15-451/651 Assignment 2

Mohit Singh, Majeed Thaika, Vy Nguyen
mohitdes — mthaika — vyn

February 8, 2018

1: A Median of Sorts

(a)

Input: sorted arrays A and B of n elements each and all are distinct
Output: median of both arrays- i.e., the nth smallest element in the union of A and B
Algorithm:

```
findMedian(A, B) {
if ($|A| = 0$) return $B[0]$
else if ($|B| = 0$) return $A[0]$
else
  $m_A = A(floor(|A|/2))$
  $m_B = B(floor(|B|/2))$
  if ($A[m_A] < B[m_B]$)
    if ($|A|$ is even)
      if ($|B|$ is even) findMedian($A[m_A + 1, ..., |A|], B[1, ...m_B]$)
      else findMedian($A[m_A + 1, ..., |A|], B[1, ...m_B - 1]$)
    else
      if ($|B|$ is even) findMedian($A[m_A, ..., |A|], B[1, ...m_B]$)
      else findMedian($A[m_A, ..., |A|], B[1, ...m_B - 1]$)
  else
    if ($|A|$ is even)
      if ($|B|$ is even) findMedian($A[1, ..., m_A], B[m_B + 1, ...|B|]$)
      else findMedian($A[1, ..., m_A - 1], B[m_B + 1, ...|B|]$)
    else
      if ($|B|$ is even) findMedian($A[1, ..., m_A], B[m_B, ...|B|]$)
      else findMedian($A[1, ..., m_A - 1], B[m_B, ...|B|]$)
}
```

Runtime Analysis:
We need to consider two cases: If $2n = |A| + |B| = 2^k$ and $2n \neq 2^k$ for some integer k. If $2n = 2^k$, then the starting and resultant intermediate lists are always even, and we can see for the algorithm that size of the list exactly halves at each step, so we get the following recurrence, where T(1)=0:

$$
\begin{align}
T(2n) &= T(n) + 1 \tag{1}\\
&= T(\frac{n}{2}) + 1 + 1 \tag{2}\\
&= T(\frac{n}{2^k}) + \sum_{i=0}^{k+1} 1 \tag{3}\\
&= T(\frac{n}{2^{log_2(n)}}) + \sum_{i=0}^{log_2(n)+1} 1 \tag{4}\\
&= T(1) + \sum_{i=0}^{log_2(n)+1} 1 \tag{5}\\
&= log_2(n) + 1 \tag{6}\\
& \tag{7}
\end{align}
$$

The second case is when $2n \neq 2^k$ for some integer k. Then, find smallest k s.t. k is a power of 2 and $2n < 2^k$ (Notice that $2n > 2^{k-1}$). We can use the exact recurrence step as above for starting arrays of size 2^k and 2^{k-1}, to get worst case number of comparisons k and k-1 respectively. Since $2^{k-1} < 2n < 2^k$, then it must be the case that $k - 1 < T(2n) < k$.

Since we must have an integer number of comparisons and $k-1 < T(2n) < k$, we can round up and conclude that the worst case number of comparisons (upper bound) is k, where k is the smallest power of 2 greater than or equal to 2n.

(b)

We will show the lower bound for any comparison based algorithm for finding the median of 2 sorted sublists using an information-theoretic argument. We know that at the start any of the elements can be the median, so we have 2n median candidates at the root of the decision tree. Since we know that any correct algorithm must have at least 2n leafs, corresponding to each element being the median, then the sum of the candidates at the child nodes (at most 2 since we're splitting using one comparison) $\geq$ sum of the candidates at parent - this also means that the size of one of the child must be $\geq \frac{|parent|}{2}$.

Using these facts, consider two cases: If $2n = 2^k$ and $2n \neq 2^k$ for some integer k. If $2n = 2^k$, then the parent must hold 2^k candidates and one of its child must hold atleast 2^{k-1} candidates - always recursing down the bigger child will give us a minimum tree height of atleast $k = log_2(n) + 1$.

The second case is when $2n \neq 2^k$ for some integer k. Then, find smallest k s.t. k is a power of 2 and $2n < 2^k$ (Notice that $2n > 2^{k-1}$). We can use the exact information-theoretic argument above as above for starting arrays of size 2^k and 2^{k-1}, to get minimum tree height of k and k-1 respectively. Since $2^{k-1} < 2n < 2^k$, then it must be the case that $k - 1 < H(2n) < k$, where H(2n) is the height of the decision tree starting with 2n candidates.

Since we must have an integer tree height and $k - 1 < H(2n) < k$, we can round up and conclude that the lower bound is k, where k is the smallest power of 2 greater than or equal to 2n.

(c)

We can see that both the upper and lower bound provided above are exactly equal.

2: Duck Soup: a Stueue? Or a Quack?

(a) Implement a max-stack S
Make an array A and a variable *counter*.
For the first push(x), add two copies of x to A, set $counter = 1$.
For every push(x) after, add x to A, set A[counter + 2] = max(A[counter],x), increase *counter* by 2.
For every pop, remove the last two elements, decrease *counter* by 2.
For every return-max, return A[counter].

Each push requires two writes to the array, one comparison, and one increment, a total of 4 operations.
Each pop requires two removals from the array, and one decrement, a total of 3 operations.
Each return-max requires one read from the array.
$\therefore$ The max-stack maintains constant time per operation.

