

TÜRKİYE CUMHURİYETİ
YILDIZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



Fibonacci search algoritması

Öğrenci no:19011095

Öğrenci adı ve soyadı:Efe Girgin

Ders hocası:Fatih Amasyalı

Ders:BLM1012-1 YAPISAL PROGRAMLAMAYA GİRİŞ

YOUTUBE ADRESİ: <https://www.youtube.com/watch?v=FkJZrbHBPjc>

FİBONACCI SEARCH ALGORİTMASI

TANIM:

- 1.Fibonacci Araması, sıralanmış bir dizideki bir öğeyi aramak için Fibonacci sayılarını kullanan karşılaştırma tabanlı bir tekniktir.
- 2.Fibonacci arama;sıralı bir dizide,fibonacci sayılarını kullanarak olası yerleri daraltan(arama bölgesini küçülten) ve 'böl ve fethet'stratejisiyle sonuca ulaşan bir algoritmadır.
- 3.Bu arama algoritması, özyineli(recursive) bir seri olan fibonacci sayılarını kullanarak sıralı bir dizi üzerinde arama yapmaktadır.
- 4.Diğer eleme yöntemleri arasında Fibonacci arama yöntemi, tek değerli fonksiyonlar için en uygun noktayı bulmak için en iyi yöntem olarak kabul edilmektedir.
- 5.fibonacci search optimal noktayı içeren aralıklarda çalışır ve optimal noktaya diğer arama algoritmalarından daha hızlı yakınsar

UYGULAMA ALANLARI-1:

- 1.Örneğin bir dosyada bir kelimenin aranması, bir ağaç yapısında(tree) bir düğümün (node) aranması veya bir dizi(array) üzerinde bir verinin aranması gibi durumlar bu algoritmaların çalışma alanlarına girer.
 - 2.Kombinatoryal optimizasyondaki sorunlar , örneğin: En kısa yol probleminin bir şekli olan araç rotalama problemi
 - 3.örneğin: Harita boyama problemi Sudoku veya bulmaca doldurma oyun teorisi ve özellikle kombinasyon oyun teorisi ,
 - 4.en iyi hamleyi seçerek sonraki (örneğin olduğu gibi yapmaya minmax algoritma) Tüm olasılıklardan bir kombinasyon veya şifre bulma
 - 5.Bir liste veya dizideki maksimum veya minimum değeri bulma Bir dizi değerinde belirli bir değerin olup olmadığını kontrol etme
- Karar değişkenlerinin optimal değerleri belirlendikten sonra amaç fonksiyonunun optimal değeri hesaplanır.
- Fibonacci arama yönteminde olduğu gibi sayısal optimizasyon yöntemlerinde, amaç fonksiyonunun değerlerinin önce karar değişkenlerinin çeşitli kombinasyonlarında bulunması şeklinde ters bir prosedür izlenir. ve daha sonra optimal çözümle ilgili sonuçlar çıkarılır

ÇALIŞMA MANTIĞI-1

Çalışma mantığı arama yapılacak olan sıralı diziye fibonacci sayılarını kullanarak parçalara bölmektir. Örneğin arama yapılacak olan alanın en fazla 2147483647 değerine sahip olabildiği integer alan olsun. Bu durumda bu sayıya ulaşana kadar olan fibonacci sayılarına ihtiyaç duyulacaktır:

{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169, 63245986, 102334155, 165580141, 267914296, 433494437, 701408733, 1134903170, 1836311903, INT_MAX};

Bu sayılar arama alanını parçalara bölmektedir. Öncelikle aranan sayının, en büyük fibonacci sayısından büyük mü küçük mü olduğu kontrol edilir. Buna göre ilk kontrolde 1836311903 sayısından küçük ise bu sefer 1134903170 sayısından küçük olup olmadığı kontrol edilir.

Şayet aranan sayı büyük ise bu sefer serinin 2 önceki sayısı mevcut sayıya eklenir. Örneğin sayımızı bulmak için 196418 sayısından büyük olup olmadığını kontrol ederken sayının büyük olduğu görüldü bu durumda aradığımız sayının 196418 indeksinden sağda olduğunu biliyoruz. Ancak bir önceki sayı olan 317811'den de küçük olduğu biliniyor bu durumda $196418 + 75025 = 271443$ sayısından büyük olup olmadığı kontrol edilir ve bu işlem böylece gider.

