Lecture 7: Probabilistic analysis

We first finish the discussion of the closest point problem. The key is that the points in the band can not be too close: you only need to check no more than 7 points below a given point to find any closer point distance. See the textbook (Section 33.4, p. 1039) for more details. The running time is then reduced to $O(n \log n)$ (if we ignore the time for sorting, which can be taken care with proper implementation).

The rest of this lecture is based on Chapter 5. The main subject of this lecture is the hiring problem in Chapter 5. It is well explained in the textbook. So I will not repeat it here. The key points are:

- Indicator variables: binary variables associated with random events. An important property of indicator variable is its expected value is equal to the probability of the probability of the associated event. Moreover, indicator variables are commonly used to express a property of a larger system. For example, the number of total hires made is equal to the summation of the indicator variables, each defined for a particular candidate. This is a recurring situation in probabilistic analysis.

- Linearity of expectation: $E(X+Y)=E(X)+E(Y)$. This works even when $X$ and $Y$ are not independent. This is widely used, e.g. when indicator variables are used. See the hiring problem explanation in Chapter 5 for more details.

- Hiring problem. The key is that the probability of the i-th candidate being hired is $1/i$. This is because the i-th candidate is hired if this person is the best among the first $i$ candidates. Note that each of these $i$ candidates has the same chance of being the best of these $i$ individuals. By defining indicator variables for this type of events, we can find the expected number of hires is $\sum_{i=1}^{n} 1/i \approx \ln(n)$. This is much smaller than $n$, the worst case number of hiring.

We give a few more examples on probabilistic analysis. The first example concerns the situation where each of $n$ sailor takes arbitrary unoccupied cabin. We want to compute the expected number of sailors who sleep in their own cabins. We define an indicator variable $S_i$ for each sailor, where $S_i = 1$ if sailor $i$ sleeps in his own cabin. Then $S = \sum_{i=1}^{n} S_i$, and we want to compute $E(S)$. Note $E(S_i) = P(sailor i sleeps in his own cabin) = \frac{1}{n}$, since each sailor has the same chance sleeping in any cabin. By the linearity of expectation, we have $E(S) = \sum_{i=1}^{n} E(S_i) = 1$.

Our second example refers to the expected summation of two fair dices. Define $S_1$ and $S_2$ be the outcome of two throwing. We want to compute $E(S) = E(S_1 + S_2)$. One may use the definition of expected value and compute the probability of $S=i$, for each valid $i$. But this is tedious to compute. It is easier to use the fact $E(S) = E(S_1) + E(S_2)$. Here $E(S_i) = \sum_{v=1}^{6} v \cdot P(S_i = v) = \sum_{v=1}^{6} v/6 = 3.5$. Thus, $E(S) = 3.5 + 3.5 = 7$.

The third example is the birthday paradox. See Section 5.4.1 for detailed explanation. I will not repeat it here.

Lecture 8: Quicksort

We first discussed how to obtain a random permutation. The details can be found in Section 5.3. Briefly, we show why picking numbers within the range $[1 \ldots n^3]$ is likely to give distinct priority values. The argument goes like this: (i) for any pair of two items, the probability of picking exactly the same priority is $1/n^3$ (why?). (ii) there are $O(n^2)$ pairs of items. (iii) so the probability of having no ties in priority values is at least $1 - O(1/n)$, which is small as $n$ increases.

Now Quicksort is yet another sorting algorithm. The way of divide is different from Mergesort: for a given array $A$, it chooses a pivot $p$ and then creates two sets $A^-$ (with elements smaller than the pivot) and $A^+$ (with elements of $A$ larger than the pivot). Then it ensures $A^-$ is before $p$, and $A^+$ is after $p$ in $A$. This can be done in place (i.e. with no additional memory), which we briefly discuss in class. Read pages 171-175 for more details. The running time depends on whether the partition is balanced (called lucky case), or unbalanced (unlucky case). In the case where we have two equal size $A^-$ and $A^+$, the running time $T(n) = 2T((n-1)/2) + \Theta(n) \leq 2T(n/2) + \Theta(n)$. In this case, $T(n) = O(n \log n)$. In the unlucky case, $T(n) = T(n-1) + \Theta(n)$, which leads to $T(n) = O(n^2)$. This can be shown by direct substitution.
Now, if we divide with $A^-$ with $n/10$ elements and $A^+$ with $9n/10$ elements, it turns out this is also a lucky case. To see this, we consider the recursion tree. Each level takes $O(n)$ time in divide and combine. Let the number of levels be $k$. We trace from the root to the leaf which going through $k$ edges. Each time, the size of subproblem reduces by a factor of $9/10$. Since at leaf the size of subproblem is $1$, we have $n(9/10)^k = 1$, i.e. $k = \log_{10/9}n$. So the total running time in this case is $O(n \log n)$.

Suppose we alternative between “lucky” and “unlucky”, where lucky means we divide the problem into half and unlucky means we only reduce the problem size by one. Then we have: $L(n) \leq 2U(n/2) + O(n)$, $U(n/2) \leq L(n/2 - 1) + O(n)$. So $L(n) \leq 2L(n/2 - 1) + 2O(n/2) + O(n) \leq 2L(n/2) + O(n)$. This leads to $O(n \log n)$ running time. This suggest it is likely that the expected running time of Quicksort is about $O(n \log n)$.

We now do a more rigorous analysis. Instead of always picking say the last element within the region, we now randomly choose the pivot. The expected running time of this randomized algorithm is analyzed in Section 7.4.2. Here are a list of major points:

1. The running time of Quicksort is proportional to the number of comparisons performed during the execution. This is because for each comparison, we spend constant time (to move elements around during partition, and partition is the main work involved in Quicksort).

2. If element $x$ and $y$ are compared, one of $x$ and $y$ must be a pivot sometime.

3. Moreover, if $x$ and $y$ are ever compared, they will be compared again. This is because the pivot will be fixed in positions and will not be involved later in comparisons. This allows us to define indicator variables for the events that two elements $x$ and $y$ are compared or not. And the frequently used linearity of expectation suggests that we now only need to know how to compute the probability of such event.

4. WLOG assume we have a permutation of $1, 2, ..., n$. The final point is, elements $i$ and $j$ (where $i < j$) are compared if $i$ or $j$ are the first to be chosen as pivot among elements: $i, i+1, ..., j$. Why? First note that if no pivot is chosen among $[i..j]$, elements from $i$ to $j$ will be within the same partition (why?). And if either $x$ or $y$ is chosen as pivot for the first time among $[i,j]$, $x$ and $y$ will be compared (why?). Then if any pivot $i < k < j$ is chosen as pivot, then $i$ and $j$ will be placed in different partition and will not be compared again. This leads to probability of $2/(j - i + 1)$. 