- The painter's partition problem
- How can one become good at Data structures and Algorithms easily?
- Understanding The Coin Change Problem With Dynamic Programming
- Subset array sum by generating all the subsets
- Make all numbers of an array equal
- Reach the numbers by making jumps of two given lengths

Some useful C++ tricks for beginners in Competitive Programming

Check if a number is perfect square without finding square root

Find triplets in an array whose AND is maximum

Sum of XOR of all subarrays

Sum of similarities of string with all of its suffixes

Sum of the first N Prime numbers

Sort the array of strings according to alphabetical order defined by another string

Length of the longest Subarray with only Even

Elements

Check if it is possible to reach a number by making jumps of two given length

Find the number in a range having maximum product of the digits

Make Binary Search Tree

Print matrix in snake pattern from the last column

Maximum of all Subarrays of size k using set in C++ STL

Number of array elements derivable from D after performing certain operations

Smallest Pair Sum in an array

Minimum number of sets with numbers less than Y

Operations required to make the string empty

Count distinct substrings that contain some characters at most k times

Color N boxes using M colors such that K boxes have different color from the box on its left

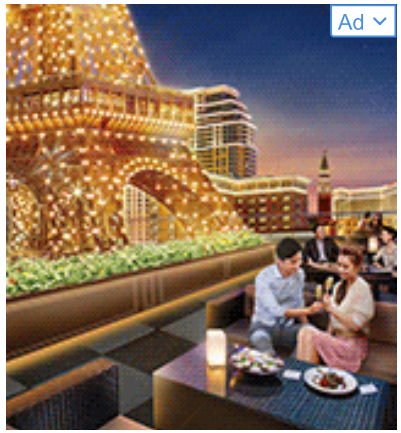
Find two numbers whose divisors are given in a random order

Count the number of vowels occurring in all the substrings of given string

Sum of the multiples of two numbers below N

Minimum number of increasing subsequences

Maximum length substring with highest frequency in a string



The painter’s partition problem | Set 2

We have to paint n boards of length {A1, A2, .. An}. There are k painters available and each takes 1 unit time to paint 1 unit of board. The problem is to find the minimum time to get this job done under the constraints that any painter will only paint continuous sections of boards, say board {2, 3, 4} or only board {1} or nothing but not board {2, 4, 5}.

Examples :

```
Input : k = 2, A = {10, 10, 10, 10}
Output : 20.
Here we can divide the boards into 2
equal sized partitions, so each painter
gets 20 units of board and the total
time taken is 20.

Input : k = 2, A = {10, 20, 30, 40}
Output : 60.
Here we can divide first 3 boards for
one painter and the last board for
second painter.
```

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

In the previous post we discussed a dynamic programming based approach having time complexity of $O(K * N^2)$ and $O(k * N)$ extra space.
In this post we will look into a more efficient approach using binary search. We know that the invariant of binary search has two main parts:
* the target value would always be in the searching range.
* the searching range will decrease in each loop so that the termination can be reached.

We also know that the values in this range must be in sorted order. Here our target value is the maximum sum of a contiguous section in the optimal allocation of boards. Now how can we apply binary search for this? We can fix the possible low to high range for the target value narrow down our search to get the optimal allocation.

We can see that the highest possible value in this range is the sum of all the elementsin the array and this happens when we allot 1 painter all the sections of the board. The lowest possible value of this range is the maximum value of the array max, as in this allocation we can allot max to one painter and divide the other sections such
that the cost of them is less than or equal to max and as close as possible to max. Now if we consider we use x painters in the above scenarios, it is obvious that as the value in the range increases, the value of x decreases and vice-versa. From this we can find the target value when x=k and use a helper function to find x, the minimum number of painters required when the maximum length of section a painter can paint is given.

C++

```cpp
// CPP program for painter's partition problem
#include <iostream>
using namespace std;

// return the maximum element from the array
int getMax(int arr[], int n)
{
    int max = INT_MIN;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the elements in the array
int getSum(int arr[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required painters for given maxlen
// which is the maximum length a painter can paint
int numberOfPainters(int arr[], int n, int maxLen)
```

```
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++) {
        total += arr[i];

        if (total > maxLen) {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

int partition(int arr[], int n, int k)
{
    int lo = getMax(arr, n);
    int hi = getSum(arr, n);

    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        int requiredPainters = numberOfPainters(arr, n, mid);

        // find better optimum in lower half
        // here mid is included because we
        // may not get anything better
        if (requiredPainters <= k)
            hi = mid;

        // find better optimum in upper half
        // here mid is excluded because it gives
        // required Painters > k, which is invalid
        else
            lo = mid + 1;
    }

    // required
```

```
    return lo;
}

// driver function
int main()
{
    int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << partition(arr, n, k) << endl;
    return 0;
}
```

Java

```
// Java Program for painter's partition problem
import java.util.*;
import java.io.*;

class GFG
{
// return the maximum element from the array
static int getMax(int arr[], int n)
{
    int max = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

// return the sum of the elements in the array
static int getSum(int arr[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
```

```
}

// find minimum required painters for given maxlen
// which is the maximum length a painter can paint
static int numberOfPainters(int arr[], int n, int maxLen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++) {
        total += arr[i];

        if (total > maxLen) {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}

static int partition(int arr[], int n, int k)
{
    int lo = getMax(arr, n);
    int hi = getSum(arr, n);

    while (lo < hi) {
        int mid = lo + (hi - lo) / 2;
        int requiredPainters = numberOfPainters(arr, n, mid);

        // find better optimum in lower half
        // here mid is included because we
        // may not get anything better
        if (requiredPainters <= k)
            hi = mid;

        // find better optimum in upper half
        // here mid is excluded because it gives
        // required Painters > k, which is invalid
```

```
            else
                lo = mid + 1;
        }

        // required
        return lo;
    }

    // Driver code
    public static void main(String args[])
    {
     int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

        // Calculate size of array.
        int n = arr.length;
            int k = 3;
     System.out.println(partition(arr, n, k));
    }
    }

    // This code is contributed by Sahil_Bansall
```

