Algorithms Tasks Spring 2024

Task name: Max Product Of Three

Team Leader Name: احمد محمد محمود ابراهيم ابوطالب

Team Leader ID: 20220040

First method:

We can use Brute Force technique to solve this problem

Pseudocode:

```
function maxProductOfThree(arr[], n):
   if n < 3:
      return 0
   maxProduct = INT_MIN
   for i from 0 to n-1:
      for j from 0 to n-1:
         for k from 0 to n-1:
            if i == j or j == k or i == k:
               continue
            product = arr[i] * arr[j] * arr[k]
            if maxProduct < product:
               maxProduct = product
   return maxProduct
```

Code in C:

Sorting.c

```c
int maxProductOfThree(int arr[], int n) {
    if(n < 3)
        return 0;
    int maxProduct = INT_MIN;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            for(int k = 0; k < n; k++) {
                if(i == j || j == k || i == k)
                    continue;
                if(maxProduct < arr[i]*arr[j]*arr[k])
                    maxProduct = arr[i]*arr[j]*arr[k];
            }
        }
    }
    return maxProduct;
}
```

Analysis of the code:

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3$$

T(n) = Θ(n^3)

Space complexity = O(1)

Second method:

We can easily observe that if numbers are sorted then the maximum product would be the product of the biggest 3 positive numbers or the product of the smallest 2 negative numbers and the biggest positive.

Pseudocode:

```
function maxProductOfThree(arr[], n):
    if n < 3:
        return 0
    mergeSort(arr, 0, n-1)

    maxProduct = 0
    max1 = arr[n-1]
    max2 = arr[n-2]
    max3 = arr[n-3]
    min1 = arr[0]
    min2 = arr[1]

    if max1 * max2 * max3 > min2 * min1 * max1:
        maxProduct = max1 * max2 * max3
    else:
        maxProduct = min2 * min1 * max1

    return maxProduct
```

Code in C:

Sorting.c

```c
int maxProductOfThree(int arr[], int n) {
    if(n < 3)
        return 0;
    mergeSort(arr, 0, n-1);
    int maxProduct = INT_MIN, max1 = arr[n-1], max2 = arr[n-2], max3 = arr[n-3], min1 = arr[0], min2 = arr[1];
    if(max1 * max2 * max3 > min2 * min1 * max1)
        maxProduct = max1 * max2 * max3;
    else
        maxProduct = min2 * min1 * max1;
    return maxProduct;
}
```

Analysis of the code:

In out code we used Merge Sort Algorithm,

T(n) = 2T(n/2) + O(n)

A = 2, b = 2 and F(n) = n

$n^{log_b a} = n^{log_2 2} = n = f(n)$ Case 2

T(n) = Θ($n^{log_b(a) \log n}$) = Θ(n log n)

Space complexity = O(n)

Third method:

We can get the biggest 3 positive numbers and the smallest 2 negative numbers in linear time using recursion.

Pseudocode:

```
int maxProductOfThree(int arr[], int n, int i, int max1, int
max2, int max3, int min1, int min2) {
   if(n < 3)
      return 0;
   if(i == n) {
      if(max1 * max2 * max3 > min2 * min1 * max1)
         return max1 * max2 * max3;
      else
         return min2 * min1 * max1;
   }
   if(arr[i] > max1) {
      max3 = max2;
      max2 = max1;
      max1 = arr[i];
   } else if(arr[i] > max2) {
      max3 = max2;
      max2 = arr[i];
   } else if(arr[i] > max3) {
      max3 = arr[i];
   }

   if(arr[i] < min1) {
      min2 = min1;
      min1 = arr[i];
   } else if(arr[i] < min2) {
      min2 = arr[i];
   }
```

```
    return maxProductOfThree(arr, n, i+1,
max1,max2,max3,min1,min2);
}
```

Code in C:

recursiveLinear.c

```c
int maxProductOfThree(int arr[], int n, int i, int max1, int max2, int max3, int min1, int min2) {
    if(n < 3)
        return 0;
    if(i == n) {
        if(max1 * max2 * max3 > min2 * min1 * max1)
            return max1 * max2 * max3;
        else
            return min2 * min1 * max1;
    }
    if(arr[i] > max1) {
        max3 = max2;
        max2 = max1;
        max1 = arr[i];
    } else if(arr[i] > max2) {
        max3 = max2;
        max2 = arr[i];
    } else if(arr[i] > max3) {
        max3 = arr[i];
    }

    if(arr[i] < min1) {
        min2 = min1;
        min1 = arr[i];
    } else if(arr[i] < min2) {
        min1 = arr[i];
    }
    return maxProductOfThree(arr, n, i+1, max1,max2,max3,min1,min2);
}
```

Analysis of the code:

Base case: $T(0) = 1$

$T(n) = T(n-1) + 1$

Expand the recurrence.

$T(n) = [T(n-2) + 1]+1 = T(n-2)+2$

$= T(n-k) + k$

$n-k = 0$

$k = n$

$T(n) = T(0) + (n) = 1 + n$

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

Space complexity = $O(n)$

Comparison between algorithms:

We can see that we solved out problem with many ways but the main difference between our algorithms the factor of time complexity

Method Number	Time Complexity
1	Cubic
2	Linearithmic
3	Linear

Team:

NAME	ID
AHMED MOHAMED MAHMOUD IBRAHIM	20220040
ABDULLAH BAKR SABRY	20220280
YOUSUF MOHAMED BIOUMY	20220575
AHMED MOHAMED ASRAN	20220038
SALMA HOSAM ELDEEN	20220208
AMIRA ATEF	20220078

Thanks For Reading ❤️