YILDIZ TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Intro Sort Algoritması

Öğrenci No : **20011044**
Öğrenci Adı Soyadı : **Yusuf Enes Kurt**
Öğrenci e-posta : **l1120044@std.yildiz.edu.tr**

Ders/Grup: **BLM1012 YAPISAL PROGRAMLAMAYA GİRİŞ / Gr-2**

Final Proje Raporu

Ders Yürütücüsü
Öğr. Gör. Dr. **Ahmet Elbir**
Haziran, 2021

Intro Sort Nedir ?

Quick Sort , Heap Sort ve Insertion Sort algoritmalarını kullanan hibrit bir sıralama algoritmasıdır.

Nasıl Çalışır ?

Intro Sort , Quick Sort'la başlar ve eğer recursion derinliği belirl, bir sınırı geçerse Quick Sort'un en kötü zaman karmaşıklığı olan O(N^2)'yi önlemek için Heap Sort'a geçer. Ayrıca sıralanacak öğe sayısı oldukça az olduğunda Insertion Sort kullanır. Buradan 3 bölüm oluşturulur;

1- Eğer bölme boyutu , maksimum derinlik sınırını aşma olasılığı olacak şekildeyse Intro Sort , Heap Sort'a geçer. Maksimum derinlik limitini 2*log(N) olarak tanımlıyoruz.
2- Eğer bölme boyutu çok küçükse Quick Sort , Insertion Sort'a dönüşür. Bu sınır 16 olarak kabul görmüştür. Bu nedenle bölme boyutu 16'dan küçükse Insertion Sort kullanılır.
3- Eğer bölme boyutu sınırın altında ve çok küçük değilse(yani 16-2*log(N) arasında) o zaman basit bir Quick Sort gerçekleştirilir.

Intro Sort'un Quick Sort'a Göre Avantajları

Quick Sort , O(N^2) gibi daha kötü bir zaman karışıklığına sahip olabiliyor ve ayrıca recursion stack alanını arttırabiliyor. Tüm bunlardan kaçınmak için , algoritmayı Quick Sort'tan başkasına çevirmemiz gerekir. Intro Sort bu sorunu Heap Sort'a geçerek çözer.

Aynı zamanda daha büyük sabit faktör nedeniyle N yeterince küçük olduğunda Quick Sort , O(N^2) Sort algoritmasından bile daha kötü performans gösterebilir. Böylece sıralamanın çalışma süresini azaltmak için Insertion Sort'a geçilir.

Neden Insertion Sort Kullanılıyor ?(Neden Bubble Değil ?)

1- Insertion Sort küçük diziler için en uygun karşılaştırma tabanlı sıralama algoritmasıdır.
2- İyi bir referans yeri vardır.
3- Uyarlanabilir bir sıralama algoritmasıdır.(Yani dizi elemanları kısmen sıralanmışsa diğer tüm algoritmalardan daha iyi performans gösterir.)

Neden Heap Sort Kullanılıyor ?(Neden Merge Sort Değil ?)

Bunun tek sebebi bellek gereksinimidir. Merge Sort O(N) alan gerektirirken Heap Sort O(1) alan gerektirir.

16 ve 2*log(N) Kriterleri Nereden Geliyor ?

Bu değerler , yapılan çeşitli testler ve araştırmalar nedeniyle ampirik olarak seçilmiştir.

