# Halime Nur İzmirli – File organizations Homework 4

İçindekiler

[Halime Nur İzmirli – File organizations Homework 4 1](#_Toc166927869)

[Dosya oluşturma 1](#_Toc166927870)

[Random sınıfı ile sayı üretme 1](#_Toc166927871)

[Sıralama Fonksiyonlarının Sürelerinin ölçülmesi 2](#_Toc166927872)

[Selection ile Sıralama - selectionSort(int arrayOfData[]) 2](#_Toc166927873)

[Insertion ile Sıralama - insertionSort(int arrayOfData[]) 3](#_Toc166927874)

[Merge ile Sıralama - mergeSort(int arrayOfData[], int firstIndex, int lastIndex) 3](#_Toc166927875)

[Sonuçların değerlendirilmesi 4](#_Toc166927876)

# Dosya oluşturma

static File *file*;

static FileWriter *fileWriter*;

static BufferedWriter *bufferedWriter*;

*file* = new File("Numbers.txt");

*fileWriter* = new FileWriter(*file*);

*bufferedWriter* = new BufferedWriter(*fileWriter*);

Programın başlangıcında 1000 adet sayıyı kaydetmek için dosya oluşturdum.

final static int ***numberOfData*** = 1000;

Sıralanacak integer veri miktarı “numberOfData”da tutuluyor. Ödevde istenildiği gibi 1000 olarak ayarlandı.

# Random sınıfı ile sayı üretme

Random random = new Random();

for(int i=0; i<***numberOfData***; i++) {

*bufferedWriter*.write(random.nextInt()+"\n");

}

For döngüsü içinde 1000 adet random sayı üretiliyor ve bu sayılar dosyaya yazılıyor.

int arrayOfData[]=new int[***numberOfData***];

int arrayOfData2[]=new int[***numberOfData***];

int arrayOfData3[]=new int[***numberOfData***];

int index=0;

int number;

Scanner scan = new Scanner(*file*);

while(scan.hasNextInt()) {

number=scan.nextInt();

arrayOfData1[index] = number;

arrayOfData2[index] = number;

arrayOfData3[index] = number;

index++;

}

Dosyadan okunan veriler “arrayOfData” isimli dizilerde tutuluyor. 3 farklı algoritma ile sıralama yapılacağı için 3 farklı array oluşturdum (arrayOfData1, arrayOfData2, arrayOfData3). Dosyadan okuma işlemi Scanner sınıfı ile yapılıyor.

# Sıralama Fonksiyonlarının Sürelerinin ölçülmesi

System.***out***.println("Selection ile siralama: ");

long startTime = System.*nanoTime*();

*selectionSort*(arrayOfData1);

long endTime = System.*nanoTime*();

System.***out***.println("Gecen süre: "+ ((double)(endTime-startTime)/1000000000));

Süreler, System.nanoTime() ile ölçülüp long tipindeki değişkenlere kaydediliyor. Fonksiyonu çağırmadan önceki süre “startTime”, fonksiyon çağırdıktan sonraki süre ölçülüp “endTime” olarak tutuluyor. Algoritmanın çalışma süresi nanosaniye cinsinden (endTime - startTime) olarak bulunuyor. Saniyeye çevirmek için 1 trilyona bölünüyor. Aynı işlem, diğer fonksiyonlar için de yapılıyor:

System.***out***.println("\nInsertion ile siralama: ");

startTime = System.*nanoTime*();

*insertionSort*(arrayOfData2);

endTime = System.*nanoTime*();

System.***out***.println("Gecen süre: "+ ((double)(endTime-startTime)/1000000000));

System.***out***.println("\nMerge ile siralama: ");

startTime = System.*nanoTime*();

*mergeSort*(arrayOfData3, 0, ***numberOfData***-1);

endTime = System.*nanoTime*();

System.***out***.println("Gecen süre: "+ ((double)(endTime-startTime)/1000000000));

Main fonksiyonu burada bitmektedir. Programın çıktısı aşağıdaki gibidir:

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

## Selection ile Sıralama - selectionSort(int arrayOfData[])

static public void selectionSort(int arrayOfData[]) throws IOException {

int min;

for(int i=0; i<***numberOfData***-1; i++) {

min=i;

for(int j=i+1; j<***numberOfData***; j++) {

if(arrayOfData[j]<arrayOfData[min]) {

min=j;

}

}

int temp = arrayOfData[i];

arrayOfData[i]=arrayOfData[min];

arrayOfData[min]=temp;

}

}

Fonksiyona parametre olarak arrayOfData1 isimli dizi yollanıyor. Bu dizi, dosyadaki verileri yazdırdığımız dizidir. Minimum sayının indexi “min” değişkeninde tutuluyor. İlk indeksten başlayarak sırayla bütün sayılar kontrol ediliyor. Küçük olanın indeksi minimum olarak atanıyor. Her adımda en küçük eleman başa alınıyor. (selection sort algoritması)

