Son Teslim tarihi: 03.06.2022 23:59:00

Comb Sort

OĞUZHAN TOPALOĞLU
Ç19052025 – Grup 1
oguzhan.topaloglu@std.yildiz.edu.tr

Bilgisayar Mühendisliği Bölümü,
Elektrik-Elektronik Fakültesi,
Yıldız Teknik Üniversitesi

Istanbul, 2022

İÇERİK:

ANLATIM VİDEOSU:

https://www.youtube.com/watch?v=XyXEylyXGn0

NOT: 7 dk altında olması için konuşurken takıldığım yerleri kestim ve x1.02 hızında render ettim o yüzden 4-5 yerde kamerada ışınlanıyor gibi görülebilirim

Comb Sort algoritması nedir?

Comb Sort algoritması, Włodzimierz Dobosiewicz ve Artur Borowy tarafından 1980'de tasarlanmış ve 1991'de Byte isimli bir dergide Stephen Lacey ve Richard Box tarafından "Comb Sort" ismi ile duyurulmuş bir dizi sıralama (sorting) algoritmasıdır.

Bubble Sort'un geliştirilmiş (iyileştirilmiş) bir versiyonudur.

Comb Sort algoritması nasıl çalışır?

Bubble Sort'ta "turtle" adını verdiğimiz terimler vardır, bu terimler dizinin sonlarında bulunan küçük değerli elemanlardır.

Bubble sort her seferinde (turda) birbirine komşu terimlere bakarak swapping yaptığından dolayı en sonlarda bulunan küçük değerli elemanların dizinin başına gelmesi epey bir tur alacaktır.

Bu yüzden bu terimlere turtle denir çünkü Bubble Sort algoritmasının yavaş çalışmasına neden olurlar.

Comb Sort, bu turtle sorununu ortadan kaldırmak için "gap" diye bir değişken tanımlar.

Bu değişken, karşılaştırma yapılacak terimlerin arasındaki boşluğu ifade eder.

Aslında bu değişken Bubble Sort'ta da vardır ancak değeri hiç değişmez ve hep 1 olarak kalır yani birbirine bitişik terimlerin karşılaştırılması yapılır.

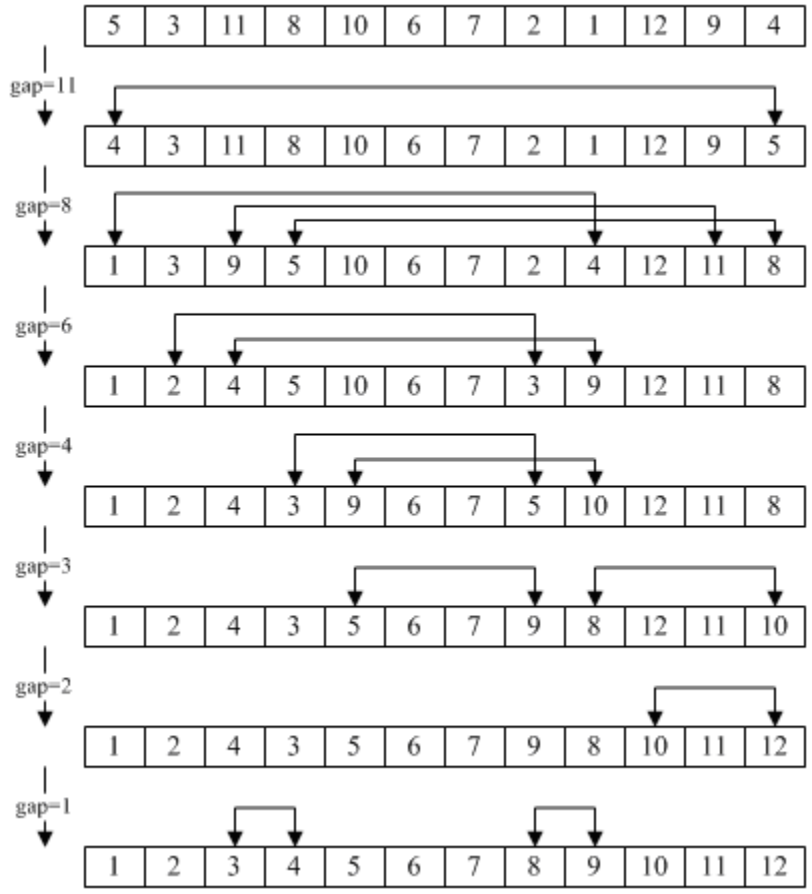
Comb Sort, turtle'lardan kurtulabilmek için hep birbirine komşu terimleri karşılaştırmak yerine aralarında "gap" kadar boşluk olan terimleri karşılaştırır ve "shrink" dediğimiz bir faktör ile gap değişkenini günceller (küçültür).

