CSCI 3104 Spring 2018

Problem Set 3

Merola, Michael

06/04/1998

Problem Set 3

Problem 1

(5 pts total) For parts (1a) and (1b), justify your answers in terms of deterministic QuickSort, and for part (1c), refer to Randomized QuickSort. In both cases, refer to the versions of the algorithms given in Lecture 3.

(a) What is the asymptotic running time of QuickSort when every element of the input A is identical, i.e., for 1 ≤ i, j ≤ n, A[i] = A[j]?

For the deterministic QuickSort algorithm in any case, best case is $\theta(nlog(n))$ and worst case is $\theta(n^2)$.

In the case that A contains all identical elements, the algorithm must still traverse the entire array. Because each value is equal to each other, the algorithm must also swap each element at every comparison. Due to both of these requirements for this case, the complexity is worst-case at $\theta(n^2)$.

(b) Let the input array A = [9, 7, 5, 11, 12, 2, 14, 3, 10, 6]. What is the number of times a comparison is made to the element with value 3?

A = [9, 7, 5, 11, 12, 2, 14, 3, 10, 6] ...initial array. **pivot** = 6 and **current** = 9

[5, 2, 3] | [6, 9, 7, 11, 12, 14, 10] ...after 1 run. 3 has been compared once. **pivot** = 3 and **current** = 5

[2, 3] | [5] | [,,,] ...after 2 runs. 3 was compared to both 2 and 5, **total comparisons** = 3

At this point, 3 has been sorted correctly and will not be touched again by the algorithm. So for this array, the element 3 has made **3** total comparisons during the algorithms run.

(c) How many calls are made to random-int in (i) the worst case and (ii) the best case? Give your answers in asymptotic notation.

(i) Worst Case: random-int always chooses smallest number

note: random-int is called everytime the algo needs a new pivot.

A = [4, 3, 5, 1, 8] ...**pivot** = 1

A = [1] [4, 3, 5, 8] ...**pivot** = 3

A = [1] [3] [4, 5, 8] ...**pivot** = 4

A = [1] [3] [4] [5, 8] ...**pivot** = 5

The algorithm takes 4 calls to random-int before the array is sorted. n=5, so the complexity is $\theta(n-1)$ or just simply $\theta(n)$

(ii) Best Case: random-int always chooses median value

A = [8, 3, 5, 1, 4] ...**pivot** = 4

A = [3, 1] [4] [8, 5] ...**pivot** = 1, 5

A = [1] [3] [4] [5] [8]

The algorithm takes 3 calls to random-int before the array is sorted. n=5, so the complexity is $\theta(n-2)$ or just simply $\theta(n)$

Problem 2

(30 pts total) Professor Trelawney has acquired n enchanted crystal balls, of dubious origin and dubious reliability. Trelawney needs your help to identify which crystal balls are accurate and which are inaccurate. She has constructed a strange contraption that fits over two crystal balls at a time to perform a test. When the contraption is activated, each crystal ball glows one of two colors depending on whether the other crystal ball is accurate or not. An accurate crystal ball always glows correctly according to whether the other crystal ball is accurate or not, but the glow of an inaccurate crystal ball cannot be trusted. You quickly notice that there are four possible test outcomes:

(a) Prove that if n/2 or more crystal balls are inaccurate, Trelawney cannot necessarily determine which crystal balls are accurate using any strategy based on this kind of pairwise test. Assume a worst-case scenario in which the inaccurate crystal balls contain malicious spirits that collectively conspire to fool Trelawney.

Trelawney cannot determine the accuracy of any crystal ball with this method because no scenario of balls paired together can determine which ball has a specific accuracy. If at least $\frac{n}{2}$ balls in the group are innacurate, than there is a $\geqslant \frac{1}{2}$ chance that an inaccurate ball is chosen to be tested in the pair. This means that at least half of the pairs tested will be impossible to tell each ball's accuracy. Otherwise, a pair with both accurate balls will only tell Trelawney that both balls are either innaccurate or accurate; but she still can't determine which. Overall, her contraption is both useless and a waste of time.

