Algorithm Design and Analysis

Homework 2

Haochen Huang
5140309485

October 30, 2016

1

1.1 Question

There are two sorted arrays nums1 and nums2 of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Example 1:
nums1 = [1, 3]
nums2 = [2]
The median is 2.0
Example 2:
nums1 = [1, 2]
nums2 = [3, 4]
The median is (2 + 3)/2 = 2.5
Input:
int nums1[]; int m;
int nums2[]; int n;
Output:
double median.

1.2 Answer

Since the time complexity is O(log (m+n)),clearly we need to use the binary search.

The problem is similar to finding the K-th element in two sorted array. Firstly, if (m+n) is odd, let k be (m+n)/2+1. Else, we need to calculate the average of k = (m+n)/2+1 and k = (m+n)/2.

For $(a_0, a_1, \ldots, a_{m/2})$,$(a_{m/2+1}, a_{m/2+2}, \ldots, a_m)$,$(b_0, b_1, \ldots, b_{n/2})$,$(b_{n/2}, b_{n/2+1}, \ldots, b_n)$,

Assume $a_{m/2} \geq b_{n/2}$, if $m/2 + n/2 + 1 \geq k$, and , we can abandon all elements in section $(a_{m/2+1}, a_{m/2+2}, \ldots, a_m)$ because these elements must be greater than $a_{m/2}$ and greater than the median. (If $a_{m/2} \leq b_{n/2}$, abandon $(b_{b/2+1}, b_{n/2+2}, \ldots, b_n)$.)

If $m/2 + n/2 + 1 \leq k$, we can abandon all elements in section $(b_0, b_1, \ldots, b_{n/2})$. (If $a_{m/2} \leq b_{n/2}$, abandon $(a_0, a_1, \ldots, a_{m/2})$.)

Time complexity O(log (m+n))

1.3 Code

C++ :

```
double Median(int nums1[], int m, int nums2[], int n){
        if((n+m)%2 ==0){
                return (calMid(nums1,m,nums2,n, (m+n)/2) +
                calMid(nums1,m,nums2,n, (m+n)/2+1))/2.0;
        }
        else
                return calMid(nums1,m,nums2,n, (m+n)/2+1);
}

int calMid(int a[], int n, int b[], int m, int k){
    if (n <= 0) return b[k-1];
    if (m <= 0) return a[k-1];
    if (k == 1) return a[0]<b[0] ? a[0] : b[0];

    if (b[m/2] >= a[n/2]){
        if ((n/2 + 1 + m/2) >= k){
            return calMid(a, n, b, m/2, k);
        }
        else{
            return calMid(a+n/2+1, n-(n/2+1), b, m, k-(n/2+1));
        }
    }
    else{
        if ((m/2 + 1 + n/2) >= k){
            return calMid( a, n/2,b, m, k);
        }
        else{
            return calMid(a, n, b+m/2+1, m-(m/2+1), k-(m/2+1));
        }
    }
}
```

2

2.1 Question

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array [-2,1,-3,4,-1,2,1,-5,4], the contiguous subarray [4,-1,2,1] has the largest sum = 6.

Input:

int A[]: the input array.

int N: length of A.

Output:

return the largest sum.

2.2 Answer

For an element with index i in the array, the possible sub array with the maximum sum is the the sum of the (i-1) element plus the i-th element. It depends on whether the sum is positive or negative.

if $sum[i] \geq 0, sum[i+1] = sum[i] + A[i+1]$ else, $sum[i+1] = A[i+1]$

Then the answer is the maximum of every sum.

Time complexity O(n)

2.3 Code

C++ :

```
int subarr(int * A,int N) {
        int sum = INT_MIN, ans = INT_MIN;
        for (int i=0; i<N; ++i){
                sum = sum>0? A[i]+sum:A[i];
                ans = max(sum,ans);
        }

        return ans;
}
```

3

3.1 Question

Given a non-empty array containing only positive integers, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

Each of the array element will not exceed 100.

The array size will not exceed 200.

Example 1:

Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].

Example 2:

Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.

Input:

int A[]: the input array.

int N: length of A.

Output:

return true or false.

3.2 Answer

First calculate the sum of the array. If the sum is odd, it must be wrong. If not, get the half of the sum.

Use a vector to record whether the number 0 to half can get by the sum of a subset. If dp[j-A[i]] is true, by adding the element A[i], j can also be added up, so dp[j] is also true. And we want the boolean value of dp[half] at last.

Time complexity O($n \times sumofarray$)

3.3 Code

C++ :

```
bool subeql(int * A,int N) {
        int sum=0,half = 0;
        for (int i=0; i<N; ++i){
                sum += A[i];
        }
        if (sum%2==1) return false;
        else half = sum/2;

        bool * dp = new bool [sum];
        for (int i=0; i<sum; ++i){
                dp[i] = false;
        }
        dp[0] = 1;

        for (int i=0; i<N; ++i){
                for (int j = half; j >= A[i]; --j) {
                    dp[j] = dp[j] || dp[j - A[i]];
                }
        }

        return dp[half];
}
```