INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

ANÁLISIS DE ALGORITMOS

PROFESORA: LUZ MARÍA SÁNCHEZ GARCÍA

INTEGRANTES:

VÁZQUEZ MORENO MARCOS OSWALDO 2016601777

DE LOS SANTOS DÍAZ LUIS ALEJANDRO 2017630451

PRÁCTICA 5 ANÁLISIS DE ALGORITMOS DIVIDE Y VENCERÁS

3CM2

1 DE ABRIL DE 2019

Introducción

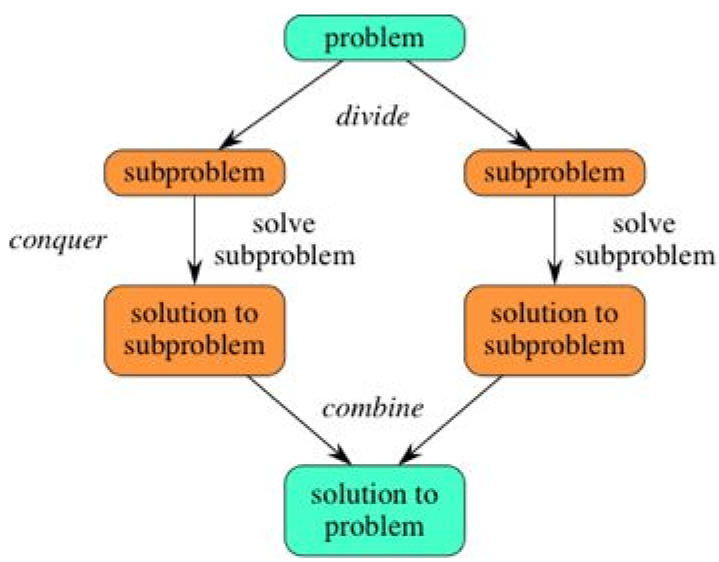
Tanto el ordenamiento por mezcla como el ordenamiento rápido emplean un paradigma algorítmico común que se basa en la recursividad. Este paradigma, divide y vencerás, separa un problema en subproblemas que se parecen al problema original, de manera recursiva resuelve los subproblemas y, por último, combina las soluciones de los subproblemas para resolver el problema original. Como divide y vencerás resuelve subproblemas de manera recursiva, cada subproblema debe ser más pequeño que el problema original, y debe haber un caso base para los subproblemas. Debes pensar que los algoritmos de divide y vencerás tienen tres partes:

Divide el problema en un número de subproblemas que son instancias más pequeñas del mismo problema.

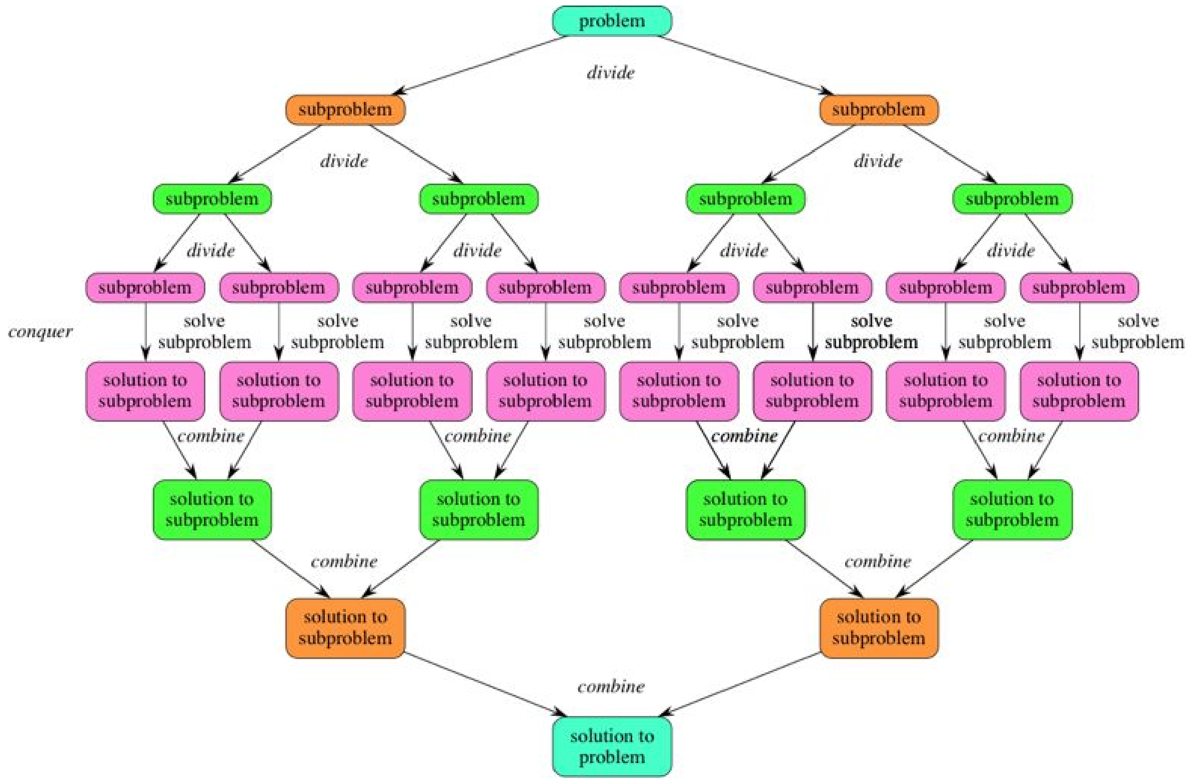
Vence los subproblemas al resolverlos de manera recursiva. Si son los suficientemente pequeños, resuelve los subproblemas como casos base.

