ALGORİTMA ANALİZİ

PROF.DR.MİNE ELİF KARSLIGİL

ÖDEV 2

6 Kasım 2022

20011623
Asude Merve EKİZ

Yöntem:

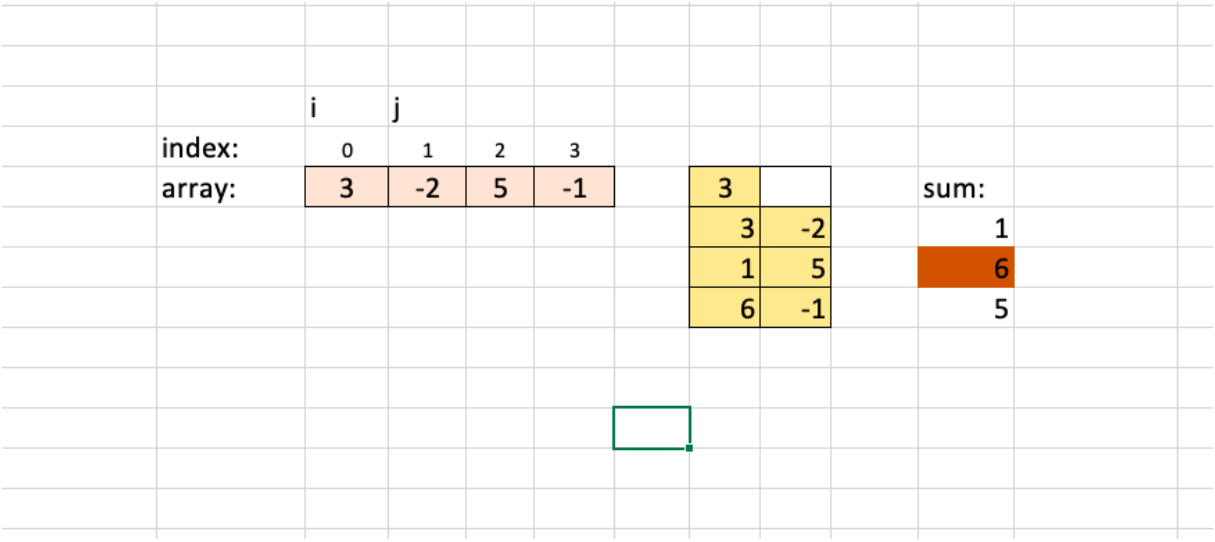
Verilen ödevde bir dizideki aralıksız en yüksek toplamı veren aralık ve toplam değer istenmektedir. Bu problemin aşağıda belirtildiği üzere iki tane çözümü bulunmaktadır.

1) Brute Force yaklaşımı
2) Divide and Conquer yaklaşımı

Brute Force yaklaşımındaki çözümü inceleyerek başlayalım:

```
for ( i = 0; i < n; i++) {
    int sum = array[i];
    for (j = i+1; j < n; j++) {
        sum += array[j];

        if (sum > maximumSubArraySum) {
            maximumSubArraySum = sum;
            start = i;
            end = j;
        }
    }
}
```

Çözümün bizim için önemli kod parçacığına baktığımızda iki döngü görmekteyiz.
i ilk eleman; j , i'nin hemen yanındaki eleman olmak üzere iç döngüye baktığımızda her seferinde array[j] üzerine eklenerek gidiliyor. Eğer sum değeri, başta oldukça düşük bir değer verdiğimiz maximumSubArraySum değerinden yüksek ise güncelleme yapılıyor.



Ancak fonksiyonda da gördüğümüz üzere bunu i nin tüm elemanları için yaptığı için biraz daha verimsiz bir algoritma.

Genel zaman karmaşıklığı : $O(n^2)$

(Analiz kısmına sonda sözde kod kısmında tekrar değinilecektir.)

2)Divide and Conquer yaklaşımı

Bu yaklaşımda ise böl ve yönet mantığı kullanılıyor ve en yüksek toplamı veren alt dizinin olabileceği 3 yer kontrol ediliyor:

Aradığımız alt dizi

I. Tamamen sol tarafta olabilir.
II. Tamamen sağ tarafta olabilir.
III. Bir kısmı solda bir kısmı sağda olacak şekilde ortada bir konumda olabilir.