Shrink değişkeni için farklı farklı değerler alınabilir, varsayılan (klasik) Comb Sort implementasyonunda 1.3 olarak alınmıştır. Yani her adımda gap = gap / 1.3 yapmalıyızdır.

Gap değişkeninin her turda güncellenmesinin nedeni de şudur: Turtle'lardan olabildiğince hızlı (algoritma çalışmaya başladıktan hemen sonra) kurtulmak istediğimizden gap'in olabildiğince büyük olmasını isteriz ancak turtle'lardan kurtulduktan sonra da gap'in küçülmesini ve Comb Sort'un bir bakıma Bubble Sort gibi davranarak turtle'ların bulunmadığı diziyi birkaç turda sıralamasını isteriz. Bu yüzden gap'in algoritmanın başında dizinin uzunluğuna eşit olmasını isterken her tur sonrasında shrink değişkeni ile güncellenerek eninde sonunda 1'e eşit olmasını isteriz.

Özetle Comb Sort, Bubble Sort'un turtle sorununu ortadan kaldırmak için gap kavramını implemente etmiş ve çalıştıkça gap'i küçülterek sonlara doğru Bubble Sort'a benzeyen bir dizi sıralama algoritmasıdır.

Görsel bir örnek:



Comb Sort algoritması nerelerde kullanılır?

Sayısal değerler içeren bir dizinin sıralanması gereken her durumda kullanılabilir.

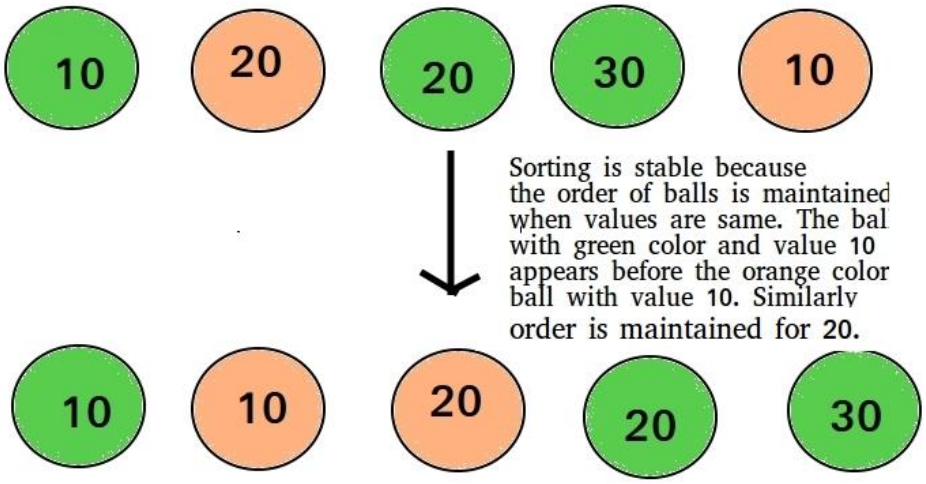
İyileştirilmiş bir Bubble Sort algoritması olduğundan dolayı Bubble Sort'un kullanıldığı her yerde kullanılarak daha iyi average case zaman karmaşıklığı elde edilebilir.

Comb Sort'un avantajları nelerdir?

1- Bellek karmaşıklığı karşılaştırma tabanlı bir sıralama algoritması (comparison based sorting algorithm) olduğundan dolayı O(1)'dir yani hiç bellek kullanmaz ve diziyi yalnızca karşılaştırmalar ve swapping'ler yaparak sıralar.
2- Çalışma şekli kolayca anlaşılabilirdir.
3- Bubble Sort'a aşırı benzemesine karşın average case'de Bubble Sort'tan daha hızlı çalışır.