Combina las soluciones de los subproblemas en la solución para el problema original.

Puedes recordar fácilmente los pasos para un algoritmo de divide y vencerás como divide, conquista, combina. Aquí está cómo ver un paso, al suponer que cada paso de dividir crea dos subproblemas (aunque algunos algoritmos de divide y vencerás crean más de dos):



Si expandimos a dos pasos recursivos más, se ve así:



Como divide y vencerás crea por lo menos dos subproblemas, un algoritmo de divide y vencerás hace muchas llamadas recursivas. (Academy, 2006)

Planteamiento del problema

Coin base en el ordenamiento obtenido a partir del archivo de entrada que tiene 10,000,000 de números diferentes, Realizar la búsqueda de elementos bajo 2 métodos de búsqueda, realizar el análisis teórico y experimental de las complejidades; así como encontrar las cotas de los algoritmos por el método Divide y Vencerás.

Diseño de la solución

A continuación, se muestran los diagramas de flujo de nuestra propuesta de solución para los algoritmos.

Primeramente, se muestra en el diagrama 1.1 el BBR.c de nuestra propuesta de solución.

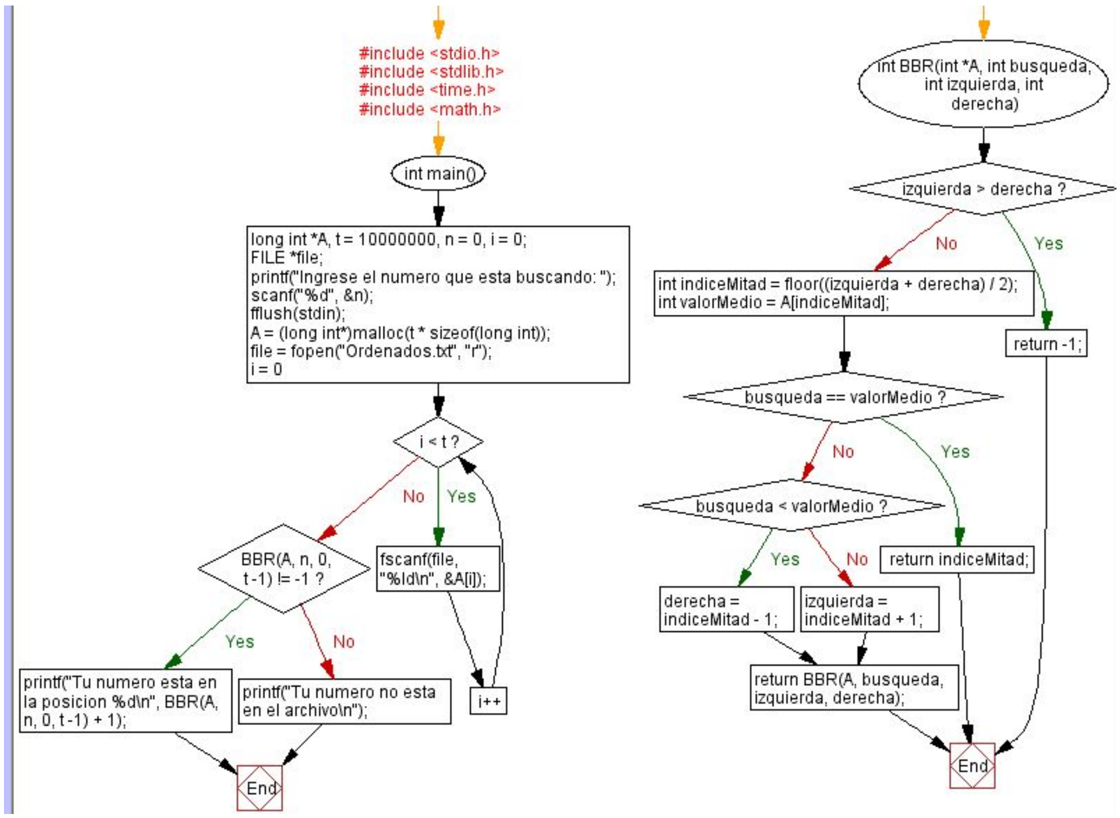


Diagrama 1.1 BBR.c

A continuación, se muestra en el diagrama 1.2 la implementación de nuestro archivo MMR.c

