INTRO TO ALGORITHM, CSE 241

PROFESSOR ZHANG

Homework 2

Peiyun(Seed) Zeng

EN:437801

Due date: October 5, 2014

PROBLEM 1

(a):Find the running time for (i) for the worst case (ii) average case (expected running time) of the given algorithm

```
GIVEM-THEIR-HATS-BACK(G, H)
1  if |H| = |G| = 1
2      then give the hat to the gentleman, return.
3      else
4           Pick a random hat h ∈ H
5           for each gentleman g ∈ G
6                do Gentleman g tries on hat h:
7                   if it fits,
8                     then give hat h to gentleman g
9                          Let gentleman group G' = G \ {g}
10                         Let hat pile H' = H \ {h}
11                         break out of for loop
12          GIVEM-THEIR-HATS-BACK(G', H')
```

Solution:

(i): For the worst case, suppose we will find the match at the last position of each iteration. Thus, for the loop from line 6-12 (**Let's call it Loop S from now on**), when the gentleman group size G is n, it takes $\Theta(n)$ to run through. at line 13, we have reduced the problem size to n-1. So the recurrence is then T(n) = c*n +T(n-1).

Now solve the recurrence T(n) = c*n + T(n-1)

$$\begin{aligned}
T(n) &= cn + T(n-1) \\
&= c(n-1) + cn + T(n-2) \\
&= c(n-2) + c(n-1) + cn + T(n-3) \\
&= 1 + 2 + 3 + ... + (n-2) + (n-1) + n + T(0) \\
&= T(0) + n(n+1)/2 \\
&= \frac{n*(n+1)}{2} \\
&= \Theta(n^2)
\end{aligned}$$

(ii): For the expected running time, we have to calculate the expected value of number of times Loop S iterates (Let's call it **X** from now on) when the gentleman group size $G = n$.

If the ith hat fits the gentleman, i times comparisons (iteration of Loop S) have already happened. The chance for the match happen at any position (any i) will be P(r) = $frac1n$ since there is equal chance for the match happen at any position.

$$\begin{aligned}
E[x] &= \sum_{i=1}^{n} i * P(r) \\
&= 1 * (\frac{1}{n}) + 2 * (\frac{1}{n}) + 3 * (\frac{1}{n}) ... + n * (\frac{1}{n}) \\
&= \frac{n^2 + n}{2n} \\
&= \Theta(n)
\end{aligned}$$

That's the expectation value for iterations of loop S happens for $G = n$, then after the hat matching finished for $G = n$, next recursive call will be left with $G = n - 1$ size gentleman group. so now the expected running time for the algorithm $E[T(n)] = E[\Theta(n) + T(n-1)]$

$$
\begin{aligned}
E[T(n)] &= E[\Theta(n) + T(n-1)] \\
&= \Theta(n) + E[T(n-1)] = cn + E[T(n-1)] \\
&= c(n-1) + cn + E[T(n-2)] \\
&= c(1) + c(2) + ... + c(n-2) + c(n-1) + c(n) + E[T(0)] \\
&E[T(0)] \text{ is zero since expectation running time for 0 size pile is zero} \\
&= c(1) + c(2) + ... + c(n-2) + c(n-1) + c(n) \\
&= \sum_{i=1}^{n} c * i \\
&= \frac{n^2 + n}{2} \\
&= \Theta(n^2)
\end{aligned}
$$

So the Final conclusion is:

$\mathbf{E[T(n)]} = \Theta(\mathbf{n^2})$.

So the expected running time for the algorithm is, in fact, same as the worst case also $\Theta(n^2)$

(b):Design an algorithm with a better time bound than the one in part (a). Be sure to explain your algorithm well, argue its correctness, and analyze its expected running time. Also analyze its worse case running time.

