Ders Adı: Algoritma Analizi

Öğrenci Bilgileri: Ömer Buğrahan Çalışkan – 17011076

Ödev İçeriği: N elemanlı dizide birbirine en yakın değere sahip iki elemanın brute-force ve quick sort algoritmalarıyla çözülerek verimliliğin karşılaştırması.

Brute-Force ve QuickSort Çözümlerin Analizi ve Farkları:

Brute-Force çözümünde iç içe 2 for döngüsü kullandığımız için karmaşıklığımız her durumda O(n^2), fakat QuickSort algoritmasında average ve best case de O n(log(n)), worst case de O(n^2) olduğundan worst case dışındaki durumlarda çözümümüz daha hızlı ve verimli olacaktır. Somut bir şekilde bu farkı görebilmek için 1.000, 10.000, 50.000 elemanlı dizilere 1 ve 100 arasinda random atama yaparak Çalışma Sürelerini karşılaştırdım.

Aldığım Sonuçlar şu şekilde:

	1.000	10.000	50.000
Brute-Force:	0.03 ms	0.204 ms	5.067 ms
QuickSort:	0 ms	0.04 ms	0.048 ms

Verilerden de görüldüğü üzere düşük dizi boyutlarında farkedilmese de dizi boyutu arttıkça brute-force çözümün quicksorta göre daha yavaş çalıştığı ve verimsiz olduğu görülmektedir.

Brute-Force Kod ve Açıklaması:

Brute-Force çözümde eleman sayısını ve elemanları aldıktan sonra iç içe 2 for döngüsü açarak bütün elemanları birbirleri ile karşılaştırdım eğer fark değerimizden daha küçük bir fark bulursa değerlerimiz güncellenecek. Dizi sıralı olmadığı için değerlerin farkını abs fonksiyonunu kullanarak mutlak değerli bir şekilde hesapladım. Çalışma süresinin hesabını doğru yapabilmek için ilgili kodu program başladığı anda değil hesaplama işleminin başladığı ve bittiği anlara yazdım.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
int main(){
        int n; // Dizi eleman sayısı
```

```
	int i,j; // Döngüler için gerekli değişken
	int tmp1,tmp2; // Aradığımız elemanların indisini tutan değişkenler
	clock_t start,end; // Programın Çalışma süresinin hesabı için oluşturulan değişkenler
	printf("Dizi eleman sayisini giriniz: ");
	scanf("%d",&n);

	int dizi[n];

	for(i=0;i<n;i++){
	printf(" %d. elemanin degerini giriniz: ",i);
	scanf("%d",&dizi[i]);
	}