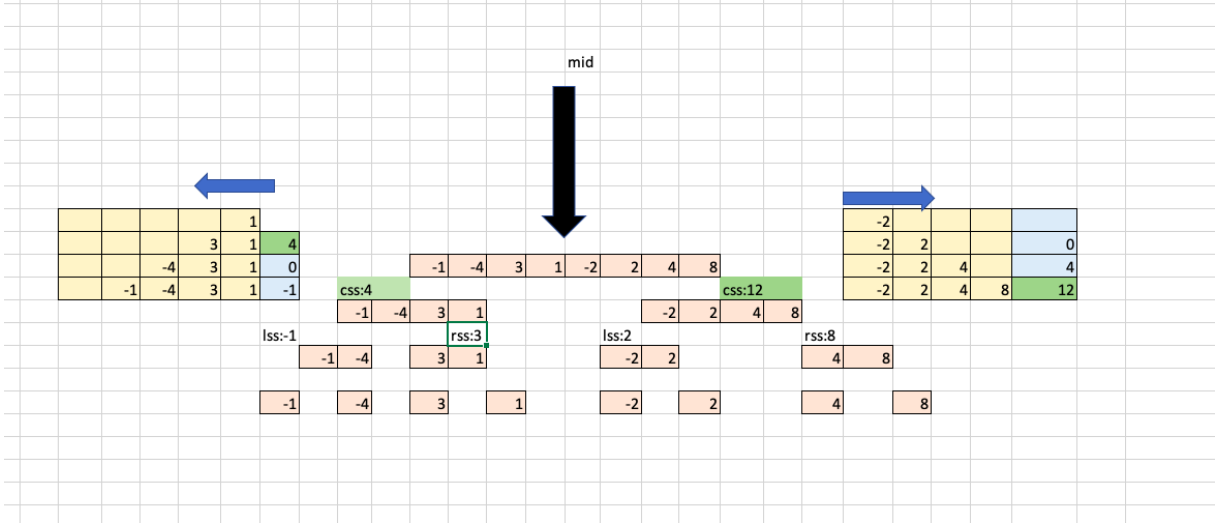
İşleyiş ise şu şekilde:

Dizi önce mümkün olduğunca küçük olana kadar özyinelemeli olarak ikiye bölünüyor. Sonrasında aşağıdan yukarıya doğru çıkarak sol max toplam, sağ max toplam bulunuyor ancak bu bizim için yeterli değil. Burada devreye çapraz toplam dediğimiz bir kavram giriyor.

```
int maxCrossingSubarray(int ar[], int low, int mid, int
high){

 int left_sum = -255;
 int sum = 0;
 int i;

 for (i=mid; i>=low; i--){
  sum = sum+ar[i];
   if (sum>left_sum){
      left_sum = sum;
      startIndex=i;
   }
 }
```

```
 int right_sum = -255;
 sum = 0;

 for (i=mid+1; i<=high; i++)
 {
  sum=sum+ar[i];
   if (sum>right_sum){
      right_sum = sum;
      endIndex=i;
   }
 }

 return (left_sum+right_sum);}
```

Aşağıdaki görsel üzerinde belirtilen işlemler yapıldıktan sonra cross Sum da bulunmuş olunuyor. (Bu kısım



yazarak anlatmak zor olduğundan videoda detaylandırıldı.)

```
int * maxSumSubArray(int ar[], int low, int high)
{
  if (high == low) // only one element in an array
  {
     startIndex=low;
     endIndex=high;
     return ar[high];
  }

  int mid = (high+low)/2;

  int maximumSumLeftSubArray = maxSumSubArray(ar, low, mid);  // maximum sum in the left subarray
  int maximumSumRightSubArray = maxSumSubArray(ar, mid+1, high);  // maximum sum in the right subarray
  int maximumSumCrossingSubArray = maxCrossingSubarray(ar, low, mid, high); // maximum sum in the array
include the middle element

   int maxValue= maximum(maximumSumLeftSubArray, maximumSumRightSubArray,
maximumSumCrossingSubArray);


    if(maxValue==maximumSumLeftSubArray){
       return maxSumSubArray(ar, low, mid);
    }
    else if(maxValue==maximumSumRightSubArray){
       return maxSumSubArray(ar, mid+1, high);
    }
    else{
       return maxCrossingSubarray(ar, low, mid, high);
    }

            }
```

Sonrasında da left right ve cross max sumlardan en yüksek olan bulunarak max value değişkenine atılıyor. Max value' nun değerine göre de hangi alt arrayin döndürüleceği belirleniyor (çünkü fonksiyon int * olarak tanımlandı.)