Comb Sort'un dezavantajları nelerdir?

1- Birbirine eşit terimler olduğunda sıralanmadan önceki eşit terim sırası sıralanma sonrasında da korunuyorsa o sıralama algoritmasına kararlı (stable) sıralama algoritması denir.



Comb Sort stable bir sorting algoritması değildir.

2- Worst case zaman karmaşıklığı Bubble Sort ile aynıdır ve O(n^2)'dir. Yani çok fazla terimi olan ve sıralanması çok zaman alacak (sıralı biçimden çok uzak) dizilerde çok yavaş çalışır.

Comb Sort için zaman karmaşıklığı incelemesi

- Best case karmaşıklığı: O(n)
- Average case karmaşıklığı: O(n^2 / 2^p) ve burada p increment sayısıdır (kimi kaynaklar O(nlogn), O(n^2) de diyor)
- Worst case karmaşıklığı: O(n^2)

Best case'in O(n) olmasının nedeni eğer array zaten sorted (sıralı) ise algoritmamız n terime teker teker bakarak dizinin sıralı olduğunu görür ve durur.

Worst case de benzer şekilde eğer dizi küçükten büyüğe sıralanacakken eğer tam tersi şekilde duruyorsa algoritmamız n * n kez işlem yapacaktır.

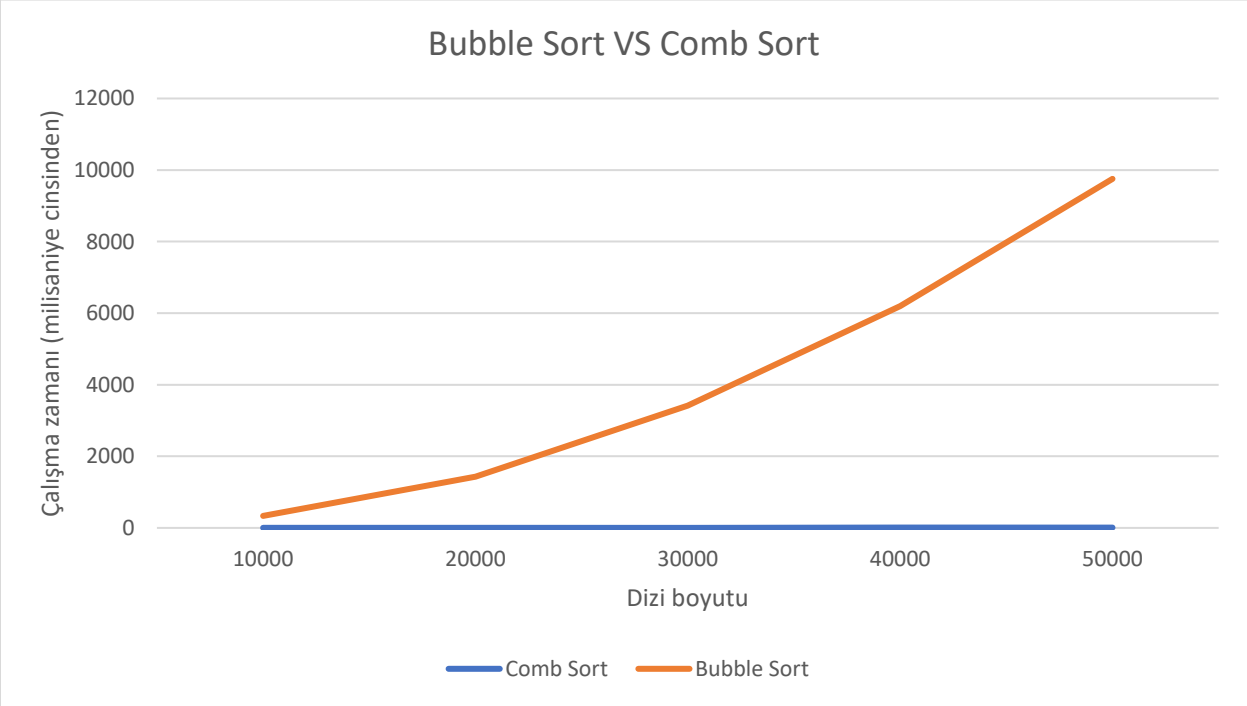
Average case'in kanıtı karmaşık olduğundan hiçbir yerde açıklamasını bulamadım, çoğu site aşağıdaki pdf'e yönlendiriyor. Bir kanıt için aşağıdaki pdf'in 16. sayfasına göz atılabilir:

https://hurna.io/assets/files/algorithms/sort/sorting.pdf

Aşağıda kod ile yapılan analiz gösterilmiştir:

```
N degeri: 10000
combSortTime: 2 milisaniye
bubbleSortTime: 333 milisaniye

N degeri: 20000
combSortTime: 2 milisaniye
bubbleSortTime: 1432 milisaniye

N degeri: 30000
combSortTime: 4 milisaniye
bubbleSortTime: 3412 milisaniye

N degeri: 40000
combSortTime: 6 milisaniye
bubbleSortTime: 6196 milisaniye

N degeri: 50000
combSortTime: 8 milisaniye
bubbleSortTime: 9746 milisaniye
```

```
Comb sort times:
2
2
4
6
8

Bubble sort times:
333
1432
3412
6196
9746

Comb sort:
*
*
**
***
****

Bubble sort:
*
****
**********
******************
**************************
```



Comb Sort'un sınırları

Worst case O(n^2) olduğundan dolayı çok büyük ve sıralı olmaktan çok uzak dizilerde çalışması çok uzun zamanlar alabilir…

Dizinin boyutu n = 10^6 ise ve dizi tam tersine dizilmiş bir şekilde ise O(10^12) olur ki bu çok uzun bir zaman demektir.

Comb Sort'un rakipleri

Worst case'i O(n^2)'den daha iyi olan bir sürü sorting algoritması vardır. Örneğin Radix Sort O(n), Heap Sort O(nlogn) karmaşıklığına sahiptir.

Tabii ki de worst case'inin iyi olması her zaman comb sort'tan daha iyi olduğu anlamına gelmez. Comb sort'un O(1) bellek karmaşıklığına (space complexity / auxiliary space) sahip olması da comb sort'un iyi bir yanıdır. (tradeoff kavramı)

Comb Sort'un C dilinde kodu

```c
/*
Oguzhan Topaloglu, C19052025 (CAP ogrencisiyim)
Yapisal Prog. Giris Projesi - Comb Sort
*/

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

// kod kac farkli boyut icin calisacak onu tanimlar
// ilk boyut 10000 olarak ayarlanmistir, her iterasyonda +10000 artacaktir
#define ITER_COUNT 3
```

```c
// iki degiskenin degerlerini degistirir
void swap(int* a, int* b){
        int temp = *a;
        *a = *b;
        *b = temp;
}


// comp sort algoritmasi
void combSort(int N, int arr[N]){
    int i, gap = N, swapped = 1;

    while (gap != 1 || swapped == 1) {
        gap /= 1.3;

        if(gap < 1)
            gap = 1;

        swapped = 0;

        for (i = 0; i < N - gap; i++) {
            if (arr[i] > arr[i + gap]) {
                swap(&arr[i], &arr[i + gap]);
                swapped = 1;
            }
        }
    }
}



// bubble sort algoritmasi, comb sort'ta gap = 1 yapilmasi ve gap'in
degistirilmemesi ile ayni sey
void bubbleSort(int N, int arr[N]){
    int i, swapped = 1;

    while (swapped == 1) {
        swapped = 0;

        for (i = 0; i < N - 1; i++) {
            if (arr[i] > arr[i + 1]) {
                swap(&arr[i], &arr[i + 1]);
                swapped = 1;
            }
        }
    }
}
```

```c
// iki tane diziyi rastgele degerlerle doldurur, iki dizi birbirinin kopyasi
olacaktir
void fillArrays(int N, int arr1[N], int arr2[N]){
srand(time(NULL));
    int i;
    int randomValue;
    for(i = 0; i < N; i++){
        randomValue = rand();
        arr1[i] = randomValue;
        arr2[i] = randomValue;
    }
}




void compareCombAndBubbleSortForN(int N, int index, int
combSortTimes[ITER_COUNT], int bubbleSortTimes[ITER_COUNT]){
    int combSortTime, bubbleSortTime;

    // hem bubble sort hem de comb sort icin birbirinin kopyasi iki dizi
    int combSortArray[N], bubbleSortArray[N];

    // dizilerin rastgele terimlerle doldurulmasi
    fillArrays(N, combSortArray, bubbleSortArray);

    // comb sort
    clock_t t1;
    t1 = clock();
    combSort(N, combSortArray);
    t1 = clock() - t1;
    combSortTime = (int) t1;

    // bubble sort
    clock_t t2;
    t2 = clock();
    bubbleSort(N, bubbleSortArray);
    t2 = clock() - t2;
    bubbleSortTime = (int) t2;

    // zamanlarin yazdirilmasi
    printf("\nN degeri: %d", N);
    printf("\ncombSortTime: %d milisaniye", combSortTime);
    printf("\nbubbleSortTime: %d milisaniye\n", bubbleSortTime);

    combSortTimes[index] = combSortTime;
    bubbleSortTimes[index] = bubbleSortTime;
}
```

```c
// utility fonksiyonu, konsola * yazdirmak icin
void printStars(int starCount){
    int j;
    for(j = 0; j < starCount; j++)
        printf("*");
}



// konsola yazisal ve simgesel olarak calisma surelerini yazdirir
void visualizeTimes(int combSortTimes[ITER_COUNT], int
bubbleSortTimes[ITER_COUNT]){
    int i;

    printf("\n\n");

    // yazisal
    printf("\nComb sort times: \n");
    for(i = 0; i < ITER_COUNT; i++) {
        printf("%d\n", combSortTimes[i]);
    }
    printf("\nBubble sort times: \n");
    for(i = 0; i < ITER_COUNT; i++) {
        printf("%d\n", bubbleSortTimes[i]);
    }

    // simgesel
    printf("\nComb sort: \n");
    for(i = 0; i < ITER_COUNT; i++) {
        // ilk calisma suresi 1 yildiz olsun
        if(i == 0)
            printStars(1);
        // obur calisma surelerinin yildizlerini ilkine gore oranlayarak koy
        else
            printStars(combSortTimes[i] / combSortTimes[0]);
        printf("\n");
    }
    printf("\nBubble sort: \n");
    for(i = 0; i < ITER_COUNT; i++) {
        // ilk calisma suresi 1 yildiz olsun
        if(i == 0)
            printStars(1);
        // obur calisma surelerinin yildizlerini ilkine gore oranlayarak koy
        else
            printStars(bubbleSortTimes[i] / bubbleSortTimes[0]);
        printf("\n");
    }
}
```

```c
int main(){
    // zamanlari saklamak icin
    int combTimes[ITER_COUNT];
    int bubTimes[ITER_COUNT];

    int i;
    for(i = 0; i < ITER_COUNT; i++){
        compareCombAndBubbleSortForN((i+1) * 10000, i, combTimes, bubTimes);
    }

    visualizeTimes(combTimes, bubTimes);
    return 0;
}
```

Comb Sort'un C kodunun açıklaması

Kodun en üstünde ITER_COUNT isimli değişken ile kodun kaç farklı boyut için çalışacağı #define preprocessor'ı ile belirtiliyor. Örneğin bu değer 5 ise 5 farklı boyutta dizi hem bubble sort hem de comb sort'tan geçirilecek.

Main fonksiyonunun içinde iki tane dizi tanımlıyorum, bu diziler her seferinde (her farklı dizi için) comb sort'un ve bubble sort'un çalışmasının ne kadar milisaniye sürdüğünü saklayacak. Daha sonra da ITER_COUNT kez compareCombAndBubbleSortForN() fonksiyonunu gerekli dizilerin referanslarını ve N boyutunu i değişkeni ile hesaplayarak geçirerek çalıştırıyorum ki sorting metotları çalışsın zamanlar dizilerimde depolansın.

compareCombAndBubbleSortForN fonksiyonu, öncelikle bu seferliğine hesaplayacağı süreleri combSortTime, bubbleSortTime değişkenleri olarak tanımlıyor. Sonra iki tane

dizi tanımlıyorum ve bu dizileri birbirlerinin kopyaları olacak şekilde fillArrays() isimli metodumla rastgele değerlerle dolduruyorum. Bu dizilerden birisini comb sort, birisini de bubble sort ile sıralayacağım. Daha sonrasında clock_t kullanımı ile (nasıl kullanacağımı internetten öğrendim kaynaklara bakabilirsiniz) öncelikle comb sort algoritması ile birinci dizimi daha sonrasında da bubble sort algoritması ile ikinci dizimi sort ediyorum. Daha sonrasında da bu seferki N değerini ve comb sort ile bubble sort'un ne kadar sürdüğünü konsola yazdırıp ayrıca referans olarak gönderdiğim main metodumdaki dizilere kaydediyorum ki ITER_COUNT kez kod çalıştıktan sonra main içindeki visualizeTimes() metodum bunları * simgesi ile bar chart şeklinde yazdırsın (bu metodu, basit işlemler ve printf metotlarından oluştuğundan dolayı açıklamayacağım, kodu hızlıca okunabilir şekilde bence)

combSort() metodunu açıklamak gerekirse, öncelikle başta gerekli indeks değişkenim olan i değişkenini ve gap ile swapped değişkenlerini tanımlıyorum. Burada gap ilk başta dizinin boyutuna eşit olacak ki turtle'lardan olabildiğinde hızlı kurtulalım. Ayrıca bu gap değişkeni her döngüde (turda) 1.3 shrink faktörüne bölünecek ki algoritma istediğimiz gibi çalışsın ve her turda Bubble Sort'a benzemeye başlasın. swapped değişkenini de "proven until guilty" mantığı ile dizinin sıralı olup olmadığını kontrol etmede kullanacağım. Proven until guilty mantığı şöyledir, öncelikle sıralı olup olmadığını bilmediğim halde sıralı olduğunu varsayacağım ve daha sonrasında sırasız olduğunu kontrol edip eğer sırasız ise bu değişkeni false yapacağım. Daha sonrasında döngümüze gelecek

olursak bu while döngüsü gap değişkeni 1 değilken veya swapped değişkeni true değilken çalışmalıdır. Bunun nedeni eğer swapped true ise hala sırasız demektir yani devam etmeliyim ve eğer gap 1 ise bu Comb Sort ile Bubble Sort birbirine eşit bir hale gelmiştir ve Comb Sort son bir kez gezerek tüm diziyi sıralı hale getirmiş demektir yani gap 1 değilse de sırasızdır diyebilir ve devam edebilirim. Döngünün içine gelecek olursak öncelikle gap değişkenini 1.3'e bölmeliyim ve eğer gap 1'den küçükse (integer bölmesi yaptığımızdan en sonunda oluşabilecek bir durum bu) gap'i 1'e eşitlemeliyim ki Comb Sort en son turda gerçekten de gerekiyorsa Bubble Sort'a eşit olsun. Daha sonrasında proven until guilty mantığı ile swapped = 0 olsun diyorum ve tüm diziyi gezmeye başlıyorum. Burada eğer baştan (N-gap) uzaklıktaki yere kadar olan her terim kendilerinden gap uzakta olan terimden büyükse bu dizinin hala sırasız olduğu anlamına gelir yani swap yapmam gerekmektedir. If statement'ı ile de aynen bunu kontrol ediyorum ve eğer swapping gerekli ise swap() utility fonksiyonum ile bu değerleri swap edip swapped değişkenimi de 1 yapıyorum ki algoritma çalışması gerektiği gibi çalışmaya devam etsin.

bubbleSort() metoduna gelecek olursak bu metot combSort() metodunun gap=1 olan haline eşittir, o yüzden combSort() metodunu kopyala yapıştır yaptım ve gap güncelleme ile ilgili olan kısımları silip array indekslemelerinde 1 ile değiştirdim çünkü Bubble Sort her zaman bitişik terimlere bakacak ve kontrol yapacak.

Kullanılan kaynaklar

https://www.youtube.com/watch?v=D7wUVtrUHO4

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

https://www.geeksforgeeks.org/comb-sort/

https://www.javatpoint.com/comb-sort

https://mycareerwise.com/programming/category/sorting/comb-sort

https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/

https://en.wikipedia.org/wiki/Comparison_sort#:~:text=A%20comparison%20sort%20is%20a,in%20the%20final%20sorted%20list.

https://hurna.io/academy/algorithms/sort/comb.html

https://hurna.io/assets/files/algorithms/sort/sorting.pdf