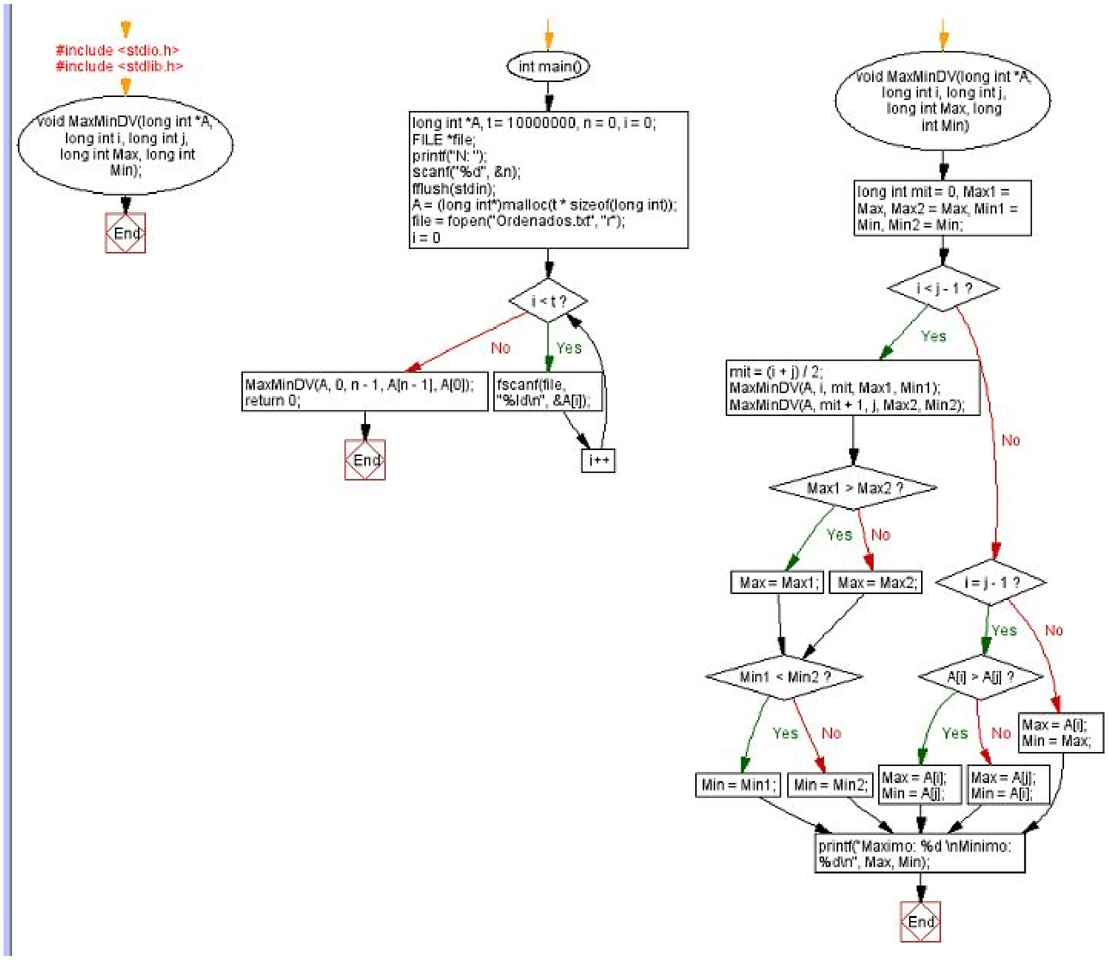


Diagrama 1.2 MMR.c

Implementación de la solución

BBR.c

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

int main()
{
    long int *A, t = 10000000, n = 0, i = 0;
    FILE *file;
    printf("Ingrese el numero que esta buscando: ");
    scanf("%d", &n);
    fflush(stdin);
    A = (long int*)malloc(t * sizeof(long int));
    file = fopen("Ordenados.txt", "r");
    for(i = 0; i < t; i++) fscanf(file, "%ld\n", &A[i]);
```

