SE220 Lecture 5&6 Notes

I. Probabilistic Analysis and Randomized Algorithm

Example: The Hiring Problem

Suppose that you need to hire a new office assistant.

1) Employment agency sends you **one** candidate **each day**.
2) You interview that person, and decide either to hire that person(**fire** your current office assistant) or not.
3) Always want the **best** person – if that applicant is **better qualified** than the current office assistant and hire the new applicant.

Goal: Estimate the price for interviewing and hiring.

Solution:

Denote the cost of interviewing and hiring by c_i and c_h. Set the number of people participating in interview and being hired be n_i and n_h. Assume that $c_i \ll c_h$.

Analyze the best and worst case respectively first.

Best case: Only hire once($n_i = n$, $n_h = 1$), since each candidate is worse than all who came before. The total cost $c = c_i n + c_h = O(c_i n + c_h)$

Worst Case: Hire each candidate who came to interview(fire the previous candidate who has been hired in the company)($n_i = n_h = n$), since each candidate is better than all who came before. This situation occurs if the candidates come in strictly increasing order of quality. The total cost $c = n(c_i + c_h) \approx c_h n = O(c_h n)$

However, in our real world, we cannot predict the result in advance, and all the analysis done above is in a known assumption. In other word, the result of the process is random. Thus, we would like to find the cost in average case.

Assume that the applicants come in a random order. When analyzing the running time, which is the cost in this problem, of a randomized algorithm, the expectation of running time over the distribution of values, the quality index of each candidate in this problem, returned by the random number generator are used as both the amount of candidates being interviewed and hired. ($n_i = n_h = E(X)$)

For convenience, use indicator random variables to represent all the possible cases.

$$X_i = \begin{cases} 1 & i\text{th candidate is hired} \\ 0 & i\text{th candidate is not hired} \end{cases}$$

and $X = \sum_{i=1}^{n} X_i$.

Denote the possibility of *i*th candidate is hired by p, thus

$E(X_i) = 1 \cdot p + 0 \cdot (1 - p) = p$

In addition, we know that candidate *i* has the chance to be better than previous *i*(From 0 to $i - 1$), thus $p = \frac{1}{i}$.

Finally, we can obtain

$$E(X) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = \sum_{i=1}^{n} \frac{1}{i} = \int_1^n \frac{dx}{x} = \ln n$$

From integral test, $\int_1^n \frac{dx}{x} \leqslant \ln n + 1 = \ln n + O(1)$

Thus, the total cost $c = \ln n\,(c_i + c_h) \approx c_h \ln n = O(c_h \ln n)$

II. Quicksort

i. Procedure

The algorithm of quicksort is based on ‘divide-and-conquer’, including 3 steps to sort an array A[p : r](ascending order).

(1) Divide: Choose a reference q to partition the array into two subarrays A[p : q-1] and A[q : r] such that

$$\forall a \leqslant A[q]\,(a \in A[p:\ q-1]) \text{ and } \forall a \geqslant A[q]\,(a \in A[q:\ r])$$

(2) Conquer: Sort two subarrays by recursive calls to quicksort.

(3) Combine: no need for that.(Since our goal is to sort every element in the array.)

Obtain the pseudocode

```
Quicksort(A,p,r){
    if(p<r){
        q=Partition(A,p,r);
        Quicksort(A,p,q-1);
        Quicksort(A,q+1,r);
    }
}
```

Obtain the pseudocode of ‘Partition’

```
Partition(A,p,r){
    x=A[r];
    i=p-1;
    for j=p to r-1{
        if(A[j]<=x){
            i=i+1;
            swap(A[j],A[i]);
        }
    }
    swap(A[i+1],A[r]);
    return i+1;
}
```

Why the condition of ‘if’ in Partition is ‘A[j]<=x’?

Since we need to assure that entries in the first subarray are less than pivot and entries in the second subarray are greater than pivot, we have to put some elements satisfying $A[j] \leqslant x$ on the ‘low side’.

Prove the correctness of this algorithm now:

Initialization: Before the loop executes, all the conditions are satisfied, because subarrays are empty.

