**Ad soyad : Yasin KARAÇAM**

**Algoritmalar – I Ödev 1**

**Quick Sort ve Merge Sort**

**Karşılaştırma**

**Soru :**

**https://www.naukri.com/code360/library/quick-sort-vs-merge-sort**

**Bu linkte verilen Quick Sort ve Merge Sort kodlarını kullanabilirsiniz. Sizden istenen öncelikle dizileri 1**

**ila 100 arasında sırasız 100 eleman içerecek şekilde rastgele oluşturmak, daha sonra bunları en kötü**

**durum için karşılaştıracak şekilde algoritmayı düzenlemek ve iki algoritmayı karşılaştırmaktır. Bunu**

**gerçekleştirmek için C kütüphanesinden clock() fonksiyonu kullanarak programın başlatılmasından bu**

**yana program tarafından kullanılan saat tiklerinin sayısı olan, program tarafından tüketilen yaklaşık**

**işlemci süresini döndüren clock() fonksiyonu yardımıyla bu süreyi belirleyebilirsiniz. Çalışan C kodunu**

**ister siz yazabilir, isterse beğendiğiniz bir tanesini bulup kullanabilirsiniz. Örnek kod için mesela**

**yukardaki linki kullanabilir siniz.**

**Sonuçları ekran çıktısı ve kullandığınız kod dahil rapor kağıdına düzenli bir şekilde geçirip**

**yorumlayarak sunmanız, ayrıca kodu e-posta yoluyla .c ve de .exe dosyası olarak bana teslim etmeniz**

**de gerekmektedir.**

**Çözüm**

**C Kodu :**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 100

void swap(int\* a, int\* b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return i + 1;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (int i = 0; i < n1; i++) L[i] = arr[l + i];

for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

int i = 0, j = 0, k = l;

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

if (L[i] <= R[j]) arr[k++] = L[i++];

else arr[k++] = R[j++];

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void generateRandomArray(int arr[]) {

srand(time(0));

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

arr[i] = rand() % 100 + 1;

}

}

void printArray(int arr[]) {

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

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int arr1[SIZE], arr2[SIZE];

generateRandomArray(arr1);

for (int i = 0; i < SIZE; i++) arr2[i] = arr1[i];

printf("Original array:\n");

printArray(arr1);

clock\_t start, end;

// QuickSort

start = clock();

quickSort(arr1, 0, SIZE - 1);

end = clock();

printf("\nQuickSort sorted array:\n");

printArray(arr1);

printf("\nQuickSort execution time: %.6f seconds\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

// MergeSort

start = clock();

mergeSort(arr2, 0, SIZE - 1);

end = clock();

printf("\nMergeSort sorted array:\n");

printArray(arr2);

printf("\nMergeSort execution time: %.6f seconds\n", ((double)(end - start)) / CLOCKS\_PER\_SEC);

return 0;

}

**Program Çıktısı:**

**Original array:**

39 61 39 51 77 36 62 65 85 32 75 41 16 6 72 6 53 56 45 61 93 58 52 88 76 11 67 94 88 23 100 79 83 91 29 12 26 91 28 10 74 2 2 89 7 74 47 11 81 91 23 73 100 74 60 27 84 78 73 24 100 72 54 34 62 82 97 39 24 24 48 97 26 2 86 84 75 32 47 55 74 69 79 25 43 90 51 78 67 23 53 18 47 6 3 60 88 100 99 11

**QuickSort sorted array:**

2 2 2 3 6 6 6 7 10 11 11 11 12 16 18 23 23 23 24 24 24 25 26 26 27 28 29 32 32 34 36 39 39 39 41 43 45 47 47 47 48 51 51 52 53 53 54 55 56 58 60 60 61 61 62 62 65 67 67 69 72 72 73 73 74 74 74 74 75 75 76 77 78 78 79 79 81 82 83 84 84 85 86 88 88 88 89 90 91 91 91 93 94 97 97 99 100 100 100 100

**QuickSort execution time:** 0.000007 seconds

**MergeSort sorted array:**

2 2 2 3 6 6 6 7 10 11 11 11 12 16 18 23 23 23 24 24 24 25 26 26 27 28 29 32 32 34 36 39 39 39 41 43 45 47 47 47 48 51 51 52 53 53 54 55 56 58 60 60 61 61 62 62 65 67 67 69 72 72 73 73 74 74 74 74 75 75 76 77 78 78 79 79 81 82 83 84 84 85 86 88 88 88 89 90 91 91 91 93 94 97 97 99 100 100 100 100

**MergeSort execution time:** 0.000011 seconds

**Analiz:**

1. **Quick Sort ve Merge Sort Algoritmaları:**
2. **Quick Sort** genellikle daha hızlı bir algoritma olup, ortalama O(n log n) zaman karmaşıklığına sahiptir. Fakat, en kötü durum senaryosunda (dizi zaten sıralı veya ters sıralıysa) O(n^2) zaman alabilir.
3. **Merge Sort** her durumda O(n log n) zaman karmaşıklığına sahiptir ve daha kararlı bir sıralama algoritmasıdır. Ancak daha fazla bellek kullanımı gerektirir
4. **Çıktı Yorumları:**
5. Rastgele oluşturulan dizilerin her iki sıralama algoritması tarafından başarıyla sıralandığı gözlemlenmiştir. Her iki algoritma da sıralama işlemi sırasında aynı sonucu üretmiştir.
6. Çalışma sürelerinin çok küçük olduğu ve sıralama işlemlerinin çok hızlı gerçekleştiği gözlemlenmiştir. Bu küçük dizilerde, algoritmaların zaman farkı genellikle gözle görülmez, ancak büyük veri setlerinde fark daha belirgin hale gelir

**Not :** Ayrıca yapılan incelemeler sonucunda programdaki çıktı olan süreler Windows’ta 0,00...00 olarak karşıma çıktı ,Linux’taki süreler ise belgedeki Outputlar gibidir(Süreler sisteminize göre değişiklik gösterebilir.)