```
    if(BBR(A, n, 0, t -1) != -1) printf("Tu numero esta en la posicion
%d\n", BBR(A, n, 0, t -1) + 1);
    else printf("Tu numero no esta en el archivo\n");
}

int BBR(int *A, int busqueda, int izquierda, int derecha)
{
    if(izquierda > derecha) return -1;

    int indiceMitad = floor((izquierda + derecha) / 2);
    int valorMedio = A[indiceMitad];

    if(busqueda == valorMedio) return indiceMitad;

    if(busqueda < valorMedio) derecha = indiceMitad - 1;
    else izquierda = indiceMitad + 1;

    return BBR(A, busqueda, izquierda, derecha);
}
```

MMR.c

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

void MaxMinDV(long int *A, long int i, long int j, long int Max, long int
Min);

int main()
{
    long int *A, t = 10000000, n = 0, i = 0;
    FILE *file;
    printf("N: ");
    scanf("%d", &n);
    fflush(stdin);
    A = (long int*)malloc(t * sizeof(long int));
    file = fopen("Ordenados.txt", "r");
    for(i = 0; i < t; i++) fscanf(file, "%ld\n", &A[i]);
    MaxMinDV(A, 0, n - 1, A[n - 1], A[0]);
    return 0;
}

void MaxMinDV(long int *A, long int i, long int j, long int Max, long int
Min)
{
    long int mit = 0, Max1 = Max, Max2 = Max, Min1 = Min, Min2 = Min;
    if (i < j - 1)
    {
        mit = (i + j) / 2;
        MaxMinDV(A, i, mit, Max1, Min1);
```

```c
            MaxMinDV(A, mit + 1, j, Max2, Min2);
            if(Max1 > Max2) Max = Max1;
            else Max = Max2;
            if(Min1 < Min2) Min = Min1;
            else Min = Min2;
        }
        else if(i = j - 1)
        {
            if(A[i] > A[j]){ Max = A[i]; Min = A[j];}
            else { Max = A[j]; Min = A[i];}
        }
        else {Max = A[i]; Min = Max;}
        printf("Maximo: %d \nMinimo: %d\n", Max, Min);
    }
```

Funcionamiento

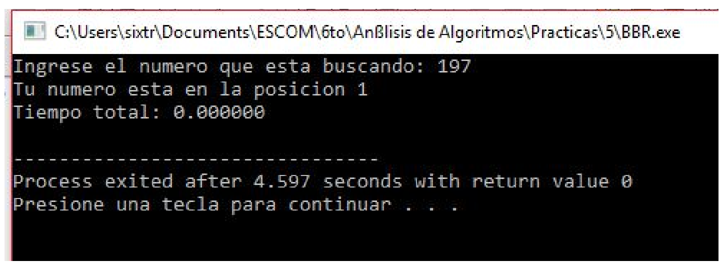


Imagen 1.1. Primera posición

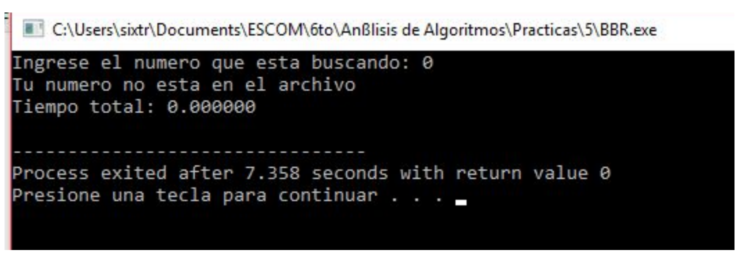


Imagen 1.2. Número que no está en el archivo

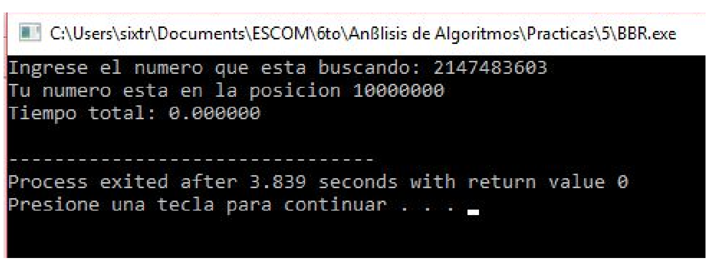
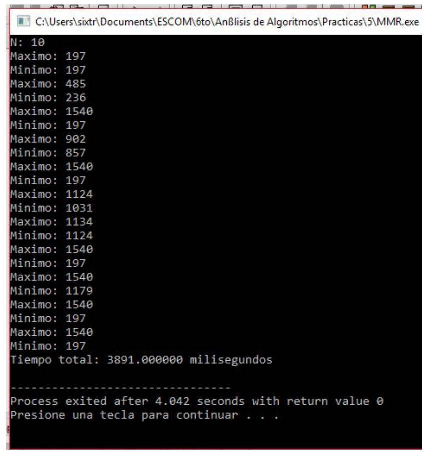