## Insertion ile Sıralama - insertionSort(int arrayOfData[])

public static void insertionSort(int arrayOfData[]) throws IOException {

int key;

for(int i=1; i<***numberOfData***; i++) {

key=arrayOfData[i];

int j=i-1;

while (j >= 0 && arrayOfData[j] > key) {

arrayOfData[j + 1] = arrayOfData[j];

j = j - 1;

}

arrayOfData[j + 1] = key;

}

}

Fonksiyona parametre olarak arrayOfData2 dizisini yolluyoruz. Fonksiyon bir anahtar değeri belirliyor. İlk adımda indeksi 1 olan yani dizideki 2. eleman, anahtar olarak seçiliyor. For döngüsü, sonraki adımlarda sırayla bir sonraki elemanı anahtar olarak seçiyor. Her anahtar değerinden başlayarak ilk elemana doğru büyüklük kontrolü yapıyo. Eğer anahtar değeri daha küçük ise kendinden önceki elemanın değerini bir sonraki elemana koyuyor. Anahtar değerinden ilk elemana kadar bütün kontroller yapıldıktan sonra anahtar değeri başa yazılıyor.

## Merge ile Sıralama - mergeSort(int arrayOfData[], int firstIndex, int lastIndex)

Bu algoritma için 2 fonksiyon kullanılıyor: mergeSort() ve merge().

public static void mergeSort(int arrayOfData[], int firstIndex, int lastIndex){

if(firstIndex<lastIndex) {

int middleIndex = (firstIndex+lastIndex)/2;

*mergeSort*(arrayOfData, firstIndex, middleIndex);

*mergeSort*(arrayOfData, middleIndex+1, lastIndex);

*merge*(arrayOfData, firstIndex, middleIndex, lastIndex);

}

}

mergeSort() fonksiyonunda, dizi tek elemana düşene kadar 2’ye bölüyor. Sonra merge() fonksiyonu çağırılıyor.

public static void merge(int arrayOfData[], int firstIndex, int middleIndex, int lastIndex) {

int n1 = middleIndex-firstIndex+1;

int n2 = lastIndex-middleIndex;

int L[] = new int[n1];

int R[] = new int[n2];

for (int i = 0; i < n1; ++i) {

L[i] = arrayOfData[firstIndex + i]; }

for (int j = 0; j < n2; ++j) {

R[j] = arrayOfData[middleIndex + 1 + j]; }

Parametre olarak verilen diziyi (arrayOfData[]), sağ (L[]) ve sol (R[]) olarak iki diziye ayırıyor.

int i = 0, j = 0;

int t=firstIndex;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arrayOfData[t] = L[i];

i++;

}

else {

arrayOfData[t] = R[j];

j++;

}

t++;

}

while (i < n1) {

arrayOfData[t] = L[i];

i++;

t++;

}

while (j < n2) {

arrayOfData[t] = R[j];

j++;

t++;

}

}

İki diziyi sırayla karşılaştırıyor ve küçükten büyüğe ilk verilen diziye (arrayOfData[]) yerleştiriyor.

# Sonuçların değerlendirilmesi

Dosyalar oluşturuldu.

Selection ile siralama:

Gecen süre: 0.0052421

Insertion ile siralama:

Gecen süre: 0.0070299

Merge ile siralama:

Gecen süre: 0.0028662 (output 1)

Dosyalar oluşturuldu.

Selection ile siralama:

Gecen süre: 0.0074309

Insertion ile siralama:

Gecen süre: 0.0075872

Merge ile siralama:

Gecen süre: 0.0045971 (output 2)

Dosyalar oluşturuldu.

Selection ile siralama:

Gecen süre: 0.0097913

Insertion ile siralama:

Gecen süre: 0.0064098

Merge ile siralama:

Gecen süre: 0.0055832 (output 3)

Dosyalar oluşturuldu.

Selection ile siralama:

Gecen süre: 0.0091037

Insertion ile siralama:

Gecen süre: 0.0070311

Merge ile siralama:

Gecen süre: 0.0022184 (output 4)

Yukarıda programın 4 farklı çıktısı verilmiştir. Bu sonuçlardan yola çıkarak; merge algoritmasının en kısa sürede sıralama yaptığını, selection ve insertion algoritmalarının ise kendi aralarında kıyaslandığında bazen selection, bazen insertion fonksiyonunun daha kısa sürede sıralama yaptığını söyleyebiliriz. Verilen dizinin karmaşıklığına göre algoritmaların çalışma süreleri değişmektedir.