Solution:

Abstract

The magic lies in the sentence **Gentlemen can tell whether the hat is too big or small**. Thus we can use this information to partition the Gentleman pile (G) and Hats pile (H). Essentially, when we randomly pick a hat from the hats pile, we use it as a pivot ($\mathbf{h_{pivot}}$) to partition the H pile. after the match happens, we use the Gentleman that fits the h_{pivot} as $\mathbf{g_{pivot}}$ to partition the H pile. if we do this recursively, we are essentially **Quicksorting** these two piles. Since $\mathbf{g_{pivot}}$ fits h_{pivot}, when each recursive call ends, we have given the correct hat back to the right gentleman. And the size of two piles are decreasing, recursive calls will terminates. So in theory, this should work. Let's move to a more detailed analysis though

So we are gonna look at Pseudo code and then analyze the correctness and complexity

Pseudo Code

Quicksort-Give-Hats-Back (G,H)

1. **if** $|G| = |H| = 1$
2. **do** return the HAT back to the GENTLEMAN
3. **return**
4. **else**
5. Randomly pick a hat from H as h_{pivot}
6. **For** each gentleman g in G
7. **do** Gentleman g tries h_{pivot}
8. **if** fits, set g as g_{pivot}
9. **if** head is larger than hat, put g to G_{sm} (Gsm is the small heads party for Gentlemen)
10. **else**, put g to G_{bt} (G_{bt} is the bigger heads party for Gentlemen)
11. **ENDFOR**
12. **For** each h in H, $h \neq h_{pivot}$
13. **do** g_{pivot} tries hat
14. **if** hat h is large, put h to H_{bt}
15. **else**, put h to H_{sm}
16. **ENDFOR**
17. return h_{pivot} to g_{pivot}
18. **fi**
19. Quicksort-Give-Hats-Back (G_{sm} ,H_{sm})
20. Quicksort-Give-Hats-Back (G_{bt},H_{bt})

Proof of correctness:

1. Prove partial correctness u

2. Prove the termination of the recursion

3. Combination of 1, 2 will prove the total correctness

1.**Partial Correctness:**

in the **initial recursive call** through **Loop 5-17**, $\mathbf{h_{pivot}}$ is correctly returned to $\mathbf{g_{pivot}}$. Then we need to prove that this correct return can happen for **(G_{sm} ,H_{sm}) and (G_{bt},H_{bt}).** in other words, will there be correct correspondence left for **(G_{sm} ,H_{sm}) and (G_{bt},H_{bt}).**

since we use $\mathbf{h_{pivot}}$ to partition Gentlemen group G, every member in $\mathbf{G_{sm}}$ will have smaller heads than $\mathbf{g_{pivot}}$. We also used $\mathbf{g_{pivot}}$ to partition hat pile H, so all hats in $\mathbf{H_{sm}}$ is gonna be smaller than $\mathbf{h_{pivot}}$. This means, everyone in $\mathbf{G_{sm}}$ will find the perfect hat all in $\mathbf{H_{sm}}$. Similarly, we can prove that everyone in $\mathbf{G_{bt}}$ will find the perfect hat all in $\mathbf{H_{bt}}$ in the end when G

and H got both reduced to 1, we give the hat to the last gentleman, so we proved that every time through the recursive call, a correct return is gonna happen

I think this is enough to prove the partial correctness of the algorithm in theory. Yet, to make it sound more professional and since i am typing the answer in stead of writing it by hand, i will also demonstrate the**Formal 3 parts (inductive) Loop invariant proof for partial correctness** here **The partial Correctness is proved**

Proof. Partial correctness for algorithm. Notation:
P as the pre condition: Gentlemen group G and Hats pile H. there is one-one correspondence between gentleman and hat
Q as the post condition:All
S_0 is code of line 5
S_1 is the code from 6-17
S_3 is code from 1-4
Loop Invariant(LI):all members in G_{sm} has smaller heads than g_{pivot} ;all members in G_{bt} has bigger heads than g_{pivot} ;all hats in H_{sm} is smaller than hpivot;all hats in H_{bt} is bigger than hpivot;

- Pre $\{S_0\}$LI– LI is true initially
  In the beginning, since G_{bt} and G_{sm} , H_{sm} and H_{bt} are all empty, so LI are automatically true

- LI $\{ S_1 \}$LI Since we used $\mathbf{h_{pivot}}$ to partition the Gentlemen pile G, and $\mathbf{h_{pivot}}$ fits $\mathbf{g_{pivot}}$. Therefore all members in G_{bt} has bigger heads than g_{pivot} and all members in G_{sm} has smaller heads than g_{pivot}
  Since we used $\mathbf{g_{pivot}}$ to partition the Hats pile h, and $\mathbf{h_{pivot}}$ fits $\mathbf{g_{pivot}}$. Therefore all members in H_{bt} is bigger than h_{pivot} and all members in H_{sm} is smaller heads than h_{pivot}
  Obviously, the lI holds after each iteration.