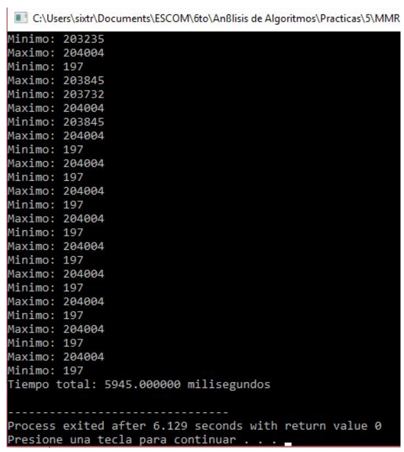
Imagen 1.3. Última posición



```
C:\Users\sixtr\Documents\ESCOM\6to\Anßlisis de Algoritmos\Practicas\5\MMR.exe
N: 10
Maximo: 197
Minimo: 197
Maximo: 485
Minimo: 236
Maximo: 1540
Minimo: 197
Maximo: 902
Minimo: 857
Maximo: 1540
Minimo: 197
Maximo: 1124
Minimo: 1031
Maximo: 1134
Minimo: 1124
Maximo: 1540
Minimo: 197
Maximo: 1540
Minimo: 1179
Maximo: 1540
Minimo: 197
Maximo: 1540
Minimo: 197
Tiempo total: 3891.000000 milisegundos

--------------------------------
Process exited after 4.042 seconds with return value 0
Presione una tecla para continuar . . .
```

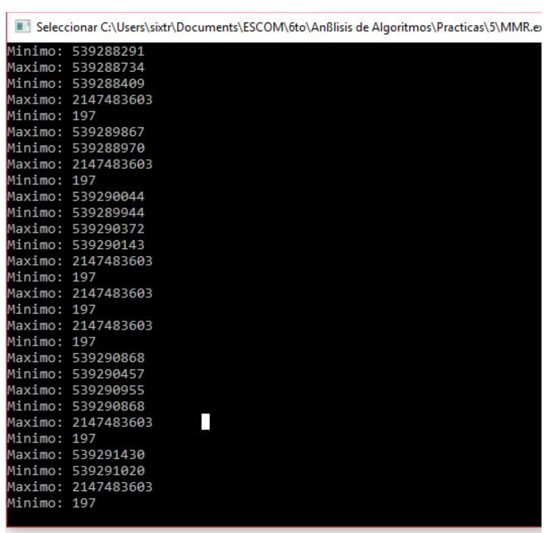
Imagen 1.4. 10 números



```
C:\Users\sixtr\Documents\ESCOM\6to\Anßlisis de Algoritmos\Practicas\5\MMR
Minimo: 203235
Maximo: 204004
Minimo: 197
Maximo: 203845
Minimo: 203732
Maximo: 204004
Minimo: 203845
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Maximo: 204004
Minimo: 197
Tiempo total: 5945.000000 milisegundos

--------------------------------
Process exited after 6.129 seconds with return value 0
Presione una tecla para continuar . . .
```

Imagen 1.5. 1000 números



```
Seleccionar C:\Users\sixtr\Documents\ESCOM\6to\Anßlisis de Algoritmos\Practicas\5\MMR.e
Minimo: 539288291
Maximo: 539288734
Minimo: 539288409
Maximo: 2147483603
Minimo: 197
Maximo: 539289867
Minimo: 539288970
Maximo: 2147483603
Minimo: 197
Maximo: 539290044
Minimo: 539289944
Maximo: 539290372
Minimo: 539290143
Maximo: 2147483603
Minimo: 197
Maximo: 2147483603
Minimo: 197
Maximo: 2147483603
Minimo: 197
Maximo: 539290868
Minimo: 539290457
Maximo: 539290955
Minimo: 539290868
Maximo: 2147483603
Minimo: 197
Maximo: 539291430
Minimo: 539291020
Maximo: 2147483603
Minimo: 197
```

Imagen 1.6. 10000000 números

Se toman en cuenta los dos últimos valores

A continuación, se muestra el caso general y el caso particular de la complejidad temporal:

Caso general:

$$T_{dyv}(n) = \begin{cases} T_{trivial}(n) & n \leq n_0 \\ T_{dividir}(n, L) + \sum_{i=1}^{L} T_{dyv}(m_i) + T_{combinar}(n, L) & n > n_0 \end{cases}$$

Caso particular (muy frecuente):