Yani özetle fibonacci sayılarının arasındaki aralık serideki her sayıda artmaktadır. Fibonacci araması ise bu işi terse çevirerek en büyük aralıktan en küçüğe doğru aralığı daraltarak arama işlemi yapar. En sonunda aralık 1'e inince sayı bulunur (ya da sayı seride yer almıyorsa bulunmadığı anlaşılır).

Bu arama algoritması bir [parçala fethet \(divide and conquer\)](#) yaklaşımıdır. Benzer bir arama yöntemi olan [ikili arama \(binary search\)](#) ile aynı algoritma karmaşıklığına sahiptir $O(\log n)$ ancak yapılan matematiksel çalışmalarda fibonacci aramasının daha hızlı olduğu gösterilmiştir.

İkili Arama ile benzerlikler:

Sıralanmış diziler için çalışır

Böl ve Yönet Algoritması.

Log n zaman karmaşıklığına sahiptir.

İkili Arama ile Farklar:

Fibonacci Araması, verilen diziyi eşit olmayan parçalara böler

İkili Arama, aralığı bölmek için bir bölme operatörü kullanır.

Fibonacci Search / kullanmaz, + ve - kullanır. Bölme operatörü bazı CPU'larda maliyetli olabilir.

Fibonacci Araması, sonraki adımlarda nispeten daha yakın öğeleri inceler. Bu nedenle, giriş dizisi CPU önbelleğine veya hatta RAM'e sığamayacak kadar büyük olduğunda, Fibonacci Araması yararlı olabilir.

Diziyi ikili arama(binary search)gibi eşit parçalara(1:1)bölmek yerine fibonacci sayılarını kullanarak 1:1,618'le orantılı parçalara bölerek karşılaştırmaları yapar.

İkili arama fibonacci aramasına göre ortalama %4 daha az karşılaştırma yapıldığı tespit edilmiştir

Binary search gibi fibonacci search de linear search algoritmasının bir rakibidir ve ondan çok daha hızlı çalışırlar

algoritma:

Aranan eleman x olsun.

Buradaki fikir, önce verilen dizinin uzunluğundan büyük veya ona eşit olan en küçük Fibonacci sayısını bulmaktır.

Bulunan Fibonacci sayısı fib (m 'inci Fibonacci sayısı) olsun. İndeks olarak $(m-2)$ 'inci Fibonacci sayısını kullanıyoruz (eğer geçerli bir indeks ise).

$(m-2)$ 'inci Fibonacci Sayısı i olsun, $arr[i]$ 'yi x ile karşılaştırırız, x aynı ise i döndürürüz. Aksi takdirde, eğer x daha büyükse, i 'den sonra alt dizi için yineleniriz, yoksa i 'den önce alt dizi için yineleniriz.

Algoritmamızın adımları:.

1.

n 'den büyük veya eşit en küçük Fibonacci Sayısını bulun. Bu sayı fib [m 'inci Fibonacci Sayısı] olsun.

Ondan önceki iki Fibonacci sayısı $fib1$ [$(m-1)$ 'th Fibonacci Number] ve $fib2$ [$(m-2)$ 'th Fibonacci Number] olsun.

2.

Burada ilk önce bulduğumuz $fib2$ değerini dizinin indisi olarak kullanıcaz yani $fib2$ dizinin kaçinci elemanını kullanacağımızı gösterecek

1. x 'i $fib2$ indisinin karşılık geldiği öğeyle karşılaştırm

2. burada indis x ile eşleşirse, zaten bulmuşuz demektir yani $fib2$ indisinin karşılık geldiği eleman x den küçük ya da büyük değilse eşittir ve sonucumuzu bulmuşuzdur

3. Aksi takdirde, x elemandan küçükse yani $arr[fib2] > x$, üç Fibonacci değişkenini iki Fibonacci aşağı hareket ettirin; bu, kalan dizinin yaklaşık üçte ikisinin ortadan kaldırıldığını gösterir.