Verimlilik

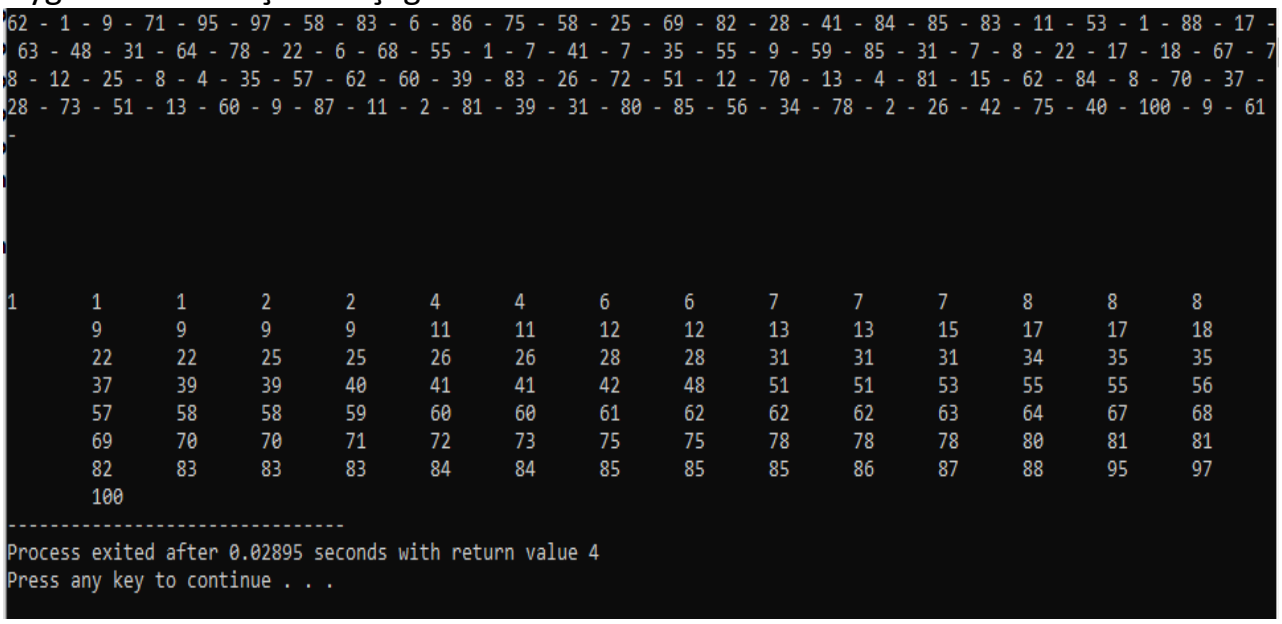
$$\begin{aligned}\mathcal{O}(n) &= \sum_{i=0}^{r}(n-i) + (n-r)\cdot log(n-r) \\ &= \sum_{i=0}^{r} n - \sum_{i=0}^{r} i + (n-r)\cdot log(n-r) \\ &= r\cdot n - \frac{r\cdot(r+1)}{2} + (n-r)\cdot log(n-r)\end{aligned}$$

Basic Operation Kıyaslamaları

Length (in 1000)	algorithm	assignments	comparisons	total
1	Introsort	21.6	17.7	39.3
	Quicksort (opt.)	21.5	17.7	39.2
	Quicksort	22.4	23.2	45.5
	Heapsort	36.1	36.3	72.4
4	Introsort	102.7	83.1	185.8
	Quicksort (opt.)	102.2	83.1	185.4
	Quicksort	105.4	106.2	211.6
	Heapsort	172.5	177.4	349.9
16	Introsort	470.9	377.8	848.7
	Quicksort (opt.)	469.2	377.8	847.0
	Quicksort	495.7	493.8	989.5
	Heapsort	803.3	838.3	1641.6
64	Introsort	2141.2	1709.3	3850.5
	Quicksort (opt.)	2134.6	1709.3	3843.9
	Quicksort	2198.8	2155.5	4354.3
	Heapsort	3658.6	3863.1	7521.7
256	Introsort	9629.7	7666.5	17296.3
	Quicksort (opt.)	9603.1	7666.5	17269.6
	Quicksort	9790.2	9499.1	19289.4
	Heapsort	16423.7	17498.0	33921.7
1024	Introsort	42482.9	33676.1	76159.0
	Quicksort (opt.)	42376.6	33676.1	76052.7
	Quicksort	43347.4	41711.0	85058.4
	Heapsort	72873.9	78192.9	151066.7

Uygulama

Uygulama ekran çıktısı aşağıdadır.



C Program Kodu

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

#define MAX 100

void IntroSort(int* data, int begin , int end );
void IntrosortUtil(int* data, int begin, int end, int depthLimit );
void InsertionSort(int* data, int begin , int end);
void MaxHeapify(int* data, int heapSize, int index , int begin);
void Heapify(int* data, int begin, int end, int heapSize );
void HeapSort(int* data, int begin, int end );
int Bolme(int* data, int left, int right );

int main(){

	int data[MAX];
	int i;

	srand(time(NULL));
	for(i=0 ; i<MAX ; i++){
		data[i] =1+rand()%100;
		printf("%d - " , data[i]);
	}
	printf("\n\n\n");


	IntroSort(data, 0, MAX-1);

	printf("\n\n\n");

	for(i=0 ; i<MAX ; i++){
		printf("%d\t" , data[i]);
	}

}
void swap(int* data, int i, int j){

  int temp = data[i];
```

```
  data[i] = data[j];
  data[j] = temp;