- Termination Condition {LI }Post Condition–LI and termination condition should give us post condition
  when terminates, G = H =1 , the last match of hat and gentleman is completed. since LI is maintained in each iteration, so every iteration a correct return happened. Thus, Post condition is satisfied

□

2.termination of the recursion

Obviously, since each iteration one good match is happening, G and H size is decreasing, so eventually the recursion is gonna terminate

3. Total correctness

Combine 1 and 2, we have proved the total correctness of the algorithm

Complexity Analysis

So this is, apparently a standard randomized Quicksort Algorithm. So let's, as Prof.Zhang suggested, introduce our friend indicator $\mathbf{X_i}$, where $\mathbf{X_i} = 1$ if the ith largest size is the pivot. $\mathbf{X_i} = 0$, otherwise. Probability of this happen to any hat P(i) = $\frac{1}{n}$

So exactly same as the randomized algorithm we solved in class. the recurrence is gonna be $T(n) = \sum_{i=0}^{n-1} Xi * (T(n-i-1) + \Theta(n))$

SO it's the exactly same thing we solved in class. Since **Prof.Zhang** said on Piazza we donot need to prove things we discussed in class. So we can just state here. The Expected running time for this randomized Quicksort Algorithm $E[T(n)] = \Theta(n * Log(n))$ and the worst case is gonna be very bad/nasty/viscous/no one loved $\Theta(n^2)$

(c):Suppose the gentlemen were less intelligent, and every time you gave a hat to them, they either said that it fit or did not fit, without giving any information on whether it was too small or too big. Does your algorithm still work? Does the algorithm given in part (a) still work?

Solution:

Obviously, the dumb code in part(a) is **still** gonna work since it is not dependent upon the property of GENTLEMAN telling the size of the hat. it is basically a brute-force algorithm that will work whatsoever, yet at a pathetic rate–$\Theta(n^2)$

Unfortunately, our smart code will **NOT** work here since it depends on the intelligence of GENTLEMEN to tell the size of the hat. The partition will not work without this

Yet,Pro Tip, if you cannot even tell whether your hat is too big or too small. **DONOT WEAR IT**

INTRO TO ALGORITHM, CSE 241

PROFESSOR ZHANG

Homework 2

Peiyun(Seed) Zeng

EN:437801

Due date: October 5, 2014

PROBLEM 2

You are given an array A of n positive integers in the range of 1 to 65536 and a positive integer w. Describe an algorithm to find the indices to a pair of integers in A whose sum is equal to w, if such a pair exists. If there are more than one such pair, it is sufficient to find any one.

Solution:

Intro

The task is to find the pair of positive integers of which the sum is the same as the given target sum value T. The "easy" solution is obviously to find all possible sums, which will result in a n^2 running. this n^2 running time will make any large size problem unsolvable. fortunately, i am more sophisticated than that. The algorithm i will present here will give a $N * Log(N)$ worst case running time

Abstract

this algorithm has two parts.

Part 1 – Sorting
Before we get down to the real business, we will sort the given array A using **Merge Sort**. obviously, We will have to pay $N * Log(N)$ price
Part 2 – Binary Searching After part 1, array A is sorted. Now Part 2 the algorithm will scan through the **sorted** array. For each element with value X, the algorithm will apply **binary search** to find another element with value **T-X**, where T is the target Sum. If find the element,return the pair. if not, return none.