(b) Implement a max-queue Q using two max-stacks

```
Make two max-stacks $S_1, S_2$.
For every enqueue($x$), push($x$) to $S_1$.
For every dequeue:
    if pop $S_2$ returns NULL, pop all of the elements on $S_1$ and push them to $S_2$.
    pop the top element off of $S_2$.
For every return-max($Q$), return max(return-max($S_1$), return-max($S_2$)).
```

Since every enqueue is just a push onto max-stack S_1, each operation takes $O(1)$. Suppose after n enqueues, a dequeue is called. This dequeue operation is expensive when stack S_2 is empty, which means it requires at worst, $O(n)$ to push all of the n enqueues from stack S_1 to stack S_2. So after $n+1$ operations, the total cost is $2n$ and the amortized cost is $\frac{2n}{n+1} < \frac{2n}{n} = 2$.
$\therefore$ The max-queue takes $O(1)$ amortized time per operation.

(c)

```
Make a max-queue $Q$ of size $r$.
enqueue the first $r$ elements

For every remaining $n - r$ element:
    return-max(Q)
    dequeue
    enqueue(x)
```

In order to output the max value among the past r streamed elements, the algorithm enqueues the first r elements and takes $O(r)$ space, then calls return-max, dequeue, and enqueue for the remaining $n-r$ elements. From part **(b)**, we know that any sequence of n operations on the max-queue takes $O(n)$.
$\therefore$ The algorithm takes a total time $O(n)$.

3: Every Day I am Hashin'

(a) Prove that $\mathcal{G}$ is not 4-universal
Proof: We will give a counterexample. Let $n = 2$ and $k = 2$. Suppose the alphabet contains $\{0, 1\}$, all of distinct keys are $< 00, 01, 10, 11 >$ and T is a 2 x 2 table where each entry is a randomly generated b-bit string

a	b
c	d

So $g_T(00) = a \oplus c$, $g_T(01) = a \oplus d$, and $g_T(10) = b \oplus c$. We can easily determine the value of $g_T(11)$ without knowing anything else about T since there is only one choice left, $b \oplus d$.

More formally, assuming that the first three events are independent, for any $m = 2^b$

$$\begin{aligned}
&Pr(g_T(x_1) = v_1 \wedge g_T(x_2) = v_2 \wedge g_T(x_3) = v_3 \wedge g_T(x_4) = v_4) && (8)\\
&= Pr(g_T(x_1) = v_1 \wedge g_T(x_2) = v_2 \wedge g_T(x_3) = v_3) && (9)\\
&\times Pr(g_T(x_4) = v_4 | g_T(x_1) = v_1 \wedge g_T(x_2) = v_2 \wedge g_T(x_3) = v_3) && (10)\\
&= \frac{1}{m^3} \cdot 1 && (11)\\
&\neq \frac{1}{m^4} && (12)
\end{aligned}$$

$\therefore \mathcal{G}$ is not 4-universal

(b) Prove that $\mathcal{G}$ is 3-universal
Proof:

Given three distinct strings a,b,c of length n, and three hash values(not necessarily distinct) v1, v2, v3, we need to prove that $\forall$ a,b,c and $\forall$ v1,v2,v3

$$P(h(a) = v1 \ \ \& \ \ h(b) = v2 \ \ \& \ \ h(c) = v3) = \frac{1}{M^3}$$

Here we consider two cases for the proof:

Case 1: $\exists$ a position i in the three strings such that $a_i \neq b_i \neq c_i$.

Here we fix every position in the matrix T, except $T_{i,a_i}, T_{i,b_i}, T_{i,c_i}$

We know by definition

$$\bigoplus_{k=1}^{n} T_{k,a_k} = v1$$

$$\bigoplus_{k=1}^{n} T_{k,b_k} = v2$$

$$\bigoplus_{k=1}^{n} T_{k,c_k} = v3$$

Xoring both sides of these three equations by the fixed values, i.e every value other than $T_{i,a_i}, T_{i,b_i}, T_{i,c_i}$ respectively.

We get:

$$T_{i,a_i} = v1 \oplus fixedvalue = c1$$
$$T_{i,b_i} = v2 \oplus fixedvalue = c2$$
$$T_{i,c_i} = v3 \oplus fixedvalue = c3$$

Now since the matrix T is chosen randomly, and three of these values are different in the same row of the matrix, the probabilities are independent. Therefore,

$$P((\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \;\&\; (\bigoplus_{k=1}^{n} T_{k,b_k} = v2) \;\&\; (\bigoplus_{k=1}^{n} T_{k,c_k} = v3)) = P((T_{i,a_i} = c1) \;\&\; (T_{i,b_i} = c2) \;\&\; (T_{i,c_i} = c3))$$
$$= P((T_{i,a_i} = c1)).P((T_{i,b_i} = c2)).P((T_{i,c_i} = c3))(Byindependence)$$

$$P((T_{i,a_i} = c1)) = P((T_{i,b_i} = c2)) = P((T_{i,c_i} = c3)) = \frac{1}{M}$$

Because, the probability that a bit string of b bits, is equal to a particular constant is $\frac{1}{2^b} = \frac{1}{M}$

Therefore,

$$P((\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \;\&\; (\bigoplus_{k=1}^{n} T_{k,b_k} = v2) \;\&\; (\bigoplus_{k=1}^{n} T_{k,c_k} = v3)) = \frac{1}{M^3}$$

Case 2:

