1. (•) ***I understand the course policies.***

(a) What is the exact probability that node j is a common ancestor of node i and node k?

Solution: Let $A^j_{i,k}$ = [node j is a common ancestor of node i and node k].
Lemma 1: Node j is an ancestor of node i if and only if node j has the smallest priority among all nodes in $[i,j]$.

Proof: If $i = j$, then node j is an ancestor of node i because each node is an ancestor of itself. In this case, $[i,j]$ has only one node and so node jl has the smallest priority. If $i \neq j$, proved by the lecture notes, node j is a proper ancestor of node i if and only if node j has the smallest priority among all nodes in $[i, j]$. Similarly, we can prove that node j is an ancestor of node k if and only if node j has the smallest priority among all nodes in $[j,k]$. □

Lemma 2: Node j is a common ancestor of node i and node k if and only if node j has the smallest priority among all nodes in $[i,k]$.

Node j is a common ancestor of node i and node k if and only if node j is an ancestor of node i and an ancestor of node k. According to **(**Lemma 1), node j should have the smallest priority among all nodes in $[i,j]$ and $[j,k]$. Thus, node j has the smallest priority among all nodes in $[i,k]$. Since each node in $[i,k]$ is equally likely to have the smallest priority, the probability that node j has the smallest priority is:

$$A^j_{i,k} = \frac{1}{k-i+1}$$

■

(b) What is the exact expected length of the unique path from node i to node k in T?

Solution: If $i = k$, the expected path length is 0. Otherwise, there are three possible cases as shown in the figure.

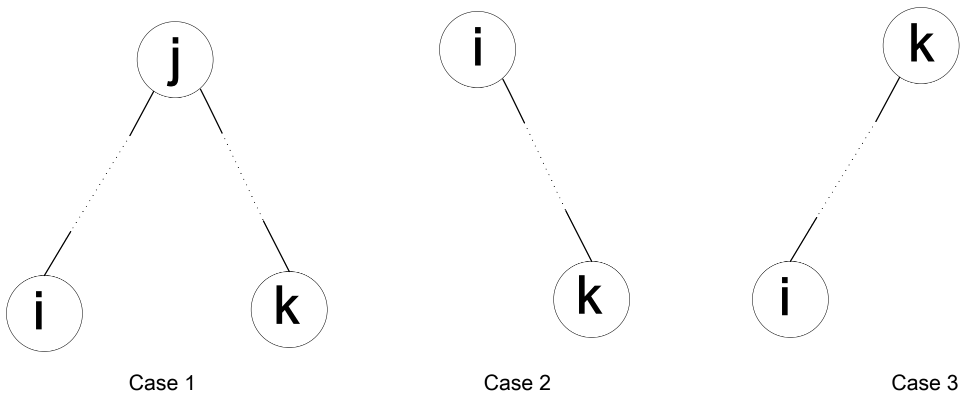


Figure 1. Three possible tree structures.

In case 1, one node j between (i,k) is a common ancestor of both node i and node k, and the nodes who are children of node j and ancestor of node i or node k should be counted in the path length. For each node t between $[1, i-1]$, node t is an ancestor of node i and node j is a common ancestor of node t, i and j if and only if node j has the smallest priority among all nodes between $[t, k]$ and node t has the smallest priority among all nodes between $[t, i]$. So the probability is:

$$\frac{1}{k-t+1} \cdot \frac{1}{i-t+1}$$

For each node t between $[i+1, j-1]$, node t is an ancestor of node i and node j is a common ancestor of node i, t, j if and only if node j has the smallest priority among all nodes between $[i, k]$ and node t has the smallest priority among all nodes between $[i, t]$. So the probability is:

$$\frac{1}{k-i+1} \cdot \frac{1}{t-i+1}$$

For each node t between $[j+1, k-1]$, node t is an ancestor of node k and node j is a common ancestor of node i, t, k if and only if node j has the smallest priority among all nodes between $[i, k]$ and node t has the smallest priority among all nodes between $[t, k]$. So the probability is:

$$\frac{1}{k-i+1} \cdot \frac{1}{k-t+1}$$

For each node t between $[k+1, n]$, node t is an ancestor of node k and node j is a common ancestor of node i, k, t if and only if node j has the smallest priority among all nodes between $[i, t]$ and node t has the smallest priority among all nodes between $[k, t]$. So the probability is:

$$\frac{1}{t-i+1} \cdot \frac{1}{t-k+1}$$

Besides that, node j must be counted in the path as it it the common ancestor of node i and node k. So in case 1, the expected number of nodes between node i and node j multiplied by its probability is:

$$\begin{aligned} P_1 \cdot E_1 \quad = \quad & \sum_{j=i+1}^{k-1} \Big(\sum_{t=1}^{i-1} \frac{1}{k-t+1} \cdot \frac{1}{i-t+1} + \sum_{t=i+1}^{j-1} \frac{1}{k-i+1} \cdot \frac{1}{t-i+1} + \\ & \sum_{t=j+1}^{k-1} \frac{1}{k-i+1} \cdot \frac{1}{k-t+1} + \sum_{t=k+1}^{n} \frac{1}{t-i+1} \cdot \frac{1}{t-k+1} + \frac{1}{k-i+1} \Big) \end{aligned}$$

In case 2, node i is an ancestor of node k, and the nodes who are children of node i and ancestor of node k should be counted in the path length.

For each node t between $[1, i-1]$, it cannot be an ancestor of node k and a child of node i.

For each node t between $[i+1, k-1]$, node t is an ancestor of node k and a child of node i if and only if node i has the smallest priority among all nodes between $[i, k]$ and node t has the smallest priority among all nodes between $[t, k]$. So the probability is:

$$\frac{1}{k-i+1} \cdot \frac{1}{k-t+1}$$

For each node t between $[k+1, n]$, node t is an ancestor of node k and a child of node i if and only if node i has the smallest priority among all nodes between $[i, t]$ and node t has the smallest priority among all nodes between $[k, t]$. So the probability is:

$$\frac{1}{t-i+1} \cdot \frac{1}{t-k+1}$$

In case 2, the expected number of nodes between node i and node j multiplied by its probability is:

$$P_2 \cdot E_2 = \sum_{t=i+1}^{k-1} \frac{1}{k-i+1} \cdot \frac{1}{k-t+1} + \sum_{t=k+1}^{n} \frac{1}{t-i+1} \cdot \frac{1}{t-k+1}$$

In case 3, node k is an ancestor of node i and the nodes who are children of node k and ancestor of node i should be counted in the path length.

For each node t between $[1, i-1]$, node t is an ancestor of node i and a child of node k if and only if node k has the smallest priority among all nodes between $[t, k]$ and node t has the smallest priority among all nodes between $[t, i]$. So the probability is:

$$\frac{1}{k-t+1} \cdot \frac{1}{i-t+1}$$

For each node t between $[i+1, k-1]$, node t is an ancestor of node i and a child of node k if and only if node k has the smallest priority among all nodes between $[i, k]$ and node t has the smallest priority among all nodes between $[i, t]$. So the probability is:

$$\frac{1}{k-i+1} \cdot \frac{1}{t-i+1}$$

In case 3, the expected number of nodes between node i and node j multiplied by its probability is:

$$P_3 \cdot E_3 = \sum_{t=1}^{i-1} \frac{1}{k-t+1} \cdot \frac{1}{i-t+1} + \sum_{t=i+1}^{k-1} \frac{1}{k-i+1} \cdot \frac{1}{t-i+1}$$

So the exact expected length of the unique path from node i to node k is:

$$\begin{aligned}
E &= P_1E_1 + P_2E_2 + P_3E_3 \\
&= \sum_{j=i}^{k}\Big(\sum_{t=1}^{i-1} \frac{1}{k-t+1} \cdot \frac{1}{i-t+1} + \sum_{t=i+1}^{j-1} \frac{1}{k-i+1} * \frac{1}{t-i+1} + \\
&\quad \sum_{t=j+1}^{k-1} \frac{1}{k-i+1} * \frac{1}{k-t+1} + \sum_{t=k+1}^{n} \frac{1}{t-i+1} * \frac{1}{t-k+1}\Big) + \frac{k-i-1}{k-i+1} \\
&= \sum_{j=i}^{k}\Big(\frac{1}{k-i}\Big(\sum_{m=2}^{i} \frac{1}{m} - \sum_{n=k-2+i}^{k} \frac{1}{n}\Big) + \frac{1}{k-i+1}\Big(\sum_{m=2}^{j-i} \frac{1}{m} + \sum_{n=2}^{k-j} \frac{1}{n}\Big) + \\
&\quad \frac{1}{k-i}\Big(\sum_{m=2}^{n-k+1} \frac{1}{m} - \sum_{n=k-i+2}^{n-i+1} \frac{1}{n}\Big)\Big) + \frac{k-i-1}{k-i+1} \\
&= H_i + H_{k+i-3} + H_{n-k+1} + H_{k-i+1} - H_k - H_{n-i+1} - 2 + \\
&\quad \sum_{j=i}^{k} \frac{H_{j-i} + H_{k-j} - 2}{k-i+1} + \frac{k-i-1}{k-i+1}
\end{aligned}$$

■

2. (•) ***I understand the course policies.***

   (a) Describe and analyze an algorithm to solve the following problem in O(n) time: Given indices i, j, i', j' as input, compute the number of elements of M smaller than $M[i, j]$ and larger than $M[i', j']$.