Detail & **Pseudo code**

Part 1–Sorting
This is just a easy merge sort

```
msort( Array A )
     if ( n == 1 ) return a

     left  = a[0] ... a[n/2]
     right = a[n/2+1] ... a[n]

     left = mergesort( left )
     right = mergesort( right )

     return merge( left, right )


merge( array a, array b )
     array c

     while ( a and b have elements )
          if ( a[0] > b[0] )
               add b[0] to the end of c
               remove b[0] from b
          else
```

```
            add a[0] to the end of c
            remove a[0] from a
    while ( a has elements )
         add a[0] to the end of c
         remove a[0] from a
    while ( b has elements )
         add b[0] to the end of c
         remove b[0] from b
    return c
```

Part 2–Binary Search

```
    Binary Search(A,low,high,Value)

     if (high < low)
         return None;
     mid = (low + high) / 2;
     if (A[mid] > value)
         return BinarySearch(A,low,mid,value);
     else if (A[mid] < value)
         return BinarySearch(A,mid,high,value);
     else
         return mid;
}
```

Part3–Main Function(Findsum)

```
FindSum(array A, int w) {
    1. A = msort(A);
    2. For i in A:
    3.   array C = array B with i removed;
         low = 0;
```

```
            high = C.length
            indices_2 = BinarySearch(C,low,high,w-element);
      4.    if indices not None
               return a tuple (i,indices_2);
      5.return None
}
```

Proof of Correctness

Part1–sorting employs a simple merge sort algorithm which we knows for sure works. So after 1 in *Findsum*, Array A is sorted. the next thing we left to prove whether the binary search searching for (T-x) works. Here, i will use proof by induction to show the correctness of the Binary Search

Proof. Binary Search

- **Base case:** n = high - low = 0
  The algorithm will return None, which is correct
- **Inductive Hypothesis**:
  Assume BinarySearch works for k = high-low, k<n.

- **Goal Statement:**
  we want to prove, if Binary Search works for a array size of k, it works for k+1
- **Inductive step:**
  There are three cases

  - **value = A[mid].** Clearly the function works correctly
  - **value < A[mid].** BinarySearch(A,low,mid,value) is called since mid-low< high-low = k, according to our inductive hypothesis, a correct indices for targeted value should be returned
  - **value > A[mid].** BinarySearch(A,mid,high,value) is called since high-mid< high-low = k, according to our inductive hypothesis, a correct indices for targeted value should be returned

- **Conclusion:**
  Since we proved that
  1.if Inductive hypothesis holds,the Goal statement holds.
  2.base case holds According to the principle of induction, we proved that BinarySearch works for array size of n

□

Partial Correctness:
Now let's get back to the FindSum function. in The loop 3, We will scan through array A, the binary search will search for w-x for each element. Since we proved the BinarySearch works, if any pair exists, we will be able to find it. If there is no valid pair, the algorithm will return None in the end, which is also correct.

Proof of termination
So, we know that the loop 2-4 will terminates since we are scanning through a finite array, which has a finite end.
Total Correctness:
Combined with the proof of partial correctness and proof of termination, the Total Correctness is proved
Complexity Analysis:
we know for the mergesort, we discussed in class, we have to pay a $\Theta(N * \log n)$ price right away (no need for showing the proof according to **Prof.Zhang**).How about our binary search? Obviously the recurrence for binary search is $T(n) = T(n/2) + O(1)$.

- $a = 1, b = 2$; $x = \log ba = 0$
- $y = 0, k = 0$
- x = y. this corresponds to Case 3 of master theorem
- $T(n) = \Theta(\log n) = c \log n$

now go back to the main function, FindSum. for step 1 we pay $\Theta(n * \log n)$ price. for loop 2-4,There are n iterations, while in each iteration, it takes, as we just proved, $\log n$ to run a

binary search. so the running time for Loop 2-4 is $\Theta(n * \log n)$
So the total complexity for the whole algorithm can be easily calculated.

$$\begin{aligned} T(n) &= \Theta(n * \log n) + \Theta(n * \log n) \\ &= \Theta(n * \log n) \end{aligned}$$

Therefore, the total complexity for the algorithm is $\underline{\Theta(\mathbf{n} * \log \mathbf{n})}$. Of course this is a worst case running time. since no randomization happens in my algorithm, no expected running time analysis is needed

Conclusion for the algorithm design: So, this algorithm works and also works with a fairly good running time $\underline{\Theta(\mathbf{n} * \log \mathbf{n})}$