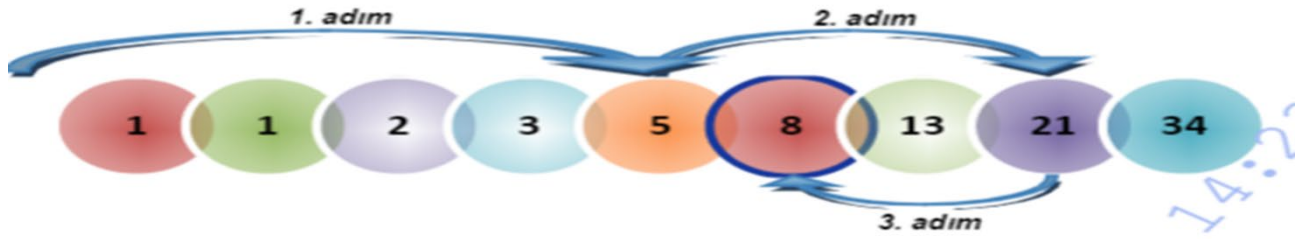
4. üstteki ifademiz de olmazsa, x elemandan büyükse yani $arr[fib2] < x$, üç Fibonacci değişkenini bir Fibonacci aşağı hareket ettirin. ve bulduğumuz yeni $fib2$ değerimizi eski $fib2$ değerimize ekleriz yani indisimizi artırırız dizimiz zaten küçükten büyüğe doğru sıralı olduğu için indisi optimum bir şekilde arttırıp indisi arttırız. Bunlar birlikte, kalan dizinin yaklaşık üçte birinin ortadan kaldırıldığını gösterir.

3.

Karşılaştırma için kalan tek bir öğe olabileceğinden, $fibMm1$ 'in 1 olup olmadığını kontrol edin. Evet ise, x 'i bu kalan öğeyle karşılaştırm. Eşleşirse, dizini döndürün.

ÇALIŞMA MANTIGI-2;

Örnek dizi $A=(1,1,2,3,5,8,13,21,34)$ ve aranan değer '8' olsun veri dizisi 9 elemanlı olduğu için eleman sayısına eşit veya büyük olann en küçük fibonacci sayısı 13'tür. dolayısıyla bir ve iki önceki fibonacci sayıları da 8 ve 5 olacaktır. iki önceki fibonacci sayısı olan 5 geçerli bir indis olduğu için bu indisteki eleman ile aranan değer karşılaştırılır. $A(5)<8$ (eger aradığımız elemandan küçükse 5'i biz tutuyoruz yani diyoruz ki ilk 5 elemanı taradık 5 eleman cepte) koşul sağlandığından dolayı (fibonacci serisini bir aşağı kaydırırız) yeni aranan alanı 3-5-8 fibonacci sayılarının bir önceki aşamada 5 ötelenmiş aralığındadır. yani $5+3=8$ indisli elemandır. $A(8)>8$ (eger aradığımız elemandan büyük ise biz 8 i tutmuyoruz $5+3$ 'teki 5'imizi tutuyoruz) olduğundan yeni arama alanı 1-2-3 fibonacci sayılarının ötelenmiş aralığındaki $5+1=6$ indisli elemandır.



Şekil 7.17 "Fibonacci arama" ile örnek işlem adımları

FİBONACCİ SAYILARIMIZ VE KODU BİLMEDEN ÖNCE BİLİNMESİ GEREKENLER

Fibonacci Sayıları özyinelemeli olarak $F(n) = F(n-1) + F(n-2)$, $F(0) = 0$, $F(1) = 1$ olarak tanımlanır.

İlk birkaç Fibinacci Sayısı 0, 1, 1'dir, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

0 . 1 . 1 . 2 . 3 . 5 . 8 . 13 . 21 . 34

↓

fib2	fib1	fib
3	5	8
5	8	13
8	13	21
13	21	34

2
(5-3)

Yeni dizi
2 3 5

$fib = fib1$
 $fib1 = fib2$
 $fib2 = fib - fib1$

0 1 1 2 3 5 8 13 21 34

↓

fib2	fib1	fib
3	5	8
5	8	13
8	13	21
13	21	34

2
(5-3)

Yeni dizi
1 2 3

$fib = fib2$
 $fib1 = fib1 - fib2$
 $fib2 = fib - fib1$

N elemanlı A dizisi içinde(veri yığını kayıt dosyasında) klavyeden girilen Degeri'Fibonacci arama (fibonacci search)algoritmasıyla arayan program

```
// fibonnaci search
```