- El problema original (de tamaño n) se divide en L subproblemas de tamaño n/b, con $L \geq 1$ y $b \geq 2$.
- $T_{trivial}(n) \in \Theta(1)$ $(n_0 \geq 1)$
- $T_{dividir}(n, L) + T_{combinar}(n, L) \in \Theta(n^k)$, con $k \geq 0$

$$T_{dyv}(n) = \begin{cases} 1 & n \leq n_0 \\ LT_{dyv}\left(\frac{n}{b}\right) + n^k & n > n_0 \end{cases}$$

Vamos a resolver la recurrencia para $n \in \{bn_0, b^2 n_0, b^3 n_0, \ldots\}$

Hipótesis: $T(n)$ continua y monótona no decreciente

Cambio de variable:

$$n \equiv b^i n_0 \Leftrightarrow i \equiv \log_b(n/n_0)$$

Nos queda la siguiente expresión:

$$\begin{aligned} t(i) &\equiv T(b^i n_0) \\ &= LT(b^{i-1} n_0) + \left(b^i n_0\right)^k \\ &\equiv Lt(i-1) + n_0^k b^{ik} \end{aligned}$$

Y la recurrencia queda:

$$t(i) - Lt(i-1) = n_0^k \left(b^k\right)^i$$

La solución general es:

$$t(i) = \begin{cases} c_1 L^i + c_2 \left(b^k\right)^i & L \neq b^k \\ c_1 \left(b^k\right)^i + c_2 i \left(b^k\right)^i & L = b^k \end{cases}$$

$$\boxed{L \neq b^k}$$

Invirtiendo el cambio nos queda:

$$T(n) = c_1 \left(\frac{n}{n_0}\right)^{\log_b L} + c_2 \left(\frac{n}{n_0}\right)^k = C_1 n^{\log_b L} + C_2 n^k$$

donde:

$$\begin{aligned} C_1 &= c_1 / n_0^{\log_b L} \\ C_2 &= c_2 / n_0^k \end{aligned}$$

Sustituyendo en la recurrencia original se obtiene:

$$\begin{aligned} n^k &= T(n) - LT(n/b) \\ &= C_1 n^{\log_b L} + C_2 n^k - L\left(C_1 \left(\frac{n}{b}\right)^{\log_b L} + C_2 \left(\frac{n}{b}\right)^k\right) \\ &= \left(1 - \frac{L}{b^k}\right) C_2 n^k \end{aligned}$$

Y de ahí: $C_2 = \left(1 - \frac{L}{b^k}\right)^{-1}$

$$\boxed{L \neq b^k}$$

Se pueden sacar conclusiones sobre el orden de magnitud de $T(n)$ independientemente del valor de C_1

La solución general es:

$$T(n) = C_1 n^{\log_b L} + \left(1 - \frac{L}{b^k}\right)^{-1} n^k$$

$L < b^k$

- $\left(1 - \frac{L}{b^k}\right)^{-1} > 0$ y $k > \log_b L$
- Por tanto: $T(n) \in \Theta(n^k) \quad \forall n \in \{bn_0, b^2 n_0, b^3 n_0, \ldots\}$
- $T(n)$ continua y monótona no decreciente $\Rightarrow \quad \forall n \; \boxed{T(n) \in \Theta(n^k)}$

$L > b^k$

- $\left(1 - \frac{L}{b^k}\right)^{-1} < 0$ y $k < \log_b L$
- $C_1 > 0$, ya que ha de ser $T(n) > 0 \quad \forall n$
- $\boxed{T(n) \in \Theta\left(n^{\log_b L}\right)}$

$$\boxed{L = b^k}$$

- De ahí obtenemos que $C_2 = 1$
- La solución nos queda:

$$T(n) = C_1 n^k + n^k \log_b \left(\frac{n}{n_0}\right) = (C_1 - \log_b n_0)\, n^k + n^k \log_b n$$

- Y por tanto, independientemente del valor de C_1, se tiene que:

$$\boxed{T(n) \in \Theta(n^k \log n)}$$

En resumen:

$$\boxed{T(n) \in \begin{cases} \Theta(n^k) & L < b^k \\ \Theta(n^k \log n) & L = b^k \\ \Theta(n^{\log_b L}) & L > b^k \end{cases}}$$

En tiempos de la búsqueda binaria tenemos:

El caso base T(0)= 1.

El caso base T(0)=n/2.

En tiempos de la búsqueda del máximo y mínimo

T(n)= T(n+1) + T(n) + 20.