Solution: We say an element x is valid if $M[i', j'] < x < M[i, j]$. For each column, we compute an array $T[1..n]$ and $B[1..n]$, where $T[c]$ and $B[c]$ denotes the smallest and largest element in column c that is valid, respectively. Then the number of elements *total* can be computed as:

$$total = \sum_{c=1}^{n}(B[c] - T[c] + 1)[B[c] \geq T[c]]$$

Apparently, if $M[i', j'] \geq M[i, j]$, *total*=0. We assume that $M[i', j'] < M[i, j]$. We can compute $T[1..n]$ and $B[1..n]$ using the following strategy. For $T[1..n]$, starting from $M[i', j']$, we can compute $T[j']$ by looking down for the first element in column j' that is greater than $M[i', j']$. Then, we move left by one column and compute $T[j' - 1]$ using the same strategy. In the worst case, we start from $M[1, n]$ and end at $M[n, 1]$. The Manhattan distance of this route is at most $2n$.Hence this takes $O(n)$ time. We can similarly compute $T[j' + 1..n']$ by looking up and right. Similar action applies to computing $B[1..n]$. The pseudo code of computing $T[1..n]$ and $B[1..n]$ is given as follows. Note that all of the following algorithms are given regarding matrix $M[1..n, 1..n]$. Thus we omit using it as the parameter.

```
TopLeft($i', j'$):
    for $c \leftarrow j'$ down to 1
        $T[c] \leftarrow +\infty$
    $r \leftarrow i'$, $c \leftarrow j'$, $stop = false$
    while $c \geq 1$ and not $stop$
        while $r \leq n$ and $M[r, c] \leq M[i', j']$
            $r \leftarrow r + 1$
        if $r \leq n$
            $T[c] \leftarrow r$
        else
            $stop \leftarrow true$
        $c \leftarrow c - 1$
return $T[1..j']$
```

```
TopRight($i', j'$):
    for $c \leftarrow j' + 1$ to $n$
        $T[c] \leftarrow +\infty$
    $r \leftarrow i'$, $c \leftarrow j' + 1$, $stop = false$
    while $c \leq n$ and not $stop$
        while $r \geq 1$ and $M[r, c] > M[i', j']$
            $r \leftarrow r - 1$
        if $r \geq 1$
            $T[c] \leftarrow r + 1$
        else if $M[r, c] > M[i', j']$ ⟨⟨r=0⟩⟩
            $T[c] \leftarrow 1$
        else
            $stop \leftarrow true$
        $c \leftarrow c + 1$
return $T[j' + 1..n]$
```

```
BottomLeft($M[1..n, 1..n], i, j$):
    for $c \leftarrow j$ down to 1
        $B[c] \leftarrow -\infty$
    $r \leftarrow i$, $c \leftarrow j$, $stop = false$
    while $c \geq 1$ and not $stop$
        while $r \leq n$ and $M[r, c] \leq M[i, j]$
            $r \leftarrow r + 1$
        if $r \leq n$
            $B[c] \leftarrow r - 1$
        else if $M[r, c] < M[i, j]$ ⟨⟨r=n+1⟩⟩
            $B[c] \leftarrow n$
        else
            $stop \leftarrow true$
        $c \leftarrow c - 1$
return $B[1..j]$
```

```
BottomRight($i, j$):
    for $c \leftarrow j + 1$ to $n$
        $B[c] \leftarrow -\infty$
    $r \leftarrow i$, $c \leftarrow j + 1$, $stop = false$
    while $c \geq 1$ and not $stop$
        while $r \geq 1$ and $M[r, c] >= M[i, j]$
            $r \leftarrow r - 1$
        if $r \geq 1$
            $B[c] \leftarrow r$
        else
            $stop \leftarrow true$
        $c \leftarrow c + 1$
return $B[j + 1..n]$
```