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

```
// burda bir fonksiyon tanımlıyoruz bu fonksiyon girilen iki degerden kucugunu bulmamızı saglıyor
```

```
//asagıda bu fonksiyonu kullanıcaz
```

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

```
{
```

```
    return (a < b) ? a : b; //a eger b den kucukse a,eger kucuk degilse b
```

```
}
```

```
/* Returns index of x if present, else returns -1 */
```

```
int fibMonaccianSearch(int arr[], int x, int n)
```

```
{
```

```
    /* Fibonacci sayılarını başlat */
```

```
    int fib2 = 0; // fibonacci serisinin ilkini kendimiz verdik
```

```
    int fib1 = 1; // fibonacci serisinin ikincisini de kendimiz verdik
```

```
    int fib = fib2 + fib1; // burda fibonacci serilerinin mantığı olan ardışık 2 terimin toplamının 3.sunu verdigini  
    islemsel olarak yaptık
```

```
    /* eger fib degiskenimiz sayının boyutundan kucukse while ı döndürdük */
```

```
    while (fib < n) {
```

```
        fib2 = fib1; //burada fibonacci serisini 1 adım ileri götürdük
```

```
        fib1 = fib;
```

```
        fib = fib2 + fib1;
```

```
    }
```

```
    //burda offset diye bir degisken tutuyoruz bunun -1 den başlamasının sebebi kodlamada diziler 1 den degil 0  
    dan baslar yani mesela 4.dedigimiz eleman dizide ar[3] olarak tutulur bunun nedeni dedigimiz gibi dizilerin 0  
    dan başlamasıdır
```

```
    int offset = -1;
```

```
//fib degerimiz eger 1 den buyukse while ı döndürmeye devam et
while (fib > 1) {
    //fib2 ile dizinin boyutunu karşılaştır küçük olani i degiskenine ata(diziler kodda 0'dan basladığı için dizinin
boyutunu yazarken de 1 çıkardık
    int i = min(offset + fib2, n - 1);

    /* arr[i] degerimiz aradığımız degerden(x) kucukse bu if in icine gir */
    if (arr[i] < x) {
        fib = fib1;
        fib1 = fib2;
        fib2 = fib - fib1;
        offset = i;
    }

    //eger arr[i] degerimiz aradigimiz degerden buyukse buraya girer
    else if (arr[i] > x) {
        fib = fib2;
        fib1 = fib1 - fib2;
        fib2 = fib - fib1;
    }

    //eger arr[i] degerimiz aradığımız degerden buyuk ya da kucuk degilse otomatikman eşittir o zaman else e
girecektir
    else
        return i;
}

/* son elemanı x ile karşılaştırmak */
printf("deneme %d",offset);
if (fib1 && arr[offset + 1] == x)
    return offset + 1;
```

```

/*eger elemanı bulamazsak -1 döndür */
return -1;
}

/* driver function */
int main(void)
{
    int n,x,k;//degiskenleri tanımladık

    printf("lutfen dizinin boyutunu aliniz\n");//kullanıcıdan dizinin boyutunu aldık

    scanf("%d",&n);

    int arr[n];//diziyi tanımladık

    //kullanıcıdan diziyi aldık
    for(k=0;k<n;k++){
        printf("%d.elemanı giriniz\n",k+1);
        scanf("%d",&arr[k]);
    }

    //kullanıcının aradığı değeri kullanıcıdan alınız

    printf("aradiginiz değeri girin\n");
    scanf("%d",&x);

    for(k=0;k<n;k++){
        printf("%d\n",arr[k]);
    }

    int ind = fibMonaccianSearch(arr, x, n);

    //indisimiz 0 dan büyükse indisi yazdır ama yukarıdaki fonksiyonda gördüğümüz gibi sayıyı bulamazsak -1
    döndür demiştik yani eger -1 döndürülürse if e değil else e girecek burda da sayıyı bulamadığımızı ekrana
    yazdıracak

    if(ind>=0)

        printf("dizimide sayımız bulundu: dizinin %d. elemanıdır",ind+1);

    else

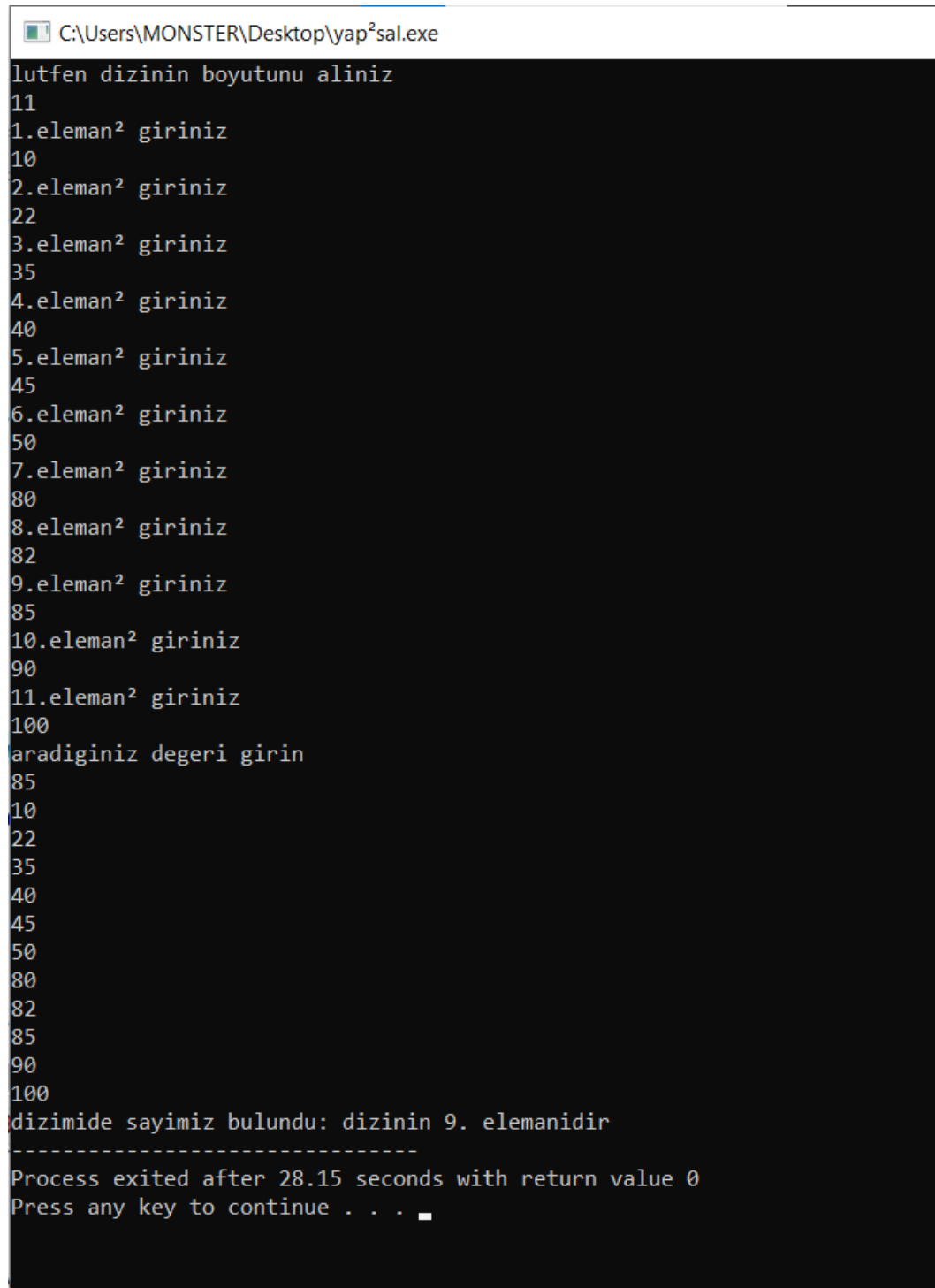
```

```
printf("%d dizide yoktur",x);

return 0;

}
```