C#

```
    // C# Program for painter's
    // partition problem
    using System;

    class GFG
    {

    // return the maximum
    // element from the array
    static int getMax(int []arr, int n)
    {
        int max = int.MinValue;
        for (int i = 0; i < n; i++)
            if (arr[i] > max)
```

```
            max = arr[i];
    return max;
}

// return the sum of the
// elements in the array
static int getSum(int []arr, int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += arr[i];
    return total;
}

// find minimum required
// painters for given
// maxlen which is the
// maximum length a painter
// can paint
static int numberOfPainters(int []arr,
                            int n, int maxLen)
{
    int total = 0, numPainters = 1;

    for (int i = 0; i < n; i++)
    {
        total += arr[i];

        if (total > maxLen)
        {

            // for next count
            total = arr[i];
            numPainters++;
        }
    }

    return numPainters;
}
```

```
static int partition(int []arr,
                     int n, int k)
{
    int lo = getMax(arr, n);
    int hi = getSum(arr, n);

    while (lo < hi)
    {
        int mid = lo + (hi - lo) / 2;
        int requiredPainters =
                      numberOfPainters(arr, n, mid);

        // find better optimum in lower
        // half here mid is included
        // because we may not get
        // anything better
        if (requiredPainters <= k)
            hi = mid;

        // find better optimum in upper
        // half here mid is excluded
        // because it gives required
        // Painters > k, which is invalid
        else
            lo = mid + 1;
    }

    // required
    return lo;
}

// Driver code
static public void Main ()
{
    int []arr = {1, 2, 3, 4, 5,
                 6, 7, 8, 9};

    // Calculate size of array.
    int n = arr.Length;
    int k = 3;
```

```
        Console.WriteLine(partition(arr, n, k));
    }
    }

    // This code is contributed by ajit
```

PHP

```
    <?php
    // PHP program for painter's
    // partition problem

    // return the maximum
    // element from the array
    function getMax($arr, $n)
    {
        $max = PHP_INT_MIN;
        for ($i = 0; $i < $n; $i++)
            if ($arr[$i] > $max)
                $max = $arr[$i];
        return $max;
    }

    // return the sum of the
    // elements in the array
    function getSum($arr, $n)
    {
        $total = 0;
        for ($i = 0; $i < $n; $i++)
            $total += $arr[$i];
        return $total;
    }

    // find minimum required painters
    // for given maxlen which is the
    // maximum length a painter can paint
    function numberOfPainters($arr, $n,
                              $maxLen)
```

```
{
    $total = 0; $numPainters = 1;

    for ($i = 0; $i < $n; $i++)
    {
        $total += $arr[$i];

        if ($total > $maxLen)
        {

            // for next count
            $total = $arr[$i];
            $numPainters++;
        }
    }

    return $numPainters;
}

function partition($arr, $n, $k)
{
    $lo = getMax($arr, $n);
    $hi = getSum($arr, $n);

    while ($lo < $hi)
    {
        $mid = $lo + ($hi - $lo) / 2;
        $requiredPainters =
                    numberOfPainters($arr,
                                     $n, $mid);

        // find better optimum in
        // lower half here mid is
        // included because we may
        // not get anything better
        if ($requiredPainters <= $k)
            $hi = $mid;

        // find better optimum in
        // upper half here mid is
```

```
            // excluded because it
            // gives required Painters > k,
            // which is invalid
            else
                $lo = $mid + 1;
        }

        // required
        return floor($lo);
    }

    // Driver Code
    $arr = array(1, 2, 3,
                 4, 5, 6,
                 7, 8, 9);
    $n = sizeof($arr);
    $k = 3;

    echo partition($arr, $n, $k) , "\n";

    // This code is contributed by ajit
    ?>
```

Output :

```
17
```

For better understanding, please trace the example given in the program in pen and paper.

The time complexity of the above approach is $O(N * log(sum(arr[])))$.

References:

https://articles.leetcode.com/the-painters-partition-problem-part-ii/

https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/

Asked in: Google, Codenation.

Recommended Posts:

Partition problem | DP-18

The painter's partition problem

Print equal sum sets of array (Partition Problem) | Set 2

Print equal sum sets of array (Partition problem) | Set 1

Partition the array into three equal sum segments

Maximum average sum partition of an array

Count number of ways to partition a set into k subsets

Bell Numbers (Number of ways to Partition a Set)

Number of ways to partition a string into two balanced subsequences

Partition a set into two subsets such that the difference of subset sums is minimum

Tiling Problem

How to get rid of Java TLE problem

0-1 Knapsack Problem | DP-10

The Celebrity Problem

Box Stacking Problem | DP-22

rsatish1110
Check out this Author's contributed articles.

If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please Improve this article if you find anything incorrect by clicking on the "Improve Article" button below.

Improved By : jit_t

Article Tags : Competitive Programming | Divide and Conquer | Dynamic Programming | Searching | Binary Search | Codenation | Google

Practice Tags : Google | Codenation | Searching | Dynamic Programming | Divide and Conquer | Binary Search

2

To-do Done

4.3

Based on **38** vote(s)

Feedback/ Suggest Improvement | Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!