atoi(argv[2]);
	}

	//Creacion del arreglo
	arr=malloc(n*sizeof(int));

	printf("\n El valor a buscar es %d en arreglo tamaño %d\n",x,n);

	//Guardado de numeros
	for(i=0;i<n;i++){
		scanf("%i",&arr[i]);
	}

	//*********************************************************************
***
	//Iniciar el conteo del tiempo para las evaluaciones de
rendimiento
	//*********************************************************************
***
	uswtime(&utime0, &stime0, &wtime0);
	//*********************************************************************
***

	//*********************************************************************
***
	//Algoritmo
	//*********************************************************************
***
	//Se llama a la funcion binaria
	i=0;
```

```
    for(int j=0; j<20; j++){

        for (i=1; i<nt; i++) {
            if (pthread_create (&thread[i], NULL, procesar,(void*)i) !=
0 ) {
                perror("El thread no pudo crearse");
                exit(-1);
            }
        }
        procesar(0);

        for (i=1; i<nt; i++){
                pthread_join(thread[i], NULL);
            }
        }

      //*****************************************************************
***
      //*****************************************************************
***
      //Evaluar los tiempos de ejecución
      //*****************************************************************
***
      uswtime(&utime1, &stime1, &wtime1);

      //Cálculo del tiempo de ejecución del programa
      printf("\n");
      printf("real (Tiempo total)  %.10f s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10f s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10f s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");

      //Mostrar los tiempos en formato exponecial
      printf("\n");
      printf("real (Tiempo total)  %.10e s\n",  wtime1 - wtime0);
      printf("user (Tiempo de procesamiento en CPU) %.10e s\n",  utime1
- utime0);
      printf("sys (Tiempo en acciónes de E/S)  %.10e s\n",  stime1 -
stime0);
      printf("CPU/Wall   %.10f %% \n",100.0 * (utime1 - utime0 + stime1
- stime0) / (wtime1 - wtime0));
      printf("\n");
      //*****************************************************************
***
      //Terminar programa normalmente
      return 0;
}

void sub(int A[], int i, int n, int B[]){
      int aux, cont=0;
      for(aux = i; aux<i+n; aux++){
            B[cont] = A[aux];
            cont ++;
      }
```

```
}

void* procesar(void* id){
      int n_thread=(int)id;
      int inicio,fin,ne,n,nt, *datos,x;

      inicio=(n_thread*n)/nt;
      if(n_thread==nt-1){
            fin=n;
            ne=fin-inicio;
      }else{
            fin=((n_thread+1)*n)/nt-1;
            ne = (fin-inicio)+1;
      }
      printf("\nThread %d\tInicio %d\tTermino %d\n",n_thread,inicio,
fin);
      int *B = (int*)malloc(ne*sizeof(int));
      sub(datos,inicio,ne,B);
      int result = fibo(B,x,ne);
      if(result!=-1){
            printf("\nNUmero %d encontrado %d\n",x,result);
      }
      if(n_thread!=0){
        pthread_exit(0);
    }
}

int fibo(int arr[], int x, int n){
      //Se inicializan los numeros de fibonacci
    int fibMMm2 = 0; //el numero en la posicion (m-2)
    int fibMMm1 = 1; //el numero en la posicion (m-1)
    int fibM = fibMMm2 + fibMMm1; //m'th

    //Guardara el numero que sea mayor a n
    while (fibM < n){
        fibMMm2 = fibMMm1;
        fibMMm1 = fibM;
        fibM = fibMMm2 + fibMMm1;
    }

    int offset = -1; //marca el rango eliminando desde el frente

    //Se compara arr[fibMMm2] con x y se inspeccionan los elementos
    while(fibM > 1){
        //comprueba si MMm2 es valido
        int i = min(offset + fibMMm2, n-1);
        // Si x es mas grande que el valor dentro de fibMm2, corta el
arreglo de desplazamiento i
        if(arr[i]<x){
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }else if(arr[i]>x){ // Si x es mas grande que el valor dentro
de fibMm2, corta el arreglo de desplazamiento i+1
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }else //el elemento se encontro
```

```
            return i;
    }
    //compara el ultimo elemento con x
    if (fibMMm1 && arr[offset + 1]==x)
        return offset +1;

    //si el elemento no se encuentra, retorna -1
    return -1;
}
```