ALGORİTMA ANALİZİ

ÖDEV 1 | SORU 1

GRUP 1

İrem ATILGAN

17061036

07.11.2020

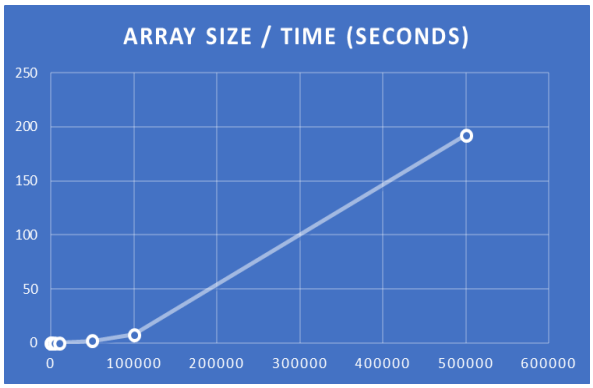
SORU 1

N elemanlı bir dizide birbirine en yakın değere sahip iki elemanın bulunması isteniyor.

A) Brute Force Yaklaşımı : Brute Force Yaklaşımı'nda dizide bulunan her bir elemanın, dizideki diğer elemanlarla farkı bulunmuş ve minimum fark ile karşılaştırılmıştır. Dizide N eleman olduğundan ve her eleman için N elemanla olan farkına (elemanın kendi kendisinin farkı alınmasa da tüm hücreler geziliyor) bakıldığından karmaşıklığı :

Best Case : $O(N^2)$
Average Case : $\theta(N^2)$
Worst Case : $\Omega(N^2)$

Array Size	Time (Seconds)
10	0
100	0
1000	0.001
5000	0.027
10000	0.085
50000	1.982
100000	7.858
500000	192.19



PROGRAM KODLARI

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

int* set_array(int*, int);          //Set array with random numbers
void quick_sort(int*, int, int);    //Quick sort algorithm
void swap(int*,int*);               //Swap two numbers
int* alt_way(int*, int);            //Alternative way for finding two elements that gives minimum difference in an array
int* linear_search(int*, int);      //Search whole array to find minimum difference between sequential numbers

int main()
{
    int length;     //array length
    int* array;
    int* indices;   //indices found for minimum difference

    printf("\nPlease enter the length of the array : ");scanf("%d",&length);

    array = (int*)malloc(sizeof(int)*length);

    if(length > 1)
    {
        array = set_array(array,length);
        indices = alt_way(array,length);
        printf("MIN DIF = %d FOUND BETWEEN NUMBERS [%d] & [%d]",abs(array[indices[0]]-array[indices[1]]),array[indices[0]],array[indices[1]]);
        //free pointers from memory
        free(indices);
        free(array);

    }
    else
        printf("The array size should be more than 1");

    return 0;
}
```

```c
int* set_array(int* arr, int length)
{
    int i;
    srand(time(0)); //Use current time as seed for random generator

    for(i = 0; i < length ; i++)
        arr[i] = rand();

    return arr;
}

void print_array(int* arr, int length)
{
    int i;
    for(i = 0; i < length; i++) {
        printf("%d\t",arr[i]);
    }
    printf("\n");
}
```

```c
int* brute_force(int* arr, int length)
{
    clock_t begin = clock();    //the time algorithm had started working
    int i,j;
    int dif;                    //difference between two numbers
    int min_dif;                //minimum difference found
    int* indices = (int*)malloc(sizeof(int)*2); //indices found for minimum difference

    //set the minimum difference with the difference of first two elements
    indices[0] = 0;
    indices[1] = 1;
    min_dif = abs(arr[0] - arr[1]);
    for(i = 0; i < length-1; i++)
    {
        for(j = 1; j < length; j++)
        {
            if(i != j) //cannot be the difference of same number
            {
                dif = abs(arr[i] - arr[j]);
                if(min_dif > dif)
                {
                    min_dif = dif;
                    indices[0] = i;
                    indices[1] = j;
                }
            }
        }
    }
    clock_t end = clock();      //the time brute force algorithm has finished
    double time_spent = (double)(end-begin) / CLOCKS_PER_SEC;   //calculate the time passed since the algorithm had started
    printf("ALGORITHM PROGRESSING TIME = %lf\n",time_spent);
    return indices;
}
```

EKRAN ÇIKTILARI