(b) Suppose Trelawney knows that more than n/2 of the crystal balls are accurate, but not which ones. Prove that floor(n/2) pairwise tests are sufficient to reduce the problem to one of nearly half the size.

By only utilizing n/2 pairwise tests, we pair two balls with each other and run a test on each pair. If a pair has any ball that glows red, we remove both balls from the set. This way, we ensure that we remove at least 1 innaccurate ball each time. Worst-case, we remove up to 1 accurate ball each time we remove a pair. Using this method, the ratio between accurate/inaccurate balls remains the same. By the end of the pairwise tests, the scale of the problem has been reduced by half (n = n/2), and the number of accurate balls is stil n/2.

(c) Now, under the same assumptions as part (2b), prove that all of the accurate crystal balls can be identified with Θ(n) pairwise tests. Give and solve the recurrence that describes the number of tests.

This function is represented by $\frac{n}{2}$ pairwise crystal-ball tests over $T(\frac{n}{2})$ ...so...

$T(n) = T(\frac{n}{2}) + \frac{n}{2}$

$= T(\frac{n}{4}) + \frac{n}{4} + \frac{n}{2}$ .....recurrence unrolling

$= T(\frac{n}{8}) + \frac{n}{8} + \frac{n}{4} + \frac{n}{2}$

...

$= \sum_{i=1}^{n} \frac{n}{2i}$ .....establish sum

$= n - 1$

Guess: n - 1 = $\theta(n)$

$\lim_{n\to\infty}[\frac{n-1}{n}$ ...l'hopitals rule

$\lim_{n\to\infty}[\frac{1}{1} = 1$ ...simplify

The limit produced a constant number, so the number of pairwise tests is confirmed to be $\theta(n)$

Problem 3

(20 pts) Professor Dumbledore needs your help. He gives you an array A consisting of n integers A[1], A[2], . . . , A[n] and asks you to output a two-dimensional n × n array B in which B[i, j] (for i < j) contains the sum of array elements A[i] through A[j], i.e., B[i, j] = A[i] + A[i + 1] + · · · + A[j]. (The value of array element B[i, j] is left unspecified whenever i ≥ j, so it doesn't matter what the output is for these values.)

Dumbledore suggests the following simple algorithm to solve this problem:

In [2]:
```
def dumbledoreSolve(A) {
    for i = 1 to n {
        for j = i+1 to n {
            s = sum(A[i..j]) // look very closely here
            B[i,j] = s
}}}
```

```
  File "<ipython-input-2-cd213584bdeb>", line 2
    for i = 1 to n :
          ^
SyntaxError: invalid syntax
```

(a) For some function g that you should choose, give a bound of the form Ω(g(n)) on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

For dumbledoreSolve, there are two main operations:

```
s = sum(A[i..j]) => j-i operations
B[i,j] = s       => 1 operation
```

With these operations, we can formulate a sum to represent the function:

$\sum_{i=1}^{n}[\sum_{j=i+1}^{n}[(j-i)+1]]$ ...reduce inner sum

$= \sum_{i=1}^{n}[\frac{1}{2}(i-n-3)(i-n)]$ ...reduce final sum

$= \frac{1}{6} * n(n^2+3n-4) = f(n)$

Make a guess for the function complexity:

$f(n) = O(n^3)$

Evaluate using limits:

$\lim_{n\to\infty}[\frac{1}{6} * \frac{n(n^2+3n-4)}{n^3}]$ ...simplify

$= \lim_{n\to\infty}[\frac{n^3+3n^2-4n}{6n^3}]$ ...apply L'Hopitals Rule

$= \lim_{n\to\infty}[\frac{3n^2+6n-4}{18n^2}] = \lim_{n\to\infty}[\frac{6n+6}{36n}] = \lim_{n\to\infty}[\frac{6}{36}]$

$= \frac{1}{6}$

The limit evaluated to $\frac{1}{6}$, which is a **constant** and not equal to 0 or infinity.