EKRAN ÇIKTIMIZ:



```
C:\Users\MONSTER\Desktop\yap2sal.exe
lutfen dizinin boyutunu aliniz
11
1.eleman2 giriniz
10
2.eleman2 giriniz
22
3.eleman2 giriniz
35
4.eleman2 giriniz
40
5.eleman2 giriniz
45
6.eleman2 giriniz
50
7.eleman2 giriniz
80
8.eleman2 giriniz
82
9.eleman2 giriniz
85
10.eleman2 giriniz
90
11.eleman2 giriniz
100
aradiginiz degeri girin
85
10
22
35
40
45
50
80
82
85
90
100
dizimide sayimiz bulundu: dizinin 9. elemanidir
-----
Process exited after 28.15 seconds with return value 0
Press any key to continue . . . █
```

```
lutfen dizinin boyutunu aliniz
11
1.elemanı giriniz
10
2.elemanı giriniz
22
3.elemanı giriniz
35
4.elemanı giriniz
40
5.elemanı giriniz
45
6.elemanı giriniz
50
7.elemanı giriniz
80
8.elemanı giriniz
82
9.elemanı giriniz
85
10.elemanı giriniz
90
11.elemanı giriniz
100
aradiginiz degeri girin
85
10
22
35
40
45
50
80
82
85
90
100
dizimide sayımız bulundu: dizinin 9. elemanıdırfun() took 0.000056 seconds to execute
```