Maintenance: While the loop is running, if $A[j] \leqslant x$, A[i] and A[j] are swapped then i and j are increased. If $A[i] < x$, only j is increased.

Termination: When the loop ends, j=r and A[i+1], the first entry in the second array, and A[r] are swapped so that all the conditions required are still satisfied through the execution of loop.

Thus, this algorithm is correct.

ii Analysis of time complexity

Worst case: Given array has already been sorted, which means we have to resort the array again. From the quicksort we can know that the total size of subarrays is $n-1$. Obtain the recurrence:

$T(n) = T(n-1) + T(0) + \Theta(n) = \Theta(n^2)$ (the time of partitioning an array to n subarrays is $\Theta(n)$)

Best case: Given array is balanced, and each subarray has less than $\frac{n}{2}$ entries.

Obtain the recurrence:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n \log_2 n)$$

However, we only find the worst and most optimized time complexity under the condition where the reference is the last index of the array. We would like to verify if it holds for any cases, using random algorithm(**The choice of reference is random**).

Worst Case: Obtain the recurrence and apply substitution method to prove it

$$T(n) = (T(q) + T(n-1-q))_{\max} + \Theta(n)\ (q \in [0, n-1])$$

Rewrite $\Theta(n) = cn$

Guess: $T(n) \leqslant dn^2 (\exists d > 0,\ d$ is constant)

Inductive hypothesis: $T(q) \leqslant dq^2$ and $T(n-1-q) \leqslant d(n-1-q)^2$

Substitution:

$$\begin{aligned}
T(n) &\leqslant (dq^2 + d(n-1-q)^2)_{\max} + cn \\
&= d[(n-1)^2 + 2q(q-n+1)]_{\max} + cn \\
&\leqslant d(n-1)^2 + cn (\text{since } q \leqslant n-1) \\
&= d(n^2 - 2n + 1) + cn \\
&= dn^2 + (c-2d)n + d \\
&\leqslant dn^2 (\exists c \leqslant 2d \text{ and } n \text{ is big enough}) \\
&= O(n^2)
\end{aligned}$$

Likewise, we can prove $T(n) = \Omega(n^2) \Rightarrow T(n) = \Theta(n^2)$

Average case: Assume that all values in array are distinct. The running time of ‘Partition’ is dominant cost, since it involves comparisons and exchanges. Start to relate the running time to the number of comparisons between values.

Let X = total number of comparisons in all calls to ‘Partition’. The goal is to compute the expectation of X.
For convenience:

Rename the elements of A as $z_1, z_2, \ldots, z_n$, indicating that $A = \{z_1, z_2, \ldots, z_n\}$ and

define $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}\ (i \leqslant j)$

Let X_{ij} is an indicator random variable referring to ‘if z_i is compared with z_j’.

$$X_{ij} = \begin{cases} 1 & z_i \text{ is compared with } z_j \\ 0 & \text{otherwise} \end{cases}$$

Let the possibility of a successful comparison between z_i and z_j be p, thus

$E(X_{ij}) = p$.

To compute X, we need to understand the whole process of comparison. Notice the set Z_{ij} we defined, we can obtain that j is determined by i to maintain the condition of $i \leqslant j$, which means they are dependent. Since

$Z_{ij} \subsetneqq A \Rightarrow i \in [1, n-1], j \in [i+1, n]$

Thus, $X = \sum\limits_{i=1}^{n-1} \sum\limits_{j=i+1}^{n} X_{ij}$

Since there are two cases of choice of ‘pivot’, which means both z_i and z_j can become ‘pivot’. We can obtain $p=\frac{1}{j-i+1}+\frac{1}{j-i+1}=\frac{2}{j-i+1}$ Thus.

$$
\begin{aligned}
E(X) &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} E(X_{ij}) = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{2}{j-i+1} \\
&= 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{j-i+1} \\
&= 2\sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{1}{k+1} \\
&< 2\sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{1}{k} \\
&= O(n\log_2 n)
\end{aligned}
$$