1 Quick Selection

1.1 Quick Selection Algorithm

The goal of Quick Selection algorithm is to find the k-th smallest element in an array. The idea of this algorithm is very similar to Quick Sort.

**Algorithm:** It first pick a random pivot number from array a, and then divides the array into two smaller sub-arrays: the left sub-array contains the numbers smaller than the pivot and the right sub-array contains the numbers larger than the pivot. After that, it counts the number of elements in the left sub-array. Assume there are i elements in the left sub-array, i.e., the pivot is the ith smallest element in the original array. If i > k, we recurse on the left sub-array, if i < k, we recurse on the right sub-array, and if i = k, we directly return the pivot. (Note that this algorithm only recurses on one side.)

**Example:** To selected the 5th smallest number in a list of numbers $a[] = 4, 2, 8, 6, 3, 1, 7, 5$, we first pick a random pivot number (say 3), then partition the array into two sub-arrays: $(2, 1, 4, 8, 6, 7, 5)$. After that, we recurse on the right sub-array, i.e., we want to find the 2nd smallest number in 4, 8, 6, 7, 5. We keep doing this until the algorithm returns, and the output of this algorithm should be 5.

**Worst Case:** Suppose we are extremely unlucky and always pick the smallest/largest element as the pivot, then the running time of this instance can be $\Theta(n^2)$.

1.2 Quick Selection Running Time Analysis

Let $X_n$ be the running time of quickselect on n numbers. Assume the elements in the list are distinct.

The expectation of the running time for the list of n items is thus:

$$E[X_n] = \sum E[x_n|pivot = i] \times Prob[pivot = i]$$

$$E[x_n|pivot = i] = \begin{cases} 
  \text{recurse on right part of } X_{n-i} & \text{if } i < k \\
  \text{output the pivot} & \text{if } i = k \\
  \text{recurse on the left part of } X_{i-1} + n & \text{if } i > k 
\end{cases}$$
\[ E[X_n] = \sum_{i=1}^{k-1} \frac{1}{n} (X_{n-i} + n) + \frac{1}{n} * n + \sum_{i=k+1}^{n} \frac{1}{n} (X_{i-1} + n) \]
\[ = n + \sum_{i=1}^{k-1} \frac{X_{n-i}}{n} + \sum_{i=\frac{n}{2}+1}^{n} \frac{X_{i-1}}{n} \]

While the worst case is \( k = \frac{n}{2} \).

\[ E[X_n] \leq n + \sum_{i=1}^{\frac{n}{2}-1} \frac{X_{n-i}}{n} + \sum_{i=\frac{n}{2}+1}^{n} \frac{X_{i-1}}{n} \]

\section{Las Vegas and Monte Carlo Algorithm}

\subsection{Las Vegas Algorithm}
1. Always output the correct answer
2. Running time is random

\subsection{Monte Carlo Algorithm}
1. Always run in a fixed amount of time
2. Result may be correct

\subsection{Monte Carlo Example}

\textbf{Problem:} Given a unit circle with its center at the origin, find the area of the circle.

\textbf{Solution:} Let \( X_i \) be the random variable which is defined as

\[ X_i = \begin{cases} 1 & \text{if } (x_i, y_i) \text{ in circle} \\ 0 & \text{if not} \end{cases} \]

The probability of having \( X_i = 1 \) is thus \( \text{Prob}[X_i = 1] = \text{area of circle} \). We also defines Count = \( \sum_{i=1}^{n} X_i \) and since the variances are bounded by \( \frac{4}{n} \), we have \( \text{Var}[X_i] = p(1-p) \leq \frac{1}{4} \) and \( \text{Var}[\text{Count}] \leq \frac{n}{4} \).

By Chebyshev inequality, we have \( \text{Prob}[|\text{count} - pn| > \sqrt{n}] \leq \frac{1}{4} \). When this does not happen, the error of our estimation in compare with the true area of the circle is bounded:

\[ |\frac{\text{Count}}{n} * 4 - P * 4| \leq \frac{4}{n}|\text{count} - pn| \leq \frac{4}{\sqrt{n}} \]

So if we choose \( n \geq \frac{16}{\epsilon^2} \), with probability \( \geq \frac{3}{4} \), \( |\frac{\text{Count}}{n} * 4 - P * 4| \leq \epsilon \)
3 Hashing

Set Problem: Maintain a dynamic subset of the universe \{0, 1, 2, ..., N-1\}.

Supported Operation:
1. insert
2. delete
3. look-up

Goal:
1. all three operations can be done in \(O(1)\)
2. space is proportional to the size of the set and independent of the size of the universe.

Design: Allocate an array \(a[0, 1, ..., m-1]\), \(m = \theta(\text{size of set})\). In which size of set equals to \(n\), which indicates the number of elements in the hashtable.

Hash function: \(f : \{0, 1, ..., m-1\} \rightarrow \{0, ..., m-1\}\) The function maps the element \(i\) to the location \(f(i)\) where it is stored.

Problem: However, in practice, there can be some \(x\) and \(y\) in the set where \(f(x) = f(y)\). We can such case a collision. The solution is to maintain a linked list at every \(a[i]\) location, and add all numbers with the same \(f(x)\) to the linked list.

However, by doing so, the running time for look-up operations become \(\theta(\text{length of list at } a[f(x)])\).

In worst cases, where the hash function allocate all the elements to a single location of the hash table, the worst case look-up performance can be as high as \(O(n)\).

4 Hash Families

A hash family is a set of hash functions in which each function \(f\) in the family \(f : \{0, 1, ..., m-1\} \rightarrow \{0, ..., m-1\}\).

4.1 Pairwise independent/Universal hash family

Say a family \(\mathcal{F}\) of hash functions is pairwise independent/universal if for every \(x \neq y \in \{0, 1, ..., N-1\}\), \(Pr_{f \in \mathcal{F}}[f(x) = f(y)] = \frac{1}{m}\).

Example: Suppose hash table already contains \(n\) numbers, \(x_1, ..., x_n\). Insert a new number \(y\) where \(y \notin \{x_1, x_2, ..., x_n\}\). What is the expected number of \(x_i\) that collides with \(y\).
Solution: Let $X_i = \begin{cases} 1 & f(x_i) = f(y) \\ 0 & f(x_i) \neq f(y) \end{cases}$

$$E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} Pr[X_i = 1]$$

$$= \sum_{i=1}^{n} Pr[f(x_i) = f(y)]$$

$$= \sum_{i=1}^{n} \frac{1}{m} = \frac{n}{m}$$