0.1 Problem 2 in homework 2

Let’s say we have an array of size \( n \) and \( T(n) \) is the expected number of lookups needed to find the smallest element by using the specified algorithm. For every \( 1 \leq i \leq n \), with probability \( 1/n \), we will pick array element \( i \). Picking array element \( i \) will leave us with the same problem, but an array of \( i - 1 \) elements rather than \( n \).

\[
T(n) = 1 + \frac{1}{n} T(0) + \frac{1}{n} T(1) + \ldots + \frac{1}{n} T(n-1)
\]

\[
= \frac{1}{n} \sum_{i=0}^{n-1} T(i) + 1
\]

\[
nT(n) = \sum_{i=0}^{n-1} T(i) + n
\]

To get rid of the summation, we write out the \( T(n - 1) \) term and then subtract it from the \( T(n) \) term.

\[
nT(n) = \sum_{i=0}^{n-1} T(i) + n
\]

\[
(n - 1)T(n - 1) = \sum_{i=0}^{n-2} T(i) + n - 1
\]

\[
nT(n) - (n - 1)T(n - 1) = T(n - 1) + 1
\]

\[
nT(n) = nT(n - 1) + 1
\]

\[
T(n) = T(n - 1) + \frac{1}{n}
\]

So all we have to do is solve the recurrence

\[
T(n) = T(n - 1) + \frac{1}{n}
\]

We do this in the standard way by telescoping. We get

\[
T(n) = T(0) + \sum_{i=1}^{n} \frac{1}{i}
\]

\( T(0) \) is of course 0, and the sum is \( O(\log n) \). Thus the expected number of lookups needed to find the smallest element is \( O(\log n) \).
0.2 Problem 3 in homework 2

(a) If there are \( j \) socks remaining in the drawer, then we have picked \( 2n - j \) without a match and thus there are \( 2n - j \) socks in the drawer that will match one of our socks on the bed. So the probability that we will pick a matching sock next is \( \frac{2n-j}{j} \). In this case, we will not need to pick any more because we’ve found a pair. If we don’t pick a matching sock (this will happen with probability \( \frac{2j-2n}{j} \)), then we’re in the same situation as before except there are \( j - 1 \) socks left in the drawer. So

\[
T(j) = 1 + \frac{2j-2n}{j}T(j-1)
\]

(b) If there are \( n \) socks left in the drawer, then there are also \( n \) socks on the bed that do not match, and therefore each sock on the bed has a matching sock in the drawer. Thus the next drawn sock will match some sock on the bed. So we will have to draw exactly one more sock and thus \( T(n) = 1 \).

(c) The program is essentially given to us by the recurrence:

```
main()
{
    n ← number of sock pairs
    T(2n,n)
}
```

```
T(j,n){
    if j = n return 1
    else return 1 + \( \frac{2j-2n}{j}T(j-1,n) \)
}
```

(d) Running the program for large values of \( n \), we see that \( T(n) \) is approximately \( \sqrt{\pi n} \).

0.3 Problem 2 in homework 3

Let the end vertices of the edge be \( u, v \). We need to check if there is a path between \( u \) and \( v \) which does not contain \( e \) since such a path implies a cycle containing \( e \). Doing this is easy as we can remove the edge \( e \) from the graph and do a DFS on the remaining graph starting from vertex \( u \) and checking if \( v \) is visited. So here is the pseudocode for the algorithm:

```
Check-Cycle(G, e)
    G' ← G − e // \( G - e \) denotes the graph \( G \) with the edge \( e \) removed
    Explore(G, u)
    If \( v \in Visited \)
        output("yes")
    else output("no")
```

Since we are doing a DFS on a graph smaller than \( G \) we get a running time of \( O(n + m) \).
0.4 Problem 4 in homework 3

From the theorem we get that there is at least one vertex with no incoming edges. Note that if there are more than one such vertex there cannot be a path covering all vertices. Let \( u \) be the unique vertex which has no incoming edges. Then \( u \) has to be the first vertex in any path covering all vertices. Now what is the next vertex in such a path? Consider the graph with \( u \) and its outgoing edges removed and let \( v \) be the unique vertex with no incoming edges, then \( v \) has to be the next vertex in the path. With this idea we can write the following algorithm:

\[
\text{Path-Cover}(G) \\
\quad \text{If } G \text{ contains only 1 vertex} \\
\quad \quad \text{output("path exists")} \\
\quad \text{If there are two vertices in } G \text{ with no incoming edges} \\
\quad \quad \text{output("no such path exists")} \\
\quad \text{Let } u \text{ be the vertex with no incoming edges} \\
\quad \text{Let } G' \text{ be the graph obtained by removing } u \text{ and its outgoing vertices from } G \\
\quad \text{Path-Cover}(G')
\]

**Running time** Finding the number of incoming edges of all vertices would take linear time. Finding a vertex with no incoming edges is also a linear time operation. The process of obtaining a new graph in each call to Path-Cover (and updating number of incoming edges for the remaining vertices) basically involves traversing all vertices and edges. So the total running time of the above algorithm is \( O(n + m) \).