Hablando acerca de la función espacial para la búsqueda binaria:

f(n) = 2n

Hablando acerca de la función espacial para la búsqueda del máximo y el mínimo:

f(n) = 5n

Medición de tiempos

Búsqueda Binaria

Número a buscar	Tamaño de n	Tiempo real	Encontrado
322486	100	0.026	No
14700764	1000	0.064	No
3124011	5000	0.029	Si
6337399	10000	0.031	Si
61394	20000	0.065	No
10393545	40000	0.033	Si
2147445644	60000	0.041	No
1295390003	80000	0.031	Si
450057883	100000	0.56	No
187645041	200000	0.042	No
1980098116	400000	0.029	No
152503	600000	0.041	No
5000	800000	0.042	No
14932823650	1000000	0.029	No
214826	2000000	0.03	No
1843349527	4000000	0.029	No
1360839354	6000000	0.03	No
2109248666	8000000	0.034	Si
2147470850	9000000	0.046	Si
0	10000000	0.3	No

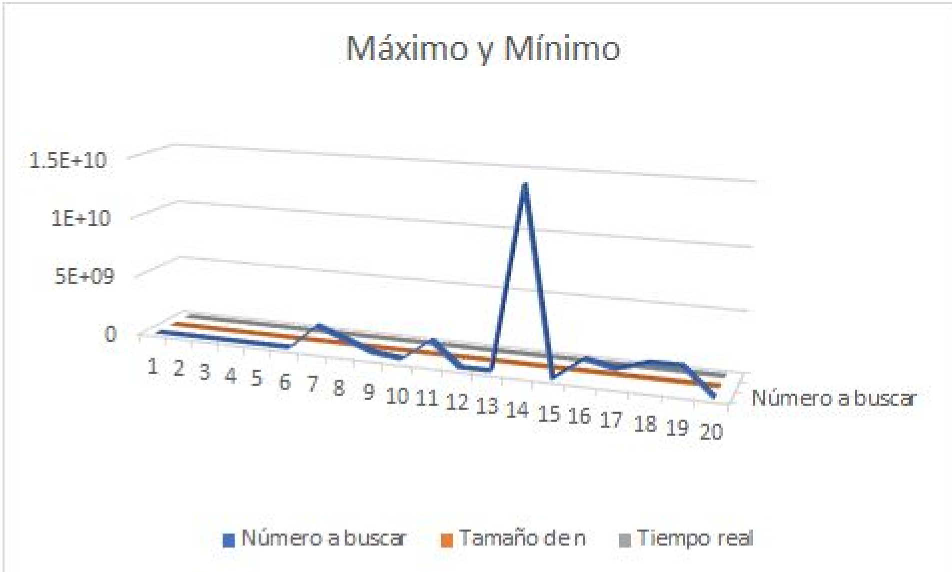
Tabla 1.1

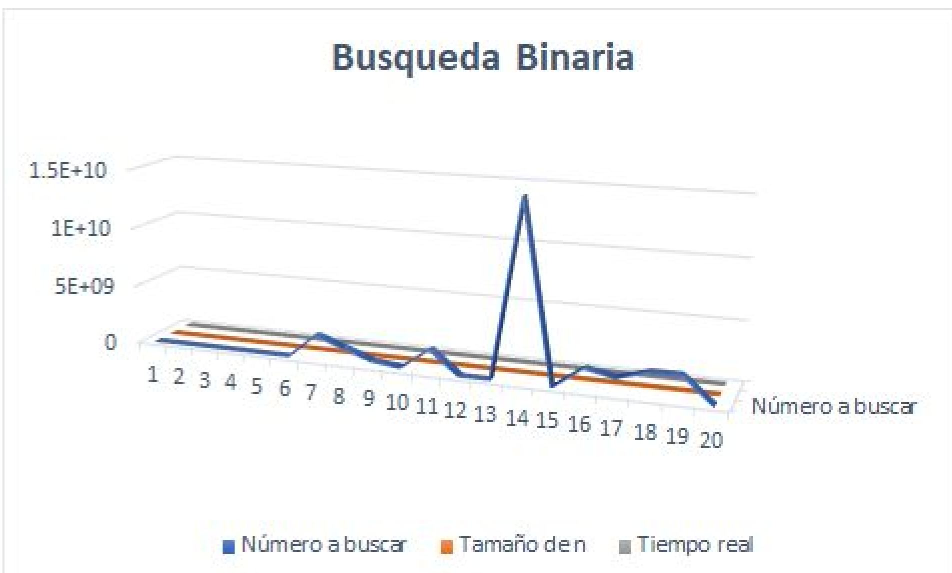
Máximo y Mínimo

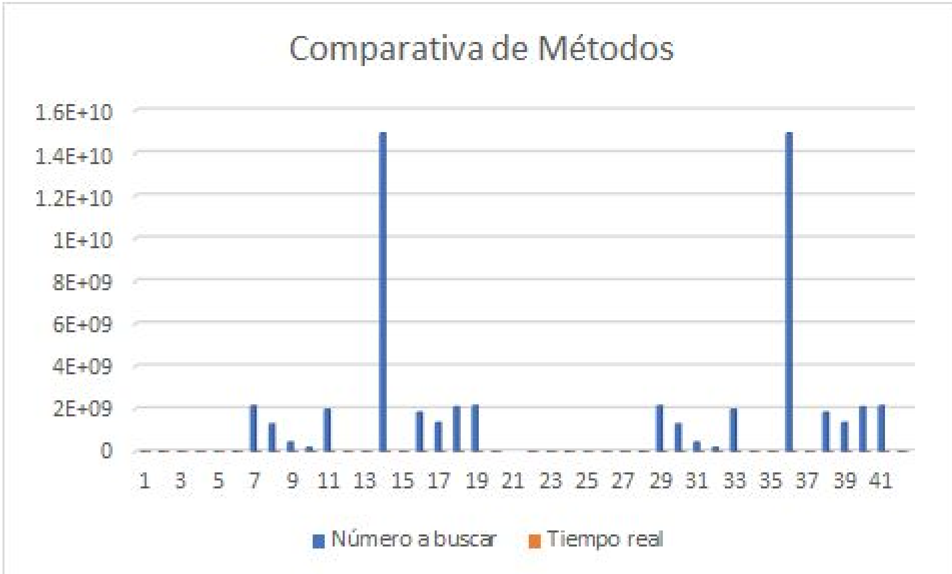
Número a buscar	Tamaño de n	Tiempo real	Encontrado
322486	100	0	No
14700764	1000	0	No
3124011	5000	0.029	Si
6337399	10000	0.001	Si
61394	20000	0.002	No
10393545	40000	0.0028	Si
2147445644	60000	0.003	No
1295390003	80000	0.0018	Si
450057883	100000	0.043	No
187645041	200000	0.0099	No
1980098116	400000	0.019	No
152503	600000	0.0099	No
5000	800000	0.029	No
14932823650	1000000	0.029	No
214826	2000000	0.034	No
1843349527	4000000	0.037	No
1360839354	6000000	0.033	No
2109248666	8000000	0.027	Si
2147470850	9000000	0.048	Si
0	10000000	0.34	No

Tabla 1.2

Gráficas de funciones







Cuestionario

1. ¿Cuál de los 2 algoritmos es más fácil de implementar?

R: Búsqueda binaria

2. ¿Cuál de los 2 algoritmos es el más difícil de implementar? R: Búsqueda máximo y mínimo

3. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por qué? R: No, consideramos que el tiempo de entrega de resultados sería mucho más prolongado.

5. ¿Si solo se realizará el análisis teórico de un algoritmo antes de implementarlo, podrías asegurar cual es el mejor? R: Talvez, depende del algoritmo. Pero generalmente no.