Finally, *total* can be computed as follows:

```
COMPUTETB($i, j, i', j'$):
    $T[1..j'] \leftarrow$ TOPLEFT($i', j'$)
    $T[j'+1..n] \leftarrow$ TOPRIGHT($i', j'$)
    $B[1..j] \leftarrow$ BOTTOMLEFT($i, j$)
    $B[j+1..n] \leftarrow$ BOTTOMRIGHT($i, j$)
    return $T[1..n]$, $B[1..n]$
```

```
NUMELEMENTS($i, j, i', j'$):
    if $M[i', j'] \geq M[i, j]$
        return 0
    $T[1..n], B[1..n] \leftarrow$ COMPUTETB($i, j, i', j'$)
    $total \leftarrow 0$
    for $c \leftarrow 1$ to $n$
        if $B[c] \geq T[c]$
            $total \leftarrow total + (B[c] - T[c] + 1)$
    return $total$
```

This is illustrated in Figure 2 ■

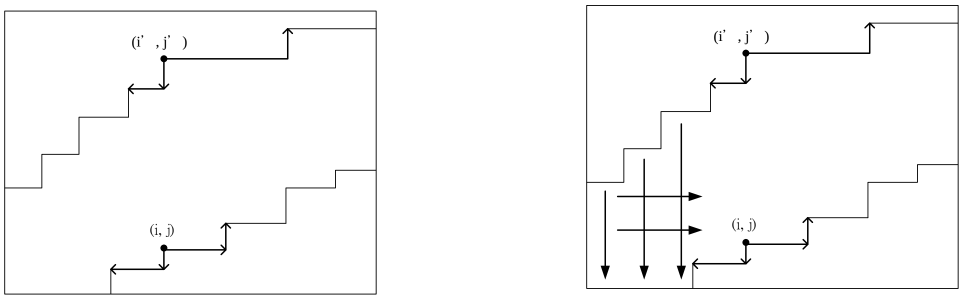


Figure 2. Illustration of algorithm and ordering.

(b) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, return an element of M chosen uniformly at random from the elements smaller than $M[i, j]$ and larger than $M[i', j']$. Assume the requested range is always non-empty.

Solution: We first compute the number of elements s between $M[i', j']$ and $M[i, j]$ using the algorithm in previous problem. Then, we label them as $\{a_1, a_2, ..., a_s\}$ using column as first order and row as second, as illustrated in Figure 2. Next, we draw a lot from $[1..s]$ and return the corresponding element. Assume that we have a function RANDOM(S) that can return a number uniformly at random from 1 to s in constant time. A pseudo code is given as follows:

```
RANDOMELEM($i, j, i', j'$):
    $T[1..n], B[1..n] \leftarrow$ COMPUTETB($i, j, i', j'$)
    $x \leftarrow$ RANDOM(S)
    for $c \leftarrow 1$ to $n$
        $t \leftarrow B[c] - T[c] + 1$
        if $x \leq t$
            return $M[T[c] + x - 1, c]$
        else
            $x \leftarrow x - t$
```

This algorithm clearly runs in $O(n)$ time. ■

(c) Describe and analyze a randomized algorithm to compute the median element of M in $O(n\log n)$ expected time.

Solution: We describe an iterative algorithm that computes the median in the matrix as follows. At each iteration, we have two bounds l and r. At first, l and r are set to $M[1,1]$ and $M[n,n]$. Then we use the algorithm in problem (2) to choose an element $k = M[x,y]$ that is within range $l < k < r$. Then, we compute the number of elements *num* within range $M[1,1]$ and $M[x,y]$. There are three cases:

i. If *num* is exactly $n^2/2$, then we have found the median and just return it.
ii. If *num* is smaller than $n^2/2$, then we know the median is within range $M[x,y]$ and r. We set $l = M[x,y]$ and iterate.
iii. If *num* is larger than $n^2/2$, then we know the median is within range l and $M[x,y]$. We set $r = M[x,y]$ and iterate.

The pseudo code is given as follows:

```
RANDOMIZEDMEDIAN($M[1..n][1..n]$):
  $(i',j') \leftarrow (1,1)$ ⟨⟨l⟩⟩
  $(i,j) \leftarrow (n,n)$ ⟨⟨r⟩⟩
  $stop \leftarrow false$
  while not $stop$
      $k \leftarrow$ RANDOMELEM($i,j,i',j'$)
      $(x,y) \leftarrow$ index of $k$
      $num \leftarrow$ NUMELEMENTS($x,y,1,1$)
      if $num = \lfloor n^2/2 \rfloor$
          $stop \leftarrow true$
      else if $num < \lfloor n^2/2 \rfloor$
          $(i',j') \leftarrow (x,y)$
      else
          $(i,j) \leftarrow (x,y)$
  return $k$
```

Analysis: We claim that this algorithm finishes after $log(n)$ iteration.