Therefore, the complexity of this algorithm for input size n is $\theta(n^3)$

Because the complexity is in terms of theta, it is also equal to lower-bound $\Omega(n^3)$

(b) For this same function g, show that the running time of the algorithm on an input of size n is also O(g(n)). (This shows an asymptotically tight bound of Θ(g(n)) on the running time.)

The complexity of this algorithm for input size n is $\theta(n^3)$

Because the complexity is in terms of theta, it is also equal to upper-bound $O(n^3)$

(c) Although Dumbledore's algorithm is a natural way to solve the problem—after all, it just iterates through the relevant elements of B, filling in a value for each— it contains some highly unnecessary sources of inefficiency. Give an algorithm that solves this problem in time O(g(n)/n) (asymptotically faster) and prove its correctness.

In [ ]:
```
def dumbledoreSolve(A) :
    n = A.length()

    for i in range(1, n):
        for j in range(i+1, n):
            if (j == i+1):
                B[i, j] = A[i] + A[j]

            B[i,j] = B[i,j-1] + A[j]

        #end inner for
    #end outer for

#end function
```

My algorithm solves the same problem faster because we access previous calulations instead of solving the same ones multiple times. For example, my algorithm calculates the sum of the element at i and at j, then stores it in the B array immediately. It then retrieves the previous sum from the B array, adds the new j value to the sum, and then adds it to the next spot in the B array. This solution reduces the complexity from $\theta(n^3)$ and $O(n^2)$.

Problem 4

(15 pts extra credit) With a sly wink, Dumbledore says his real goal was actually to calculate and return the largest value in the matrix B, that is, the largest subarray sum in A. Butting in, Professor Hagrid claims to know a fast divide and conquer algorithm for this problem that takes only O(n log n) time (compared to applying a linear search to the B matrix, which would take O(n^2) time).

Hagrid says his algorithm works like this:

• Divide the array A into left and right halves

• Recursively find the largest subarray sum for the left half

• Recursively find the largest subarray sum for the right half

• Find largest subarray sum for a subarray that spans between the left and right halves

• Return the largest of these three answers

On the chalkboard, which appears out of nowhere in a gentle puff of smoke, Hagrid writes the following pseudocode for his algorithm:

In [ ]:
```
hagridSolve(A) {
    if(A.length()==0) { return 0 }
    return hagHelp(A,1,A.length())
}

hagHelp(A, s, t) {
    if (s > t) { return 0 }
    if (s == t) { return A[s] }

    m = (s + t) / 2

    leftMax = sum = 0
    for (i = m-1, i >= s, i--) {
        sum += A[i]
        if (sum > leftMax) { leftMax = sum }
    }

    rightMax = sum = 0
    for (i = m+1, i <= t, i++) {
        sum += A[i]
        if (sum > rightMax) { rightMax = sum }
    }

    spanMax = leftMax + rightMax
    halfMax = max(hagHelp(s, m-1), hagHelp(m+1, t) )

    return max(spanMax, halfMax)
}
```

(i) Identify and fix the errors in Hagrid's code

In [ ]:
```
hagridSolve(A) {
    if(A.length()==0) { return 0 }
    return hagHelp(A,0,A.length()-1) ##FIXED BOUNDS
}

hagHelp(A, s, t) {
    if (s > t) { return 0 }
    if (s == t) { return A[s] }

    m = (s + t) / 2

    leftMax = sum = 0
    for (i = m-1, i >= s, i--) {
        sum += A[i]
        if (sum > leftMax) { leftMax = sum }
    }

    rightMax = sum = 0
    for (i = m, i <= t, i++) { ##FIXED i
        sum += A[i]
        if (sum > rightMax) { rightMax = sum }
    }

    spanMax = leftMax + rightMax
    halfMax = max(hagHelp(s, m-1), hagHelp(m, t) ) ##FIXED BOUNDS

    return max(spanMax, halfMax)
}
```

Sources

People

Krish Dholakiya

Gustav Solis