}

void InsertionSort(int* data, int begin , int end ) {              //Kısmen
sıralanmış dizilerde oldukça verimlidir.
        int i , j , key , left , right;

        left = begin;
        right = end;

        for (i = left+1 ; i <= right ; i++)
        {

                key = data[i];
                j = i - 1;


                while ( (j >= left) && (data[j] > key) )
                {


                        data[j + 1] = data[j];
                        j--;
                }
                data[j + 1] = key;
        }
}

void MaxHeapify(int* data, int heapSize, int index , int begin) {
        int left = (index + 1) * 2 - 1;                          // Üçgen kurulur
        int right = (index + 1) * 2;
        int largest = 0;
        int temp;
        int i;


        if (left < heapSize && data[begin+left] > data[begin+index])
                largest = left;
```

```
	else
		largest = index;

	if (right < heapSize && data[begin+right] > data[begin+largest])
		largest = right;

	if (largest != index)                                   //en büyük sayı
üçgenin üstüne gelir
	{
		temp = data[begin+index];
		data[begin+index] = data[begin+largest];
		data[begin+largest] = temp;



		MaxHeapify(data, heapSize, largest , begin);
	}
}

void HeapSort(int* data, int begin , int end) {
	int heapSize = end-begin+1;
	int p , i , j , temp;

	for (p = (heapSize-1 ) / 2 ; p >= 0 ; p--)                    //Döngü
tamamlandığında herhangi bir sayı kendisinden büyük bir sayının üstünde
bulunmaz.
		MaxHeapify(data, heapSize, p , begin);                  //Yani en
büyük sayı ağacın en üstündedir.

	for (i = end-begin ; i > 0 ; i--)
	{

		temp = data[begin+i];
		data[begin+i] = data[begin];
		data[begin] = temp;

		heapSize--;
		MaxHeapify(data, heapSize, begin , begin);
	}
```

```
}

int Bolme(int* data, int left, int right ) {          // son elemanı bulunması
gereken yere koyar ve kendisinden küçük elemanları soluna koyar
        int pivot = data[right];
        int temp;
        int i = left;
        int j = left;

        for (j = left ; j < right ; j++)
        {


                if (data[j] <= pivot)
                {
                        temp = data[j];
                        data[j] = data[i];
                        data[i] = temp;
                        i++;
                }
        }

        data[right] = data[i];
        data[i] = pivot;

        return i;                                       // son elemanın en son konumunu
döndürür
}


int findPivot(int* data, int a1, int b1, int c1)
{
        if(data[a1] < data[b1] && data[b1] < data[c1])
                return b1;

        if(data[a1] < data[c1] && data[c1] < data[b1])
                return c1;

        if(data[b1] < data[a1] && data[a1] < data[c1])
                return a1;
```

```
        if(data[b1] < data[c1] && data[c1] < data[a1])
                return c1;

        if(data[c1] < data[a1] && data[a1] < data[b1])
                return a1;

        if(data[c1] < data[b1] && data[b1] < data[a1])
                return b1;
}

void IntroSort(int* data, int begin , int end ) {
        int depthLimit;

        depthLimit = 2 * floor(log(end-begin)/log(2));



        IntrosortUtil(data, begin, end, depthLimit);

        }




void IntrosortUtil(int* data, int begin, int end, int depthLimit ){
        int size , temp,i;
        int p , pivot;
        size = end - begin;


        if (size < 16)
        {
        //      printf("Insertion Sort\t");
                InsertionSort(data, begin , end );
        }
        else if (depthLimit == 0)
        {
        //      printf("Heap Sort\t");
                HeapSort(data, begin, end );
```

```
        }
        else
        {
        //      printf("Quick Sort\t");
                depthLimit = depthLimit - 1;

                /*pivot = findPivot(data , begin , begin + (size / 2)+1  , end);

                swap(data , pivot , end);*/

                p = Bolme(data, begin, end );                   //Kısmi sıralama
                IntrosortUtil(data, begin, p-1, depthLimit);
                IntrosortUtil(data, p+1, end, depthLimit );
        }


}
```

Video Linki : https://www.youtube.com/watch?v=d_-DJwtCAKw

KAYNAKLAR ;

https://web.archive.org/
https://www.geeksforgeeks.org/
https://www.programmingalgorithms.com/
https://algorithmist.com/