Problem 1.

Problem 2. We can turn a Van Emde Boas Queue into a tree that supports Find, Predecessor, and Successor for integers in the range $\{0, 1, 2, \dots, u-1\}$ as follows. Let a vEB tree T store the current minimum and maximum values in $T.min$ and $T.max$, and let it maintain an array of child vEB trees $T.children$ of length $\sqrt{u}$, where $T.children[i]$ points to a vEB tree that maintains the values $i\sqrt{u}, i\sqrt{u}+1, \dots, (i+1)\sqrt{u}-1$. Also, T maintains an auxiliary vEB tree $T.summ$ which keeps track of which children are non-empty (that is, i is in $T.summ$ iff $T.children[i]$ is non-empty).

Then for an integer $x \in \{0, 1, 2, \dots, u-1\}$, we define the operations as follows:

- **Find**(x, T):
  - Let $i = \lfloor x/\sqrt{u} \rfloor$ and $low = x \mod \sqrt{u}$.
  - If $T.summ[i] = 1$ and $T.children[i][low] = 1$, return True, else False.

Problem 3. (a) For each array, the probability that the k-th inserted item maps to an already-occupied bucket is $(k-1)/n^{1.5}$, so the overall collision probability (i.e. the probability that both buckets are already occupied) is $(k-1)^2/n^3$. If we insert $1, \ldots, n$ items sequentially, the cumulative number of expected collisions is given by

$$E\left[\sum_{i=1}^{n} I_{collision}\right] = \sum_{i=1}^{n} \frac{(i-1)^2}{n^3} = \frac{1}{n^3}\sum_{i=0}^{n-1} i^2 = \frac{n(n-1)(2n-1)}{6n^3} = \frac{2n^2-3n+1}{6n^2} = \boxed{O\left(\frac{1}{3}\right)}$$

(b) Suppose we are using a 2-universal hash family, and randomly selecting between two hash functions to map items to buckets. There is a $1/n^{1.5}$ probability that items i and j map to the same bucket for each array, so the probability that i maps to any already-occupied bucket in an array is $n/n^{1.5} = 1/\sqrt{n}$. Therefore, the overall collision probability (i.e. collisions in both arrays) for i is $(1/\sqrt{n})^2 = 1/n$. Then the total number of expected collisions is $n \cdot 1/n = O(1)$, which is the same outcome as in part (a). Since 2-universal hashing is efficient, we know that we can also implement bashing efficiently as well.

(c) We have shown that the expected number of collisions is $O(1)$, which means that the probability of the two hash functions being perfect is some probability $p < 1$. Then the probability of having to draw k times is p^k, and the sum of all possible k (i.e. the expected number of tries) is $O(n)$, producing a perfect bash function in linear time and quadratic space.

Problem 4.

Problem 5. 1. We can create an auxiliary graph G' with vertices representing each location at each time step; that is, each $v \in G$ corresponds to a set of vertices $v_0, v_1, v_2, \ldots \in G'$ where v_i represents the room at time step i. The edges of G' are constructed as follows: for each undirected edge (v, w) representing corridor/stairway e in G, we will add edges $(v_0, w_1), (v_1, w_2), \ldots$ and so on to represent people moving from v at time i to w at time $t+1$, as well as in reverse, that is $(w_0, v_1), (w_1, v_2), \ldots$ for people moving in the opposite direction, and set each of the weights of these edges to c_e (the capacity of the corridor/stairway). We will also add an edge from each vertex v_i to v_{i+1} represent people staying in the same location with functionally infinite capacity (i.e. the total number of people in the building). Thus, we have constructed a graph that models the movement of people throughout the building with respect to time. Then, if the last time step in G' is i, we can use max-flow to determine if all people can be evacuated from s to t in time i (i.e. flow(s) = flow(t)) and use binary search on the time range to find the minimum time for which evacuation is possible for everyone.

2. We can use the same technique in part (a) and add a super-source and super-sink vertex to the graph to represent the differing start/exit locations. For instance, we can add a super-source vertex S with edges to each of the starting locations s_i at time step 0 and capacities equal to the number of people in each location, and add another super-sink T with edges to all exit locations at any time step and infinite capacities. Then the flow from S to T represents the movement of people from the starting locations to the exits, and we can use max-flow to solve this variant as well.

3. We make a slight modification to the technique presented in part (a): for each edge (v, w) representing corridor e in G, let the transit time of e be y_e. We will add an edge from v_i to w_{i+y_e} with capacity y_e for all time steps i to represent people entering the corridor from v at time i and exiting at time $i + y_e$ from w (note that this still means more than y people can be in the corridor at a single time). We will similarly add edges from w_i to v_{i+y_e} with capacity y_e for all time steps i as well. Lastly, we retain the edges from v_i to v_{i+1} with infinite capacity. Thus, this new graph represents the movement of people through the building where corridors have different transit times, and we can use max-flow to determine the minimum time to evacuate everyone.