Container With Most Water

Given n non-negative integers a_1, a_2, ..., a_n, where each represents a point at coordinate (i, a_i). n vertical lines are drawn such that the two endpoints of line$i$ is at (i, a_i) and $(i, 0)$. Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container.

Solution 1

The O(n) solution with proof by contradiction doesn't look intuitive enough to me. Before moving on, read the algorithm first if you don't know it yet.

Here's another way to see what happens in a matrix representation:

Draw a matrix where the row is the first line, and the column is the second line. For example, say `n=6`.

In the figures below, `x` means we don't need to compute the volume for that case: (1) On the diagonal, the two lines are overlapped; (2) The lower left triangle area of the matrix is symmetric to the upper right area.

We start by computing the volume at `(1,6)`, denoted by `o`. Now if the left line is shorter than the right line, then all the elements left to `(1,6)` on the first row have smaller volume, so we don't need to compute those cases (crossed by `---`).

```
  1 2 3 4 5 6
1 x ------- o
2 x x
3 x x x
4 x x x x
5 x x x x x
6 x x x x x x
```

Next we move the left line and compute `(2,6)`. Now if the right line is shorter, all cases below `(2,6)` are eliminated.

```
  1 2 3 4 5 6
1 x ------- o
2 x x       o
3 x x x     |
4 x x x x   |
5 x x x x x |
6 x x x x x x
```

And no matter how this `o` path goes, we end up only need to find the max value on this path, which contains `n-1` cases.

```
  1 2 3 4 5 6
1 x ------- o
2 x x - o o o
3 x x x o | |
4 x x x x | |
5 x x x x x |
6 x x x x x x
```

Hope this helps. I feel more comfortable seeing things this way.

written by kongweihan original link here

Solution 2

Start by evaluating the widest container, using the first and the last line. All other possible containers are less wide, so to hold more water, they need to be higher. Thus, after evaluating that widest container, skip lines at both ends that don't support a higher height. Then evaluate that new container we arrived at. Repeat until there are no more possible containers left.

C++

```
int maxArea(vector<int>& height) {
    int water = 0;
    int i = 0, j = height.size() - 1;
    while (i < j) {
        int h = min(height[i], height[j]);
        water = max(water, (j - i) * h);
        while (height[i] <= h && i < j) i++;
        while (height[j] <= h && i < j) j--;
    }
    return water;
}
```

C

A bit shorter and perhaps faster because I can use raw int pointers, but a bit longer because I don't have `min` and `max`.

```
int maxArea(int* heights, int n) {
    int water = 0, *i = heights, *j = i + n - 1;
    while (i < j) {
        int h = *i < *j ? *i : *j;
        int w = (j - i) * h;
        if (w > water) water = w;
        while (*i <= h && i < j) i++;
        while (*j <= h && i < j) j--;
    }
    return water;
}
```

written by StefanPochmann original link here

Solution 3

The idea is : to compute area, we need to take min(height[i],height[j]) as our height. Thus if `height[i]<height[j]`, then the expression `min(height[i],height[j])` will always lead to at maximum `height[i]` for all other j(i being fixed), hence no point checking them. Similarly when `height[i]>height[j]` then all the other i's can be ignored for that particular j.

```
class Solution {
public:
    int maxArea(vector<int> &height)
    {
        int j=height.size()-1,i=0,mx=0;

        while(i<j)
        {
            mx=max(mx,((j-i)*(min(height[i],height[j]))));

            if(height[i]<height[j])
             i++;
             else if(height[i]>=height[j])
             j--;
        }
        return mx;
    }
};
```

written by franticguy original link here

From Leetcoder.