Garrett Pement, Lance Umagat

2/24/2016

Analysis of Algorithms

Intro

Here, we will be going over the pseudocode for Method 1, 2, and 3 of the prefix-suffix method, as well as the pseudocode for the Divide and Conquer method. The goal of these algorithms is to find a pair of elements in an array that, when added, give a sum closest to 0.

Note Timings for sort we used in in the code

Method 1

```
Method1(array sub_a[], array sub_b[])
Sum = 1000000
For(I = 0; I = sizeof(sub_a[]); I++)
        Minsum = 0
        For(J= 0; J = sizeof(sub_b[]); J++)
                {
                        Minsum = (a[i] + b[j]);
                        If abs(minsum) < sum
                                Sum = minsum
                }
```

T(n) = 2T(n/2) +cn^2

T(n) = Θ (n^2)

Runtime = Θ (n^2)

Method 2

```
Method2(arr):
        Min_left = 0
        Min_right = 1
        Mid = (len(arr) -1) //2
        Min_sum = int(arr[0] + int(arr[1])
        For I in range(0, mind +1):
                Sum = 0
                For j in range(mid + 1, len(arr)):
                        Sum = int(arr[i]) + int(arr[j])
                        If(abs(min_sum) > abs (sum) ):
                                min_sum = sum
                                min_l = i
                                min_r = j
        CtZ = int(arr[min+l] + int(arr[min_r])
        Return(int(min_l), int(min_r))
```

$$T(n) = \begin{cases} O(1) & n <= 1 \\ T(n/2) + O(n) & n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 4T(n/4) + 2n \\ &= 8T(n/8) + 3n \\ &\dots \\ &= 2^kT(n/2^k) + kn \end{aligned}$$

$T(n) = n + n\log n = \Theta(n\log n)$

Method 3

```
Method3(arr):
    Arr[] = binarysort(arr[]); //this sorts the array from lowest value to highest value
    Min_left = 0
    Min_right = 1
    Mid = (len(arr) -1)//2
    Min_sum = int(arr[0] + int(arr[1])
    For I in range(0, mind +1):
        Sum = 0
        For j in range(mid + 1, len(arr)):
            Sum = int(arr[i]) + int(arr[j])
            If(abs(min_sum) > abs (sum) ):
                min_sum = sum
                min_l = i
                min_r = j
    CtZ = int(arr[min+l] + int(arr[min_r])
    Return(int(min_l), int(min_r))
```

$$T(n) = \begin{cases} O(1) & n <= 1 \\ T(n/2) + O(n) & n > 1 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 4T(n/4) + 2n \\ &= 8T(n/8) + 3n \end{aligned}$$

…

= 2^kT(n/2^k) + kn

T(n) = n + nlogn = Θ (nlogn)

Divide and Conquer

(Using Method 2)

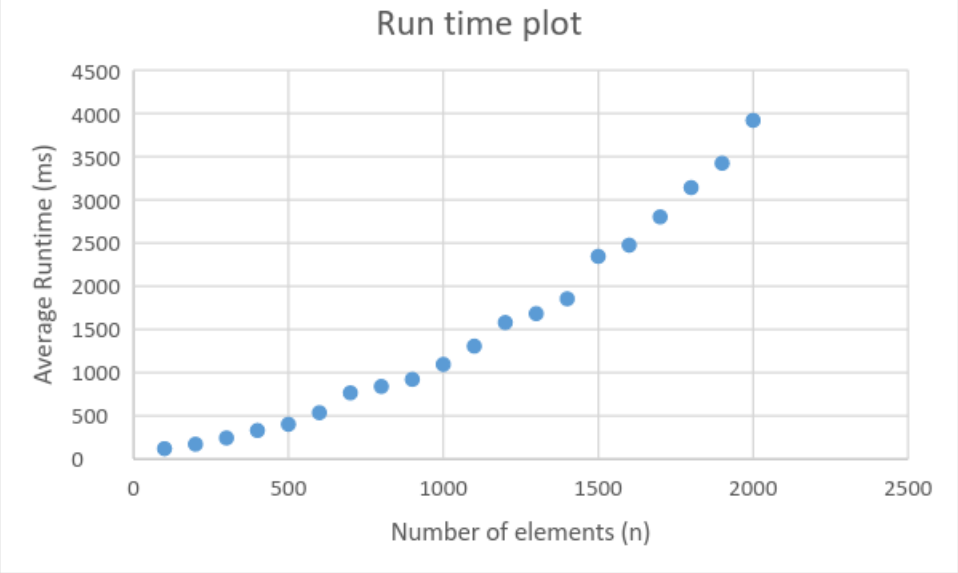
```
Function(arr, size):
    Arr1_size  = size // 2
    Arr1 = []
    For I in range (0 to arr1_size)
        Arr1.append(arr[i])

    Arr2_size = size // 2
    Arr2 = []
    For I in range (arr2_size, size):
        Arr2.append(arr[i])
    Return Check(enumerate(arr1,arr1_size,0), enumerate(arr2,arr2_size, arr1_size),
enumerate2(arr1, arr2, arr1_size, arr2_size));
```

//enumerate is a helper function that enumerates an array, with the size being the second value and the index being the third.

//enumerate2 is a helper function that enumerates the entirety of two arrays together

Time complexity = *O(nlogn)* .



# of elements	Average Runtime (ms)
100	117.77
200	169.0768
300	241.42
400	327.95
500	399.67
600	533.96
700	763.28
800	837.77
900	919.79
1000	1093.4
1100	1304.74

1200	1577.99
1300	1681.41
1400	1852.91
1500	2344.18
1600	2473.99
1700	2801.48
1800	3140.61
1900	3423.12
2000	3919.23