İllüstrasyon: Algoritmayı aşağıdaki örnekle anlayalım:

i	1	2	3	4	5	6	7	8	9	10	11	12	13
ar[i]	10	22	35	40	45	50	80	82	85	90	100	-	-

YAPTIĞIMIZ ANALİZLER

→ 10 22 35 40 45 50 80 82 (85) 90 100

$x = 85$ (Aradığımız değer)
 $n = 11$ (Dizinin boyutu)

→ fib → 13 olarak çıktı

$5 + 8 \rightarrow 13$

fib → 13
fib1 → 8
fib2 → 5

fib2	fib1	fib	offset	$i = \min(\text{offset}, \text{fib2})$	$\text{arr}[i]$	Notlar
5	8	13	-1	(4)	$\text{arr}[4] = 45$	i'yi offsete ata ve fibonacci yi 1 küçült
(3)	5	8	(4)	(7)	$\text{arr}[7] = 82$	i'yi offsete ata ve fibonacci yi 1 küçült
(2)	3	5	(7)	8	$\text{arr}[8] = 90$	i ile bir işlem yapma fibonacci serisini 2 küçült
(1)	1	2	(7)	8	$\text{arr}[8] = 85$	i'yi offsete almadığımızdan önceki değer kaldı return i

Zaman Karmaşıklık analizi:

En kötü durum, biz onu bulmaya devam ederken hedefimiz dizinin daha büyük (2/3) kısmında olduğunda ortaya çıkar.

Başka bir deyişle, her seferinde dizinin daha küçük (1/3) kısmını ortadan kaldırıyoruz.

Bir kez n için, sonra (2/3) n için, sonra (4/9) n için ve bundan sonra için çağırırız.

Fibonacci search rakibi olan binary search gibi karmaşıklığı $O(\log n)$ cinsindendir

$$fib(n) = \left\lceil \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n \right\rceil \sim c * 1.62^n$$

*for $n \sim c * 1.62^n$ we make $O(n')$ comparisons. We, thus, need $O(\log(n))$ comparisons.*

Logarithmic Zamanlı : $O(\log N)$

Bir loopun time complexity'si yahut big o notasyonu , loopun değerleri bir sabit tarafından bölünüp çarpıldığı zaman $O(\log N)$ olur.

$O(\log N)$ $O(N)$ 'den daha hızlıdır.

Örneğin : Binary Search

```
while ( low <= high ) {  
    mid = ( low + high ) / 2;  
    if ( target < list[mid] )  
        high = mid - 1;  
    else if ( target > list[mid] )  
        low = mid + 1;  
    else break; }
```


Fibonacci araması, $O(\log n)$ ortalama ve en kötü durum karmaşıklığına sahiptir

Fibonacci dizisi, bir sayının kendisinden önceki iki sayının toplamı olma özelliğine sahiptir.

Bu nedenle dizi, tekrarlanan toplama ile hesaplanabilir.

kaynakça

<http://bilgisayarkavramlari.com/2008/08/05/fibonacci-arama-algoritmasi-fibonacci-search-algorithm/>

<https://www.geeksforgeeks.org/fibonacci-search/>

Algoritma geliştirme ve programlama kitabı/ fahri vatansever

https://www.researchgate.net/publication/242791340_An_improvement_on_Fibonacci_search_method_in_optimization_theory

https://en.wikipedia.org/wiki/Fibonacci_search_technique