Proof: At i-th iteration, let s_i and t_i be the number of elements that within the bound from l and r to median m, respectively. Since we will choose an element from range s_i to t_i at each time, either s_i or t_i is subject to change. Hence, the expect value of s_i can be computed as follows:

$$E[s_i] = s_{i-1} \cdot \frac{t_{i-1}}{s_{i-1}+t_{i-1}} + \sum_{1}^{s_{i-1}-1} \cdot \frac{1}{s_{i-1}+t_{i-1}}$$

Similarly we can get the expect value of t_i:

$$E[t_i] = t_{i-1} \cdot \frac{s_{i-1}}{s_{i-1}+t_{i-1}} + \sum_{1}^{t_{i-1}-1} \cdot \frac{1}{s_{i-1}+t_{i-1}}$$

Then we have:

$$\begin{aligned} E[s_i+t_i] &= E[s_i]+E[t_i] \\ &= \frac{2s_{i-1}t_{i-1}+s_{i-1}^2/2+t_{i-1}^2/2+s_{i-1}+t_{i-1}}{s_{i-1}+t_{i-1}} \\ &\leq \frac{(s_{i-1}+t_{i-1})^2}{2} \end{aligned}$$

Thus:

$$\frac{E[s_i + t_i]}{s_{i-1} + t_{i-1}} \leq \frac{s_{i-1} + t_{i-1}}{2}$$

□

Thus, the expect value of remaining elements within range l to r at i-th round is no more than half of the previous round. Hence, at most after $\log(n)$ rounds, the remaining elements is 0. The RandomElem and NumElements in the algorithm takes $O(n)$ time. Hence, the overall runtime is $O(n \log(n))$. ■

3.

(●) ***I understand the course policies.***

Solution: Suppose the weight of the first edge is k, it means the smallest weight among all the n edges connected to the first vertex is k. So the weight of one edge falls into region dk, and the weights of all other n-1 edges fall into the region $[k, 1]$. So the probability Pr[The weight of the first edge is k] $= \frac{dk}{1} * (\frac{1-k}{1})^{n-1}$. As the first vertex is chosen randomly and the weights are distributed uniformly at random from the real interval $[0, 1]$, The expected weight of the first edge will be:
$\int_0^1 (1-k)^{n-1} k dk = \frac{1}{n} * \frac{1}{n+1}$
The $ith(1 \leq i \leq n)$ edge has n-i+1 choices, so the expected weight of the ith edge is:
$\int_0^1 (1-k)^{n-i} k dk = \frac{1}{n-i+1} * \frac{1}{n-i+2}$
Thus, the expected weight of the resulting Hamiltonian cycle is:
$\sum_{i=1}^{n} \frac{1}{n-i+1} * \frac{1}{n-i+2} = \sum_{i=1}^{n} \frac{1}{i} * \frac{1}{i+1} = \sum_{i=1}^{n} (\frac{1}{i} - \frac{1}{i+1}) = \frac{n}{n+1}$ ■

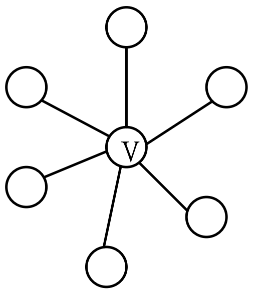
4. (•) ***I understand the course policies.***

(a) Prove that VertexCover can return a vertex cover that is $\Omega(n)$ times larger than the smallest vertex cover.

Solution: Consider a graph that has n vertices. We label the vertices as $0, 1, \dots, n-1$. Each vertex $i \neq 0$ is connected to vertex 0. Then, the optimal solution is 1. However, in the worst case, VertexCover may pick an edge $(i, 0)$ and always choose the endpoint i. Then, the algorithm will return $n-1$. Hence, it is $\Omega(n)$ times larger than the smallest vertex cover. ■

(b) Prove that the expected size of the vertex cover returned by RandomVertexCover is at most $2 \cdot \text{OPT}$, where OPT is the size of the smallest vertex cover.