```
Please enter the length of the array : 10000
ALGORITHM PROGRESSING TIME = 0.102000
MIN DIF = 0 FOUND BETWEEN INDICES 7 [19881] & 4048 [19881]
--------------------------------
Process exited after 3.537 seconds with return value 0
Press any key to continue . . .
```

```
Please enter the length of the array : 100000
ALGORITHM PROGRESSING TIME = 9.251000
MIN DIF = 0 FOUND BETWEEN INDICES 0 [3008] & 22554 [3008]
--------------------------------
Process exited after 15.05 seconds with return value 0
Press any key to continue . . .
```

B) Alternatif Yaklaşım : Alternatif Yaklaşım'da ilk olarak mevcut dizi Quick Sort sıralama algoritması ile artacak şekilde sıralanır. Daha sonra sıralanan dizideki tüm elemanlar gezilerek kendilerinden bir önceki sayı ile aralarındaki fark bulunur ve minimum fark ile karşılaştırılır (Linear Search). İki aşamanın karmaşıklıkları ele alınırsa:

Quick Sort

Best Case : $O(N * log_2 N)$
Average Case : $\theta(1.39 * N * log_2 N)$
Worst Case : $\Omega(N^2)$

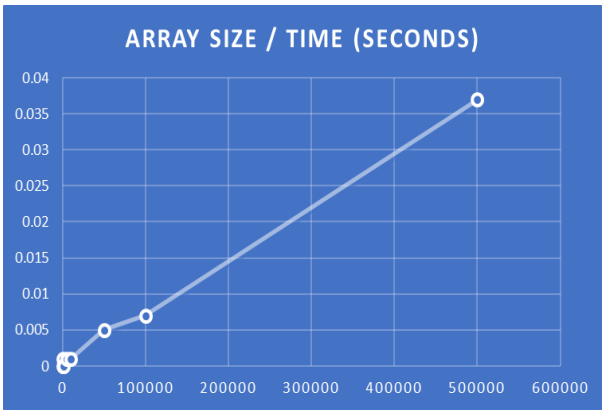
Linear Search

Best Case : $O(N)$
Average Case : $\theta(N)$
Worst Case : $\Omega(N)$

Sonuç :

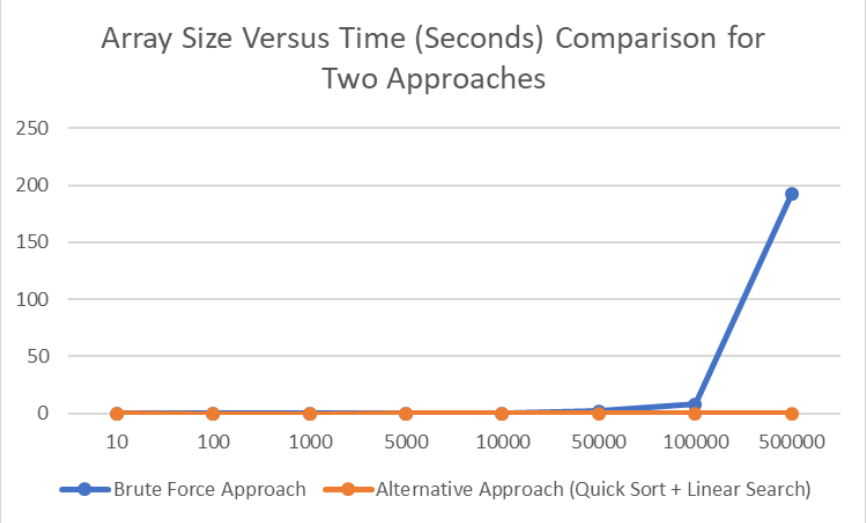
Best Case : $O(N * log_2 N)$
Average Case : $\theta(1.39 * N * log_2 N)$
Worst Case : $\Omega(N^2)$

Array Size	Time (seconds)
10	0.001
100	0
1000	0
5000	0.001
10000	0.001
50000	0.005
100000	0.007
500000	0.037



SONUÇ

İki yaklaşımın da dizi boyutuna bağlı olarak çalışma süreleri aşağıdaki grafikte karşılaştırmalı olarak verilmiştir. Görüldüğü üzere ikinci yaklaşım, karmaşıklığı Brute-Force yaklaşımına göre çok daha düşük olduğu için, en yüksek başarıyı göstermektedir.



PROGRAM KODLARI

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

int* set_array(int*, int);          //Set array with random numbers
void quick_sort(int*, int, int);    //Quick sort algorithm
void swap(int*,int*);               //Swap two numbers
int* alt_way(int*, int);            //Alternative way for finding two elements that gives minimum difference in an array
int* linear_search(int*, int);      //Search whole array to find minimum difference between sequential numbers

int main()
{
    int length;     //array length
    int* array;
    int* indices;   //indices found for minimum difference

    printf("\nPlease enter the length of the array : ");scanf("%d",&length);

    array = (int*)malloc(sizeof(int)*length);

    if(length > 1)
    {
        array = set_array(array,length);
        indices = alt_way(array,length);
        printf("MIN DIF = %d FOUND BETWEEN NUMBERS [%d] & [%d]",abs(array[indices[0]]-array[indices[1]]),array[indices[0]],array[indices[1]]);
        //free pointers from memory
        free(indices);
        free(array);

    }
    else
        printf("The array size should be more than 1");

    return 0;
}
```

```c
int* set_array(int* arr, int length) {
    int i;
    srand(time(0)); //Use current time as seed for random generator
    for(i = 0; i < length ; i++)
        arr[i] = rand();

    return arr;
}

//Alternative way for finding two elements that give minimum difference in an array
int* alt_way(int* arr, int length)
{
    clock_t begin = clock();    //the time algorithm had started working
    quick_sort(arr,0,length-1); //sort the array (in ascending order)
    printf("SORTED!..\n");
    int* indices = linear_search(arr,length);   //find indices that give minimum difference by calculating difference of two sequential elements
    printf("COMPLETED!..\n");
    clock_t end = clock();                      //the time brute force algorithm has finished
    double time_spent = (double)(end-begin) / CLOCKS_PER_SEC; //calculate the time passed since the algorithm had started
    printf("ALGORITHM PROGRESSING TIME = %lf\n",time_spent);

    return indices;
}
```

```c
//Sort array with Quick Sort
void quick_sort(int* arr, int left, int right) {
    if(left < right)
    {
        int i = left;
        int j = right;
        int pivot = left;
        while(i < j)
        {
            while(arr[pivot] >= arr[i] && i < right)    i++;
            while(arr[pivot] < arr[j])  j--;
            if(i < j) {
                swap(&arr[i],&arr[j]); //swap the numbers which one of them is larger and one of them is smaller than pivot
            }

        }
        swap(&arr[pivot],&arr[j]);  //set pivot to its new place

        //divide array into two parts and keep sorting
        quick_sort(arr,left,j-1);
        quick_sort(arr,j+1,right);
    }
}
```

```c
//Search whole array to find the indices that give minimum difference between two sequential elements
int* linear_search(int* arr, int length) {
    int i;
    int* indices = (int*)malloc(sizeof(int)*2); //indices found for minimum difference

    int dif;                                    //difference between two numbers
    int min_dif;                                //minimum difference found

    //set the minimum difference with the difference of first two elements
    indices[0] = 0;
    indices[1] = 1;
    min_dif = abs(arr[0]-arr[1]);

    for(i = 2; i < length; i++)
    {
        dif = abs(arr[i]-arr[i-1]); //calculate the difference between two sequential numbers
        if(dif < min_dif)
        {
            min_dif = dif;
            indices[0] = i;
            indices[1] = i-1;
        }
    }

    return indices;
}
```