Karmaşıklığa baktığımızda ise öncelikle bağıntıyı yazıyoruz:

$$T(n) = O(1) \text{ if } n = 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \text{ if } n > 1$$

ardından master teoremi uyguladığımızda da aşağıdaki sonucu elde ediyoruz.

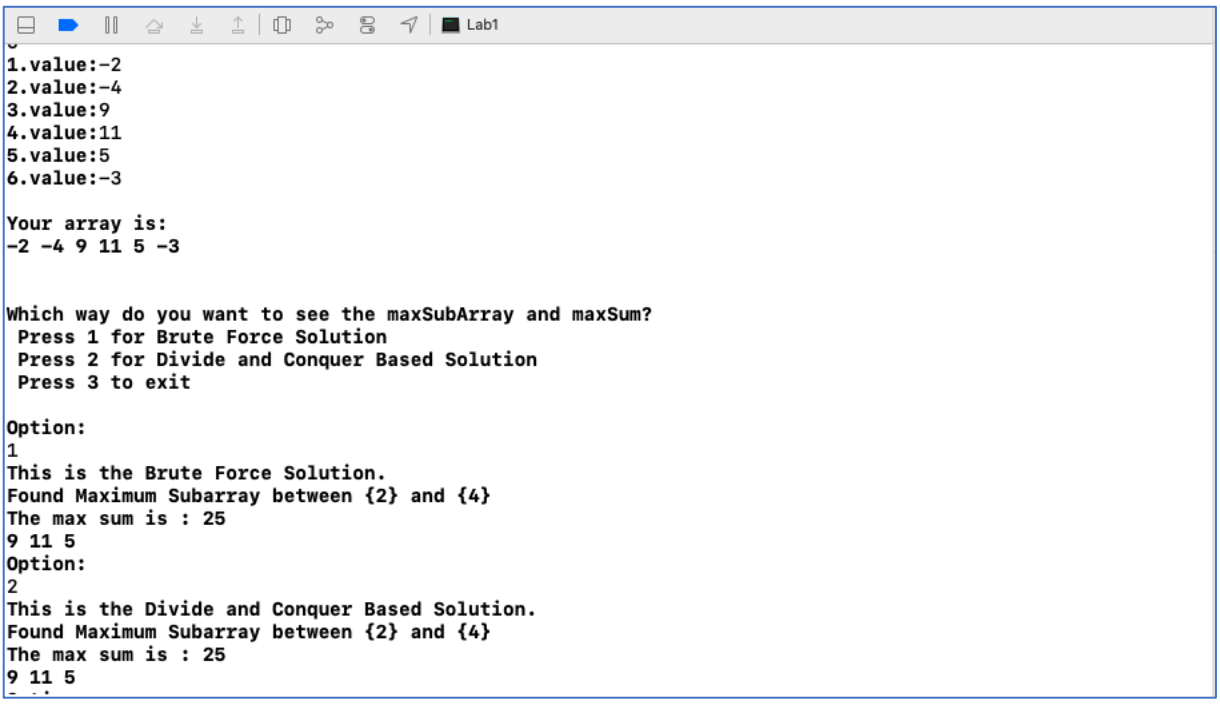
a=2 b =2 d=1 olduğundan master teoremine göre zaman karmaşıklığı O(n*log(n)) olarak bulunur.

