Problem 1

We saw in class that hashing with chaining, using universal hashing, leads to $O(1)$ expected time per operation for the dynamic dictionary problem. One downside of hashing with chaining is that it is not cache-friendly: computers are really fast at accessing sequential locations in memory, and slow at accessing a sequence of random locations (due to pre-fetching, amongst other optimizations). The reason hashing with chaining is not cache-friendly is that the nodes in the linked list we traverse during an operation may be at very different memory locations.

A common remedy to the above issue in practice is to not use hashing with chaining, but rather to use a scheme known as linear probing. In this scheme, we have an array $A$ of length $m$ and a hash function $h : [U] \rightarrow [m]$. Recall that we are maintaining a set $S \subseteq [U]$ with $|S| = n$ subject to query, insertion, and deletion. In linear probing, we perform insertion using the following algorithm:

```
Algorithm insert(k, v):
1. i ← h(k)
2. while A[i] is not NULL:
   if A[i].first = k:
      A[i].second ← v
      return
   else:
      i ← (i + 1) % m
3. A[i] ← (k, v)
```

The idea is that we try to insert $(k, v)$ at location $h(k)$, unless $A[h(k)]$ is already occupied by some other item. In such a case, we scan right in the array until we find an empty location and store $(k, v)$ there instead. Deletion and query are performed similarly.

Prove that if $m > 10n$ and $h$ is a uniformly random function mapping $[U]$ to $[m]$, then the expected time per insertion is $O(1)$. You may use the fact, without proof, that for any integers $1 \leq k \leq n$, $\binom{n}{k} \leq (en/k)^k$.

**Hint:** For a key $k$ being inserted, condition on how $h$ acts on $S \setminus \{k\}$. What is the expected runtime of the insertion of $k$ as a function of which cells are already occupied in the table?
Problem 2

In class we showed that when using hashing with chaining, the expected runtime of an operation is $O(1)$, but we could of course be unlucky and have some operations taking much longer than constant time. In particular, the worst case operation time is the length $L$ of the longest linked list. Given that the set $S$ of items we are maintaining is of size $n$, problem 4 of problem set 9 implies that with high probability, $L$ is at most $O\left(\log \frac{n}{\log \log n}\right)$ if $h$ is a uniformly random function from the set of all functions mapping $S$ to $[m]$ (see the pset9 solutions for details). As we saw in class, we prefer to use smaller hash families so that $h$ can be stored in memory using many fewer random bits.

Prove that if $h$ is drawn at random from a universal hash family $H$ with $m = n$, then $E L = O(\sqrt{n})$. This is of course not nearly as good as $O\left(\log \frac{n}{\log \log n}\right)$, but it is better than the trivial upper bound of $n^2$. You may use the fact, without proof, that if $\Phi : \mathbb{R} \to \mathbb{R}$ is a convex function and $X$ is a real-valued random variable, then $\Phi(E X) \leq E \Phi(X)$ (this is known as Jensen’s inequality). Hint: $\Phi(z) = z^2$ is convex.

Problem 3

In class we analyzed a toy model in which vertices $0, 1, \ldots, n+1$ are connected in a path, we start at some vertex $i$, and in every time step we move to a uniformly chosen random neighbor of our current location. We showed the expected number of steps to reach 0, starting at $i$, is exactly $n^2 - (n - i)^2$. We then showed via a coupling argument that the random walk 2SAT algorithm finds a satisfying assignment of a satisfiable formula in at most $O(n^2)$ time steps.

Use coupling to show the 3SAT random walk algorithm also finds a satisfying assignment of a satisfiable formula in a number of steps at most that of the corresponding toy model.

Problem 4

Hoeffding’s inequality states that when flipping $t$ independent coins each with probability $p$ of heads, the probability of seeing at least $(p + \varepsilon)t$ heads is at most $e^{-2\varepsilon^2 t}$.

Consider now the definition of the complexity class $\text{BPP}_p$: a language $L$ is in $\text{BPP}_p$ if there exists a polynomial-time verifier $V$ and constant $c > 0$ such that

- $\forall x \in L$, $\mathbb{P}_y(V(x, y) = 1) > 1 - p$
- $\forall x \notin L$, $\mathbb{P}_y(V(x, y) = 1) < p$

where $y$ is a uniformly random binary string of length at most $cn^c$ and $V$ runs in time at most $cn^c$, where $n$ is the length of $x$. We use $\text{BPP}$ to denote the class $\text{BPP}_{1/3}$.

(a) (7 points) Use Hoeffding’s inequality to show: $\forall k > 0$, $\text{BPP}_{1/3} = \text{BPP}_{1/2^{2^k}}$.

(b) (3 points) Suppose $f : \mathbb{N} \to \mathbb{N}$ is some function such that $f(n) = n^{\omega(1)}$. Show then that $\text{BPP}_{1/3} = \text{BPP}_{1/2^{2^{(n)}}}$ implies $\mathbb{P} = \text{BPP}$.