EKRAN ÇIKTILARI

```
Please enter the length of the array : 100
SORTED!..
COMPLETED!..
ALGORITHM PROGRESSING TIME = 0.001000
MIN DIF = 2 FOUND BETWEEN NUMBERS [17798] & [17796]
--------------------------------
Process exited after 2.297 seconds with return value 0
Press any key to continue . . .
```

```
Please enter the length of the array : 1000
SORTED!..
COMPLETED!..
ALGORITHM PROGRESSING TIME = 0.000000
MIN DIF = 0 FOUND BETWEEN NUMBERS [880] & [880]
--------------------------------
Process exited after 2.847 seconds with return value 0
Press any key to continue . . .
```

```
Please enter the length of the array : 10000
SORTED!..
COMPLETED!..
ALGORITHM PROGRESSING TIME = 0.001000
MIN DIF = 0 FOUND BETWEEN NUMBERS [30] & [30]
--------------------------------
Process exited after 1.941 seconds with return value 0
Press any key to continue . . .
```

```
Please enter the length of the array : 50
SORTED!..
COMPLETED!..
ALGORITHM PROGRESSING TIME = 0.002000
MIN DIF = 8 FOUND BETWEEN NUMBERS [29351] & [29343]
--------------------------------
Process exited after 3.707 seconds with return value 0
Press any key to continue . . .
```