Homework 1 - BIOMATH 205

SIMON LEE

October 4, 2023

1 Chapter 1

Q22: Write a fast accurate function to evaluate $\tan x$ exploiting the facts

$$tan(x+\pi) = tan(x)$$

$$tan(2x) = \frac{2\tan x}{1-\tan^2 x}$$

$$tanx \approx x + x^3(\frac{1}{3} + \frac{2}{15}x^2 + \frac{17}{315}x^4) \text{ as } x \downarrow 0$$

```
function fast_tan(x)
    # Check if x is close to multiples of
    if abs(x) %   0
        return 0.0
    end

    # Reduce the angle to the range (-/2, /2)
    x = mod(x + /2, ) - /2

    # Use the Taylor series approximation for small x
    if abs(x) < 0.1
        x_cubed = x^3
        return x + x_cubed * (1/3 + x_cubed * (2/15 + x_cubed * 17/315))
    end

    # Calculate tan(x) using the identity tan(2x)
    tan_half_x = fast_tan(x / 2)
    return 2 * tan_half_x / (1 - tan_half_x^2)
end

# Example tan(1) = 1.56...:
x = 1.0
result = fast_tan(x)
println("tan($x)  $result")

>>> tan(1.0)  1.5574012055538191
```

2 Chapter 2

Q2: Show the worst case of quicksort takes on the order of n^2 operations.
Answer:
To demonstrate that the worst-case scenario for the QuickSort algorithm takes on the order of n^2 operations, we can construct an example and analyze its behavior. The worst-case scenario occurs when the pivot chosen at each step consistently divides the array into two unbalanced subarrays, one of size $n-1$ and the other of size 0.

Here is an example that showcases the behavior.

Consider an array of n elements sorted in ascending order:

$1, 2, 3, 4, 5, ..., n-1, n$

In this case, if we chose the lst element as the pivot, the partitioning step will result in one subarray of size $n-1$, and one subarray of size 0.

Now we have reduced the array with one less element. If we continue to always choose the last element as the pivot this behavior will repeat reducing our array size by a single entry.

At each step, we have n elements, and we perform n comparisons to partition them. Since we repeat this process n times (each time with $n-1$ elements, then $n-2$ and so on), the total number of comparisons become:

$n + (n-1) + (n-2) + ... + 1$ which is equal to the arithmetic sequence $\frac{n(n+1)}{2}$

Therefore, the number of comparisons in the worst case scenario is on the order of $\frac{n(n+1)}{2}$ which is proportional to n^2

From this example we see why worst case scenario would result in n^2 operations.

Q4 Given a sorted array of numbers of lengthnand a number c, write a Julia program to find the pair of numbers in the array whose sum is closest to c. An efficient solution can find the pair in $O(n)$ time?

Answer

```
function find_closest_pair(arr::Vector{Int}, c::Int)
    if length(arr) < 2
        return (0, 0)  # Not enough elements for a pair
    end
```

```
    left, right = 1, length(arr)
    closest_diff = abs(arr[left] + arr[right] - c)
    closest_pair = (arr[left], arr[right])

    while left < right
        current_diff = abs(arr[left] + arr[right] - c)
        if current_diff < closest_diff
            closest_diff = current_diff
            closest_pair = (arr[left], arr[right])
        end

        if arr[left] + arr[right] < c
            left += 1
        else
            right -= 1
        end
    end

    return closest_pair
end

# Example of the question
arr = [1, 2, 4, 7, 10, 14]
c = 8
result = find_closest_pair(arr, c)
println("Closest pair to $c is $result")

>>> Closest pair to 8 is (1, 7)
```