	int fark = 9999; // Programın çalışabilmesi ve ilk farkı bulabilmesi için fark değişkenine büyük bir değer atıyoruz.
	start = clock(); // Sayacın başlatılması.
	for(i=0;i<n-1;i++){
		for(j=i+1;j<n;j++){
			if(fark > abs(dizi[i]-dizi[j])){ // Eğer fark değerimiz kontrol ettiğimiz elemanların farkından büyükse bu elemanların yerini tutuyoruz ve fark değerimizi güncelliyoruz.
				tmp1 = i;
				tmp2 = j;
				fark = abs(dizi[i]-dizi[j]);
			}
		}
	}
	end = clock(); // Sayacın Bitişi.
		printf("\n%d ve %d indexindeki elemanlar arasindaki minimum fark: %d",tmp1,tmp2,fark);
		printf("\nRunning Time: %f ms",(double)(end-start)/CLOCKS_PER_SEC);
}
```

QuickSort Kod ve Açıklaması:

Brute-Force Çözümüne kıyasla daha verimli ve hızlı olacağını düşündüğümden diziyi QuickSort algoritması ile sıralayıp ardından en yakın 2 değeri bulmaya karar verdim. Veri Yapıları ve Algoritmalar dersinde öğrendiğimiz QuickSort algoritması bir pivot seçilerek

elemanların bu değerden büyük ve küçük olmasına göre sağına veya soluna atma esasına dayanır kodda gerekli açıklamalar satır satır yapılmıştır. Dizi açıldıktan ve elemanlar alındıktan sonra dizimiz sıralanacak fakat dizi sıralandığı takdirde dizinin orijinal halindeki elemanların indisleri değişecek bu yüzden diziyi sıralamadan önce orjdizi adında farklı bir diziye kopyaladık. Sıralama, fark ve minimum farklarımızın indislerini tutan değerleri bulduktan sonra bu değerleri sıralanmamış orjdizi dizisinde aradık ve bulduğumuzda tmp değerini değiştirip break komutuyla döngüden çıktık. Çalışma süresinin hesabını doğru yapabilmek için ilgili kodu program başladığı anda değil hesaplama işleminin başladığı ve bittiği anlara yazdım.

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

void quickSort(int *dizi,int first,int last){
   int i,j;
   int pivot; // Pivot elemanı tutacak sayaç değişkeni
   int tmp; // Yer değiştirme işlemi için kullanılacak değişken
   pivot = first; // Pivota ilk eleman atiyoruz

   // Eğer son eleman ilk elemandan büyükse, son eleman ilk elemandan büyük olduğu sürece baştan ve sondan pivottan büyük olan ve pivottan küçük olan bir eleman seçip yer değiştiriyoruz
   if(last>first){
      pivot=first;
      i=first;
      j=last;
      while (i<j){
         while (dizi[i]<=dizi[pivot] && i<last && j>i){ // Baştan pivottan büyük olan bir eleman buluyoruz
            i++;
         }
         while (dizi[j]>=dizi[pivot] && j>=first && j>=i){ // Sondan pivottan küçük olan bir eleman buluyoruz
            j--;
         }
         if (j>i){ // Swap işlemi yapılır
```

```
            tmp=dizi[i];
            dizi[i]=dizi[j];
            dizi[j]=tmp;
          }
      }
      // Yeniden pivot seçiyoruz ve bölünen bağlı listenin diğer parçalarını tekrar quick sort fonksiyonuna gönderiyoruz
      tmp=dizi[j];
      dizi[j]=dizi[pivot];
      dizi[pivot]=tmp;
      quickSort(dizi,first,j-1);
      quickSort(dizi,j+1,last);
   }
}
int main(){
            int i,j; // Döngüler için gerekli değişken
            int tmp1,tmp2; // Aradığımız elemanların indisini tutan değişkenler
            clock_t start,end;  // Programın Çalışma süresinin hesabı için oluşturulan değişkenler

            int *dizi,n; // Dizimiz ve boyutunu tutacak n değeri.
   printf("Dizinin boyutunu giriniz: ");
            scanf("%d",&n);
   dizi=(int *)malloc(n*sizeof(int));
   for(i=0;i<n;i++){
      printf("Dizinin %d. elemani :",i);
                        scanf("%d",&dizi[i]);
   }
            printf("\nDizinin siralanmamis durumu: ");
   for(i=0;i<n;i++){
      printf("%d ",dizi[i]);
   }
   // Diziyi sıralayınca indislerin yeri değişeceği için dizinin sıralanmamış orjinal halini farklı bir dizi açıp ona atıyoruz.
   int orjdizi[n];
```

```
	for(i=0;i<n;i++){
		orjdizi[i]=dizi[i];
	}
	start = clock(); // Sayacın başlatılması.
	quickSort(dizi,0,n-1); // Dizinin quicksort ile sıralanmasi. Fonksiyona diziyle beraber ilk ve son eleman indisi gonderiliyor.
   printf("\nDizinin siralanmis durumu: ");
   for(i=0;i<n;i++){
      printf("%d ",dizi[i]);
   }
	int fark = 9999; // Programın çalışabilmesi ve ilk farkı bulabilmesi için fark değişkenine büyük bir değer atıyoruz.
	for(i=0;i<n-1;i++){
		if(fark > dizi[i+1] - dizi[i]){ // Eğer fark değerimiz kontrol ettiğimiz elemanların farkından büyükse bu elemanların yerini tutuyoruz ve fark değerimizi güncelliyoruz.
			tmp1 = i;
			tmp2 = i+1;
			fark = dizi[i+1]-dizi[i];
		}
	}
	// Sıraladığımız dizide elemanların yerleri değiştiği için değişmeden önceki yerlerini bulup indisleri tutan tmp değerlerimizi güncelliyoruz.
	for(i=0;i<n;i++){
		if(dizi[tmp1]==orjdizi[i]){
			tmp1=i;
			break;
			}
		}
		for(i=0;i<n;i++){
			if(dizi[tmp2]==orjdizi[i]){
				tmp2=i;
				break;
			}
		}
	end = clock(); // Sayacın Bitişi.
```

```
        printf("\n\n%d ve %d indexindeki elemanlar arasindaki minimum fark: %d",tmp1,tmp2,fark);
        printf("\nRunning Time: %f ms",(double)(end-start)/CLOCKS_PER_SEC);
}
```