

YILDIZ TEKNİK ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ



Öğrenci No: 20011017

Ad – Soyad: Ömer Diner

Öğrenci E-postası: omer.diner@std.yildiz.edu.tr

BLM 1012 -YAPISAL PROGLAMLAMAYA GİRİŞ DÖNEM PROJESİ

QUICKSORT SIRALAMA ALGORİTMASI

Ders Yürütücüsü

Doç. Dr. Mehmet Fatih AMASYALI

Haziran , 2022

İÇERİK

- Quicksort algoritması nedir? Ne işe yarar? Nasıl çalışır?
- Kullanım alanları
- Avantaj ve dezavantajları
- Karmaşıklığı
- Sınırları ve rakipleri
- Çalışmasını açıklarken kullanılan ekran çıktıları
- C dilindeki kodu
- Kaynaklar

VIDEO ADRESİ

<https://drive.google.com/file/d/19CFNkKc4dC2jJE6LPSPZFMv8A-V0QL3/view?usp=sharing>

Quicksort Algoritması

Quicksort , oldukça verimli bir sıralama algoritmasıdır ve veri dizisinin daha küçük dizilere bölünmesine dayanır. Bu algoritma dizideki herhangi bir değeri pivot olarak seçerek diğer bütün elemanları bu pivot elemana göre büyük veya küçük diye sınıflandırır. Daha sonra bu sınıflandırma içerisinde de en alt noktaya ulaşana kadar aynı sınıflandırmayı yapmaya devam eder. Bu açıdan parçala ve fethet yaklaşımlarından bir tanesidir. Seçilen pivottan büyük elemanları pivotun sağ tarafına, küçük elemanları ise sol tarafına koyar. Bu aşamadan sonra pivot elemanın sağında kalan alt diziden ve solundaki alt diziden de yeni pivotlar seçilir ve onlar da quicksort mantığına göre kendi içlerinde pivottan küçük olanların sola , büyük olanların da sağa geçmesiyle sıralanır. Bu şekilde sıralanırken bölünen diziler en son birleştirilir ve ortaya sıralanmış dizi çıkar.

Algoritma adımları şu şekilde özetlenebilir:

1. Diziden herhangi bir eleman pivot eleman olarak seçilir.
2. Dizi, pivot elemandan küçük olan bütün elemanlar pivot elemanın soluna , pivot elemandan büyük olan bütün elemanlar pivot elemanın sağına gelecek biçimde düzenlenir.
3. Quicksort algoritması özyineli çağrılarak , oluşan alt diziler tekrar sıralanır.
4. Algoritma eleman sayısı sıfır olan bir alt diziye ulaşana kadar bu işlem devam eder.
5. Eleman sayısı sıfır olan bir alt diziye ulaşıldığında algoritma bu dizinin sıralanmış olduğunu varsayar ve sıralama işlemi tamamlanmış olur.

Böl ve Fethet

Türkçesi Böl ve Fethet olan Divide and Conquer, bir algorithmadan öte problemi çözmeye yönelik yapılan yaklaşımdır. Kısaca şöyle açıklayabiliriz ki elimizdeki problemi uygun bir şekilde benzer problemlere parçalıyoruz. Daha sonra bu parçaları kendi içerisinde ayrı ayrı çözüyoruz ve bütün ufak parçaları çözdükten sonra bu parçaları birleştirerek gerçek çözüme ulaşıyoruz.

Avantajları

- Sıralama esnasında , ek alana ihtiyaç duyulmaz. Bellek kullanımı azdır.
- Quicksort, sıralamayı hızlı ve verimli bir şekilde gerçekleştirmek için en çok tercih edilen algoritmaların başında gelir. Kullanıcıların aynı sonuca diğer sıralama algoritmalarından çok daha hızlı ulaşmasını sağlar.

Dezavantajları

- Kırılgandır, özyinelemeli olduğundan uygulamadaki basit bir hata fark edilmeyebilir ve kötü performans göstermesine neden olabilir.
- En kötü senaryoda, ikinci dereceden (yani n^2) zaman alır.

Karmaşıklığı

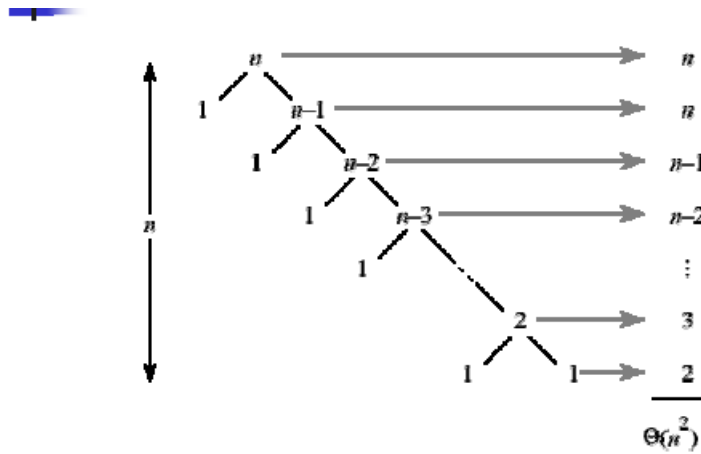
Her seferinde diziyi ikiye bölmeye çalışıyoruz ve o iki kısmı kendi içerisinde sıralıyoruz , bu nedenle yapacağımız bölme sayısı yaklaşık $\log_2(N)$ 'dir . Partition fonksiyonundan da gelen her alt dizi için toplam N işlemi katarsak algoritmanın karmaşıklığı $O(N \log_2(N))$ ' dir. Ancak burada dikkat etmemiz gereken nokta , seçeceğimiz pivot elemanının her zaman seçilen dizinin en ortasında bulunamayacağı için her seferinde diziyi tam ortadan bölemeyeceğimiz durumların da bulunmasıdır. Bu yüzden en kötü ihtimalle N^2 karmaşıklığa gitme ihtimali de bulunmaktadır.

Yani quicksortta şanslıysak ve güzel pivotlar gelirse hep iki eşit parçaya bölerek gitmek isteriz. ($O(n)=n \cdot \log n$). Şanssızsak kötü pivotlar seçerek diziyi hiç parçalayamadan gideriz ($O(n)= n^2$).

Zaman Karmaşıklığı

- **En kötü zaman: $O(n^2)$**

Pivot olarak seçilen öge listedeki en küçük veya en büyük öge olduğunda en kötü durum olarak kabul edilir. Bu genellikle liste neredeyse sıralı olduğunda olur. Seçilen pivot en büyük eleman olduğunda, dizi $n-1$ 'e bölünür. Bölme işlemleri eşit olmayacak, bu nedenle ağaç yüksekliğinde önemli bir fark olacaktır. Ağacın uzunluğu n 'dir ve aşağı doğru gittikçe fonksiyonun yaptığı iş de n , $n-1$, $n-2$... diye değişecektir. Dolayısıyla, quicksortun en kötü zaman karmaşıklığı $O(n^2)$ 'dir.



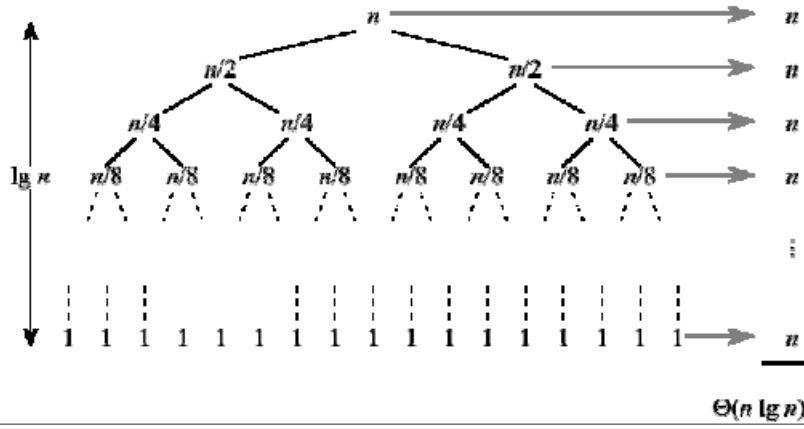
- **Ortalama zaman: $O(n * \log n)$**

Quicksortta, eşit olarak dengeli alt diziler elde edemediğimiz durum ortalama durum olarak kabul edilir ve ortalama durum zaman karmaşıklığı $O(n * \log n)$ olur.

- En iyi zaman: $O(n * \log n)$

Ortalama eleman pivot olarak seçildiğindeki durum en iyi durum olarak kabul edilir. Pivotun diziyi böldüğünü biliyoruz ve pivot olarak ortalama bir eleman seçildiğinde, dizi eşit alt dizilere bölünür. Ağacın yüksekliği minimum olur ve bu durumda alt seviyelerin sayısı $\log_2 n$ 'dir. Bu nedenle, quicksortun en iyi durum zaman karmaşıklığı $O(n * \log n)$ 'dir.

- Partition diziyi düzgün bir şekilde ikiye böler $T(n) = 2T(n/2) + \Theta(n)$



RAKİPLERİ

Quicksort genel olarak ortalama duruma bakıldığında en hızlı algoritma kabul edilir ve sıklıkla kullanılır. Farklı veri setleri için onlara uygun algoritmalar kullanmak gerekse de genel olarak aşağıdaki tablodan da görebileceğimiz gibi quicksort çok önemli bir sıralama algoritmasıdır.

Algoritma	En iyi Durum (Best Case)	Ortalama Durum (Average Case)	En kötü Durum (Worst Case)
Seçerek Sıralama (Selection Sort)	$O(n^2)$	$O(n^2)$	$O(n^2)$
Kabarcık Sıralaması (Bubble Sort)	$O(n)$	$O(n^2)$	$O(n^2)$
Ekleme Sıralama (Insertion Sort)	$O(n)$	$O(n^2)$	$O(n^2)$
Birleştirme Sıralaması (Merge Sort)	$O(n * \log n)$	$O(n * \log n)$	$O(n * \log n)$
Hızlı Sıralama (Quick Sort)	$O(n * \log n)$	$O(n * \log n)$	$O(n^2)$

EKRAN ÇIKTILARI

```
C:\Users\OMER\Desktop\20011017.exe
Enter the size of the input array:10
8 1 6 5 9 4 7 5 9 7
Array after sorted:
1 4 5 5 6 7 7 8 9 9
CPU time for algorithm to run was 0.000 .
Total 19 function(quicksort+partition) calls.

-----
Process exited after 3.979 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\OMER\Desktop\20011017.exe
Enter the size of the input array:10
6 3 5 8 4 4 7 1 7 5
Array after sorted:
1 3 4 4 5 5 6 7 7 8
CPU time for algorithm to run was 0.000 .
Total 25 function(quicksort+partition) calls.

-----
Process exited after 0.6301 seconds with return value 0
Press any key to continue . . .
```

Bu iki çıktıdan görülebileceği gibi verilen dizinin rastgeleliğine bağlı olarak toplam işlem sayısı aynı boyuttaki dizilerde de farklılık gösterebilir.

```
C:\Users\OMER\Desktop\yggprojesi\20011017.exe
Enter the size of the input array:50000

CPU time for algorithm to run was 15.000 .
Total 103282 function(quick sort+partition) calls.
Visual representation of complexity(one star for each CPU time): *****
-----
Process exited after 2.805 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\OMER\Desktop\yggprojesi\20011017.exe
Enter the size of the input array:100000

CPU time for algorithm to run was 24.000 .
Total 223174 function(quick sort+partition) calls.
Visual representation of complexity(one star for each CPU time): *****
-----
Process exited after 2.241 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\OMER\Desktop\yggprojesi\20011017.exe
Enter the size of the input array:200000

CPU time for algorithm to run was 50.000 .
Total 503857 function(quick sort+partition) calls.
Visual representation of complexity(one star for each CPU time): *****
-----
Process exited after 1.435 seconds with return value 0
Press any key to continue . . .
```

```
C:\Users\OMER\Desktop\yggprojesi\20011017.exe
Enter the size of the input array:400000

CPU time for algorithm to run was 95.000 .
Total 1101706 function(quick sort+partition) calls.
Visual representation of complexity(one star for each CPU time): *****
-----
Process exited after 2.084 seconds with return value 0
Press any key to continue . . .
```

Çıktılarda da görüldüğü üzere ilk başta 50.000 olarak girdiğim input sayısını diğer çalıştırmalarımda hep 2 katına çıkartarak 400.000'e kadar denemeler yaptım. Dizi boyutu iki katına çıksa da çalışma zamanı ve fonksiyon çağrıları yaklaşık olarak iki kattan biraz daha az veya biraz daha fazla bir artış gösterdi. Ama boyut iki katına çıktıkça göstergelerin 4 katına çıkmaması , açıkça bu algoritmanın $O(N^2)$ gibi verimsiz bir karmaşıklıkla çalışmadığını gösteriyor.

KODLAR

20011017.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <math.h>
5  #define SIZE 500000
6
7  void quickSort(int * arr, int left, int right, int * counter);
8  int partition(int * arr, int left, int right, int * counter);
9  void generateRandomNumbersForArr(int * arr, int size);
10 void printArray(int * arr, int size);
11 void swap(int * a, int * b);
12
13 int main() {
14     //for the random function to work correctly
15     srand(time(NULL));
16     //variable declarations
17     clock_t start, stop;
18     double duration;
19     int arr[SIZE];
20     int * counter;
21     int count = 0, n, i, stars;
22     counter = &count; //i used the counter as a pointer to access this variable from other functions
23
24     printf("Enter the size of the input array:");
25     scanf("%d", &n);
26
```

```
27 //assigning random numbers to an array so that the user doesn't have to enter
28 generateRandomNumbersForArr(arr, n);
29 //printArray(arr,n);
30
31 start = clock(); // cpu beginning time    clock()->This function returns the number of clock ticks elapsed since the start of the program.
32
33 //calling quicksort with the left value=0 and right value=size-1
34 quickSort(arr, 0, n - 1, counter);
35
36 stop = clock(); // cpu time after the sort algorithm executes
37
38 duration = (double)(stop - start); //elapsed time
39
40 stars = duration ;
41
42 //printf("\nArray after sorted:\n");
43 //printArray(arr,n);
44
45 printf("\nCPU time for algorithm to run was %.3lf .", duration);
46
47 //writes n and the total number of function calls made
48 printf("\nTotal %d function(quickSort+partition) calls.\n", count);
49
50 printf("Visual representation of complexity(one star for each CPU time): ");
51 // a notation with stars to better visually see the complexity
52 for (i = 0; i < stars; i++) {
53     printf("*");
54 }
55
56 return 0;
57 }
```

```
58 void quickSort(int * arr, int left, int right, int * counter) {
59     (* counter)++;
60
61     if (left < right) {
62         int pivot = partition(arr, left, right, counter); //returns the correct position of the pivot we selected
63         quickSort(arr, left, pivot - 1, counter); //sorts the part before the pivot
64         quickSort(arr, pivot + 1, right, counter); //sorts the part after the pivot
65     }
66 }
67
68
69
70
71
72
```

```





73 int partition(int * arr, int left, int right, int * counter) {
74     (* counter) ++;
75
76     int j; // right pointer
77     int pivot = arr[right]; //i chose the pivot as the last element of the array
78
79     int i = (left - 1); // left pointer
80
81     // traverse each value between left and right and compare with pivot
82     for (j = left; j < right; j++) {
83
84         if (arr[j] <= pivot) {
85
86             // if element smaller than pivot is found , swap it with the bigger element pointed by i
87             i++;
88
89             // swap element at i with element at j
90             swap(& arr[i], & arr[j]);
91         }
92     }
93
94     swap(& arr[i + 1], & arr[right]); //finally puts the pivot in its proper position
95
96     return i + 1; //index of the pivot
97 }
98

```

```

98
99 void generateRandomNumbersForArr(int * arr, int size) {
100     int i;
101     for (i = 0; i < size; i++) {
102         arr[i] = rand() % (size); // generates random numbers up to size
103     }
104 }
105
106
107 //simple swap function using pointers
108 void swap(int * a, int * b) {
109     int tmp = * a;
110     * a = * b;
111     * b = tmp;
112 }
113
114 void printArray(int * arr, int size) {
115     int i;
116     for (i = 0; i < size; i++) {
117         printf("%d ", arr[i]);
118     }
119 }

```

 Compile Log
  Debug
  Find Results
  Close

KAYNAKLAR

<https://bilgisayarkavramlari.com/2008/08/09/hizli-siralama-algoritmasi-quick-sort-algorithm/>

<https://medium.com/human-in-a-machine-world/quicksort-the-best-sorting-algorithm-6ab461b5a9d0>

<https://www.techquintal.com/advantages-and-disadvantages-of-quick-sort/>

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_quick_sort.htm

<https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort>

<https://www.simplilearn.com/tutorials/data-structure-tutorial/quick-sort-algorithm>