Algorithms HW3

Greyson Brothers

March 7, 2023

Statement of Integrity: I, Greyson Brothers, attempted to answer each question honestly and to the best of my abilities. I cited any and all help that I received in completing this assignment.

1 Problem

The following problem is known as the Dutch Flag Problem. In this problem, the task is to rearrange an array of characters R, W , and B (for red, white, and blue, which are the colors of the Dutch national flag) so that all the R's come first, all the W 's come next, and all the B's come last. Design a linear, in-place algorithm that solves this problem.

For this algorithm we will perform 2 iterations. First, we will loop through the list of characters and move all red elements to the front of the list insertion sort style. Then, we loop through the list in reverse, moving all blue elements to the back of the list in the same fashion. When this is completed, the list will be sorted R-W-B, taking 2n operations (linear).

Algorithm 1 Dutch Flag Sort

```
array ←                                                    ▷ len(n)
j ← 0                                                      ▷ 1
for i = 0; i < len(array); i++ do                          ▷ n
    if array[i] == "R" then                                ▷ n
        temp = array[i]                                    ▷ 0 → n
        array[i] = array[j]                                ▷ 0 → n
        array[j] = array[i]                                ▷ 0 → n
    end if
    if array[j] == ""R" then                               ▷ n
        j++                                                ▷ 0 → n
    end if
end for
j ← len(array) − 1                                         ▷ 1
for i = len(array)-1; i >= 0; i-- do                       ▷ n
    if array[i] == "B" then                                ▷ n
        temp = array[i]                                    ▷ 0 → n
        array[i] = array[j]                                ▷ 0 → n
        array[j] = array[i]                                ▷ 0 → n
    end if
    if array[j] == "B" then                                ▷ n
        j--                                                ▷ 0 → n
    end if
end for
```

Note the conditional logic in the if statements can execute any number of times from 0 to n depending on the input array. For analysis we take the worst case n for all statements. Also, in practice, the worst case in one loop (i.e. an array with all R's) is the best case for the other loop.

$$T(n) = (1+n+n+n+n+n+n+n)+(1+n+n+n+n+n+n+n) = 14n+2 = O(n)$$

2 Problem

Give an O(n lg k)-time algorithm to merge k sorted lists into one sorted list, where n is the total number of elements in all the input lists. (Hints: Use a heap for k-way merging.)

Here we utilize a min heap for k-way merge. We loop n times, each time popping the min node from the heap and adding it to the output sorted array. The heap will contain at most k elements at all times, with those k elements being the minimum value of each of our k sorted subarrays. We start with the first value in each subarray. When an node is popped, we insert the next minimum element in that subarray. Each node in the heap contains a subarray and a current index, whereby the value of the node is the subarray at that index. Of where we are in each subarray for pops and inserts.

Heapify down takes ln n time. Inserting and popping the min take constant time.

Algorithm 2 K-Way Merge

```
arrays ←                                          ▷ k arrays of len(n)
output ←                                          ▷ len(kn)
heap ←                      ▷ O(1) insert, O(1) pop, O(ln n) heapify
for i = 0; i < k; i++ do                                         ▷ k
    heap.insert(subarray=arrays[i], idx=0)                       ▷ k
    heap.heapify()                                          ▷ $k$ ln $k$
end for
$j \leftarrow 0$                                                   ▷ 1
while heap not empty() do                                        ▷ n
    subarray, idx = heap.pop()                                   ▷ n
    heap.heapify()                                           ▷  ln $k$
    output[j] = subarray[idx]                                    ▷ n
    j++                                                          ▷ n
    idx++                                                        ▷ n
    if idx < n then                                              ▷ n
        heap.insert(subarray, idx)                        ▷ $0 \rightarrow n$
        heap.heapify()                               ▷ $0 \rightarrow n$ ln $k$
    end if
end while
```

$$T(n) = (k + k + k\ln k) + (1 + n + n + n\ln k + n + n + n + n + n + n\ln k)$$

$$= O(k\ln k + n\ln k) = O(n\ln k)$$

3 Problem

3.1 a)

Given the random process described in the homework, what is the expected number of incoming links to node v_j in the resulting network? Give an exact formula in terms of n and j, and also try to express this quantity asymptotically (via an expression without large summations) using $\theta(\cdot)$ notation.