Solution: Proof: First, consider a graph G^* with n vertices and $n-1$ edges, where v is a vertex with edges connecting other vertices. The illustration of G^* with 7 vertices and 6 edges is as follows:



Claim: We show that for G^*, the expected size of the vertex cover returned by RandomVertexCover is at most 2.

Proof of Claim: Let $d = n-1 = \text{degree}(v)$. $\Pr[v \text{ is added into } C \text{ in } i\text{-th iteration}] = (1/2)^i$.

$$\text{E[size of vertex cover]} = \sum_{i=1}^{d} \frac{1}{2}^i \cdot i = d\frac{1}{2}^d + (d-1)\frac{1}{2}^{d-1} + \cdots + 2\frac{1}{2}^2 + \frac{1}{2}$$

Then,

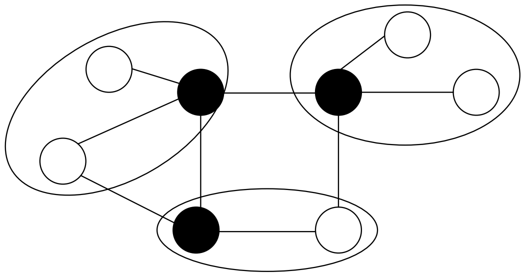
$$2\text{E[size of vertex cover]} = d\frac{1}{2}^{d-1} + \cdots + 2\frac{1}{2} + 1$$

So

$$\text{E[size of vertex cover]} = 1 + \frac{1}{2} + \cdots + \frac{1}{2}^d \leq 2$$

Now we consider a regular graph G, where OPT is the size of the smallest vertex cover, and S is the set of the veritces in the smallest vertex cover. Then we can divide G into OPT clusters. In the beginning, every vertex v in S constructs a cluster containing itself. Then If a un-clustered vertex is covered by only one vertex in S, then they are clustered together. If a un-clustered vertex is covered by more than one vertex in S, the vertex can be clustered with either of the vertices in S that cover it. The following figure shows an example for this clustering. Black nodes represent the minimum vertex cover. A eclipse represents a cluster, so there are three clusters in the example.

So in each cluster, there is a vertex v has edges connecting all other vertices in the cluster because it covers all other vertices. So the graph constructed by the cluster is a G^* which



is mentioned above. Then the expected size of the vertex cover for the cluster computed by RandomVertexCover is at most 2. Since there are OPT clusters, the expected size of the vertex cover for G computed by RandomVertexCover is at most 2·OPT.

□

■

(c) Prove that the expected weight of the vertex cover returned by RANDOMWEIGHTEDVERTEXCOVER is at most 2OPT,where OPT is the weight of the minimum-weightvertexcover.A correct answer to this part automatically earns full credit for part.

Solution: We don't know. ■

5. (•) ***I understand the course policies.***

(a) For a particular bin b, the probability Pr[Bin b has exactly k balls] = $C_n^k(\frac{1}{m})^k(\frac{m-1}{m})^{n-k}$. As there are m bins in total, the expected number of bins that contain exactly k balls is:
$mC_n^k(\frac{1}{m})^k(\frac{m-1}{m})^{n-k} = m * \frac{n!}{k!(n-k)!} * (\frac{1}{m})^k * (\frac{m-1}{m})^{n-k}$

(b) Suppose after the ith round, B_i bins are left. From the last question we know:
$\frac{B_i}{B_{i-1}} = (1 - \frac{1}{B_{i-1}})^n \le e^{-\frac{n}{-B_{i-1}}}, (B_0 = n)$
From the inductive equation above, we get:
$B_1 = ne^{-1}$
$B_2 = ne^{-(1+e)}$
$B_3 = ne^{-(1+e+e^{1+e})}$
...
So after kth round, the dominate factor in B_k will be:

For this analysis we see, the experiment will end after $O(log * n)$ rounds.

(c) For any particular bin B, it remains after k rounds if and only if in every of the k rounds, all the n balls are thrown to the other bins rather than B. So the probability that Pr[B remains after k rounds] is:
$Pr = \prod_{i=0}^{k-1}(1 - \frac{1}{B_i})^n$
We know that $B_i \le n$ for any $i \in [0, k-1]$. So:
$Pr \le (1 - \frac{1}{n})^{nk} \le (\frac{1}{e})^k$
So after $O(log * n)$ rounds, the probability that any articular bin B can be left is $\frac{1}{n}$. So with high probability, no bins can be left after $O(log * n)$ rounds and thus the experiment ends after $O(log * n)$ rounds.

(d) We don't know.

(e) We don't know.