6. ¿Qué recomendaciones darían a nuevos equipos para realizar esta práctica? R: Apoyarse de la práctica anterior para obtener el documento con los números previamente ordenados.

Plataforma experimental

La ejecución de los algoritmos anteriores se llevó a cabo en una computadora personal que se describe en la siguiente imagen.

Imagen 3.1 Plataforma Experimental

El compilador utilizado fue gcc integrado dentro del IDE DevC en un sistema operativo de 64 bits Windows 10.

Conclusiones

En conclusión, podemos decir que el método de Divide y Vencerás como nuevo método para nosotros se convierte en uno de los métodos más rápidos y prácticos para nuestro uso, ya que divide el problema como en el antiguo Grecia, lugar en donde se decidió dividir a la gente para así poder conquistar, aquí dividimos el problema en partes para buscar un número de una manera más fácil a lo que podría ser buscarlo desde el inicio hasta el final, en una búsqueda lineal.

Por lo anterior, se dice que la técnica de diseño de algoritmos Divide y Vencerás trata de resolver un problema de forma recursiva, a partir de la solución de subproblemas del mismo tipo pero de menor tamaño.

El término Divide y Vencerás, en su acepción más amplia, es algo más que una técnica de diseño de algoritmos. Se suele considerar una filosofía general para resolver problemas y se utiliza en muchos ámbitos.

Bibliografía

Bibliografía

Academy, K. (15 de 6 de 2006). *Algoritmo Divide y Vencerás .* Obtenido de https://es.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms

Díaz, I. N. (2006). *FIET UNICAUCA*. Obtenido de http://artemisa.unicauca.edu.co/~nediaz/EDDI/cap02.htm

g, S. (s.f.).

García, L. M. (15 de 03 de 2019). *Classrom.* Obtenido de Práctica 5 Ánalisis de Algoritmos: https://classroom.google.com/c/NzEzNTUxMDU1NVpa/a/NzgxNTE2NTA3MVpa/details

https://classroom.google.com/u/0/c/NzEzNTUxMDU1NVpa/a/NzYwMTgwNzg2NFpa/details