Let X be the random variable whose value equals the number of incoming links our node v_j has. Let $\mathbf{k}$ be the event such that node new node v_k links to existing node v_j for $k > j$. Let I be the indicator random variable such that

$$X_{\mathbf{k}} = I(\mathbf{k}) = \begin{cases} 1 & \text{if } v_k \text{ links to node } v_j \\ 0 & \text{otherwise} \end{cases}$$

where $X_{\mathbf{k}}$ is the random variable associated with event $\mathbf{k}$. It follows that $X = X_1 + X_2 + ... + X_n$. The probability of $\mathbf{k}$ occurring is then:

$$P(X_{\mathbf{k}}) = P(I(\mathbf{k})) = \begin{cases} \frac{1}{k} & \text{if } I(\mathbf{k}) = 1 \\ \frac{k-1}{k} & \text{if } I(\mathbf{k}) = 0 \end{cases}$$

We can compute the expected number of times a new node links to v_j via $E[X_{\mathbf{k}}]$, where $X_{\mathbf{k}} = I(\mathbf{k})$.

$$\begin{aligned} E[X_{\mathbf{k}}] &= E[I(\mathbf{k})] \\ &= 1 \cdot P(I(\mathbf{k}) = 1) + 0 \cdot P(I(\mathbf{k}) = 0) \\ &= P(I(\mathbf{k}) = 1) \\ &= \frac{1}{k} \end{aligned} \tag{1}$$

Given a set of nodes $v_1, ...v_n$, we want to know the expected value $E[X]$ of the number of incoming links to an arbitrary node v_j. By the linearity of expectation, we have the following:

$$\begin{aligned} E[X] &= E[X_1 + ... + X_n] \\ &= E[X_1] + ... + E[X_n] \\ &= 0 + ... + 0 + E[X_{j+1}] + ... + E[X_n] \\ &= 0 + ... + 0 + \frac{1}{j+1} + ... + \frac{1}{n} \\ &= \sum_{i=j+1}^{n} \frac{1}{i} \\ &= f(n, j) \end{aligned} \tag{2}$$

Rewriting the sum using A.7 from the CLRS textbook we are able to bound it as follows:

$$\begin{aligned} f(n,j) &= \sum_{i=j+1}^{n} \frac{1}{i} \\ &= \sum_{i=1}^{n} \frac{1}{i} - \sum_{i=1}^{j} \frac{1}{i} \\ &= ln(n) + O(1) - ln(j) - O(1) \\ &= \Theta(ln(\frac{n}{j})) \end{aligned} \tag{3}$$

Thus we have our theta bound on our summation $f(n,j)$.

3.2 b)

Part (a) makes precise a sense in which the nodes that arrive early carry an "unfair" share of connections in the network. Another way to quantify the imbalance is to observe that, in a run of this random process, we expect many nodes to end up with no incoming links. Give a formula for the expected number of nodes with no incoming links in a network grown randomly according to this model.

Let Z be a random variable equal to the number of nodes with no incoming links at the end of our random process. Let $I(m)$ be the indicator random variable such that:

$$Z_m = I(\mathbf{m}) = \begin{cases} 1 & \text{if } v_m \text{ has no incoming links} \\ 0 & \text{otherwise} \end{cases}$$

where Z_m is the random variable associated with event $\mathbf{m}$. It follows that $Z = Z_1 + Z_2 + ... + Z_n$. We can more easily compute the probability of $\mathbf{m}$ occurring by using the compliment and the result of 3(a) as follows:

$$\begin{aligned} P(Z_{\mathbf{m}}) &= P(I(\mathbf{m})) \\ &= 1 - P(I(\mathbf{m})^c) \\ &= 1 - \left(\frac{1}{m} + \frac{1}{m+1} + ... + \frac{1}{n}\right) \\ &= 1 - \ln\frac{n}{m} \end{aligned} \tag{4}$$

We can then find the expected value $E[Z_m]$ using some linearity of expectation and compliment tricks as follows:

$$\begin{aligned}
E[Z_m] &= E[I(m)] \\
&= E[1 - I(m)^c] \\
&= 1 - E[I(m)^c] \\
&= 1 - (1 \cdot P(I(m)^c) = 1) + 0 \cdot P(I(m)^c) = 0)) \\
&= 1 - \ln \frac{n}{m}
\end{aligned} \tag{5}$$

By using linearity of expectation once more we can find $E[Z]$:

$$\begin{aligned}
E[Z] &= E[Z_1 + ... + Z_n] \\
&= E[Z_1] + ... + E[Z_n] \\
&= (1 - \ln \frac{n}{1}) + (1 - \ln \frac{n}{2}) + ... + (1 - \ln \frac{n}{n}) \\
&= n - (\ln n - \ln 1) - (\ln n - \ln 2) - ... - (\ln n - \ln n) \\
&= n - n \ln n + \ln 1 + \ln 2 + ... + \ln n \\
&= n - n \ln n + \ln n! \\
&= f(n)
\end{aligned} \tag{6}$$

thus giving us the expectation of the number of nodes with no incoming links as a function of n.