Uygulama

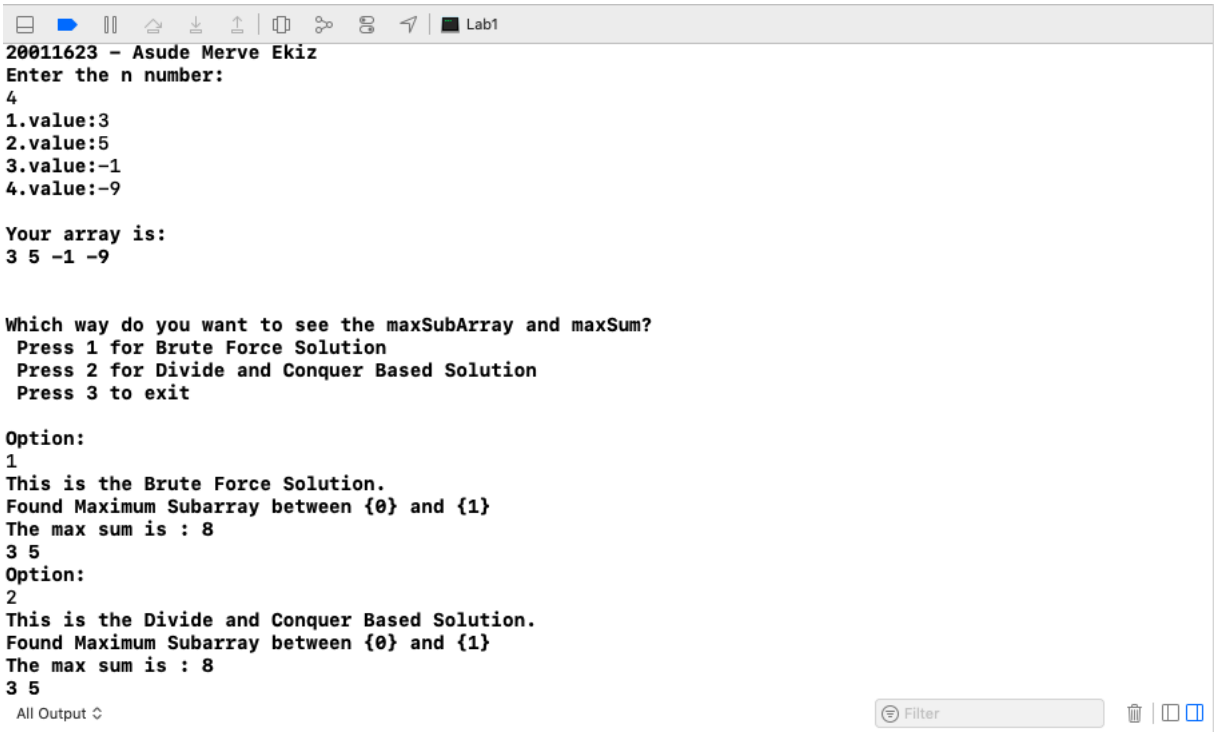
Bu kısımda çeşitli case'ler ile program denenmiştir. Ekran görüntüleri aşağıdadır:

```
1.value:8
2.value:-30
3.value:36
4.value:2
5.value:-6
6.value:52
7.value:8
8.value:-1
9.value:-11
10.value:10
11.value:4

Your array is:
8 -30 36 2 -6 52 8 -1 -11 10 4


Which way do you want to see the maxSubArray and maxSum?
 Press 1 for Brute Force Solution
 Press 2 for Divide and Conquer Based Solution
 Press 3 to exit

Option:
1
This is the Brute Force Solution.
Found Maximum Subarray between {2} and {10}
The max sum is : 94
36 2 -6 52 8 -1 -11 10 4
Option:
2
This is the Divide and Conquer Based Solution.
Found Maximum Subarray between {2} and {10}
The max sum is : 94
36 2 -6 52 8 -1 -11 10 4
Option:
0
Program ended with exit code: 0
```

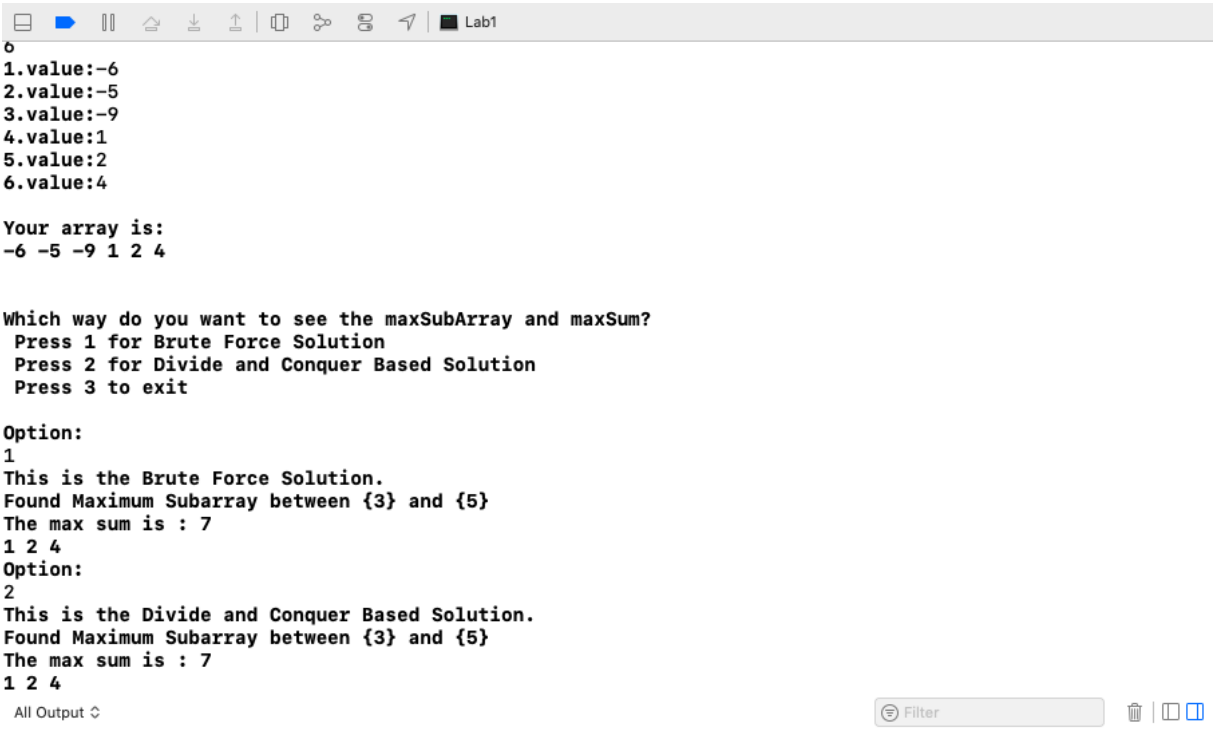
Çıktı 1: Soruda verilen örnek



Çıktı 2: 6 elemanlı bir dizi, max toplamı veren sub arrayin bir kısmı solda bir kısmı sağda.



Çıktı 3: 4 elemanlı dizi, max sum sub array sol taraftaki toplamdan geliyor.



Çıktı 4: 6 elemanlı dizi, max sum sub array sağ taraftaki toplamdan geliyor.

Analiz

1)Brute Force Yaklaşımı

```
maxSubArrayByBruteForce(n,array){
        maxSubArraySum ←-255                          | 1 atama işlemi
        for i← 0 to n do {                            | 1 atama işlemi  n +1 karşılaştırma
            sum ← array[i]                            | N tane atama işlemi
            for j← i+1 to n do{                       | 1 toplama 1 atama n ~karşılaştırma
                sum = sum + array[j]                  | N *N tane:
                                                      | Toplama ve atama işlemi
                //conditional part
            }
        }
    }
```

Psödö kod üzerinde yaptığımız analiz sonucunda düşük değerli kısımları da attığımızda iç içe iki döngü olmasından ötürü

Genel zaman karmaşıklığı : $O(n^2)$ olarak bulunmaktadır.

2)Divide and Conquer Yaklaşımı

```
maxSumSubArray(array, low,high){
        if (high == low){
            startIndex←low
            endIndex←high
        }
        Return array[high]

  mid = (low +high) DIV 2
  maximumSumLeftSubArray← maxSumSubArray(array, low, mid)
  maximumSumRightSubArray ← maxSumSubArray(array, mid+1, high)
  maximumSumCrossingSubArray ← maxCrossingSubarray(array, low, mid, high)
  maxValue← maximum (maximumSumLeftSubArray, maximumSumRightSubArray,
maximumSumCrossingSubArray)

}
```

Maksimum geçiş alt dizisini bulmak için O(n) zaman gerekiyor.
İki alt dizi olduğundan bu kısım için toplam zaman karmaşıklığı 2T(n/2)
Bu belirtilenlerle rekürans bağıntısını yazalım

$$T(n) = O(1) \text{ if } n = 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) \text{ if } n > 1$$

Master teoremini uyguladığımızda da:
a =2
b=2
d=1 ve 2 = 2 üssü 1 olduğundan karmaşıklık O (n*log(n)) olarak bulunmaktadır.

Video anlatım linki ektedir:
https://youtu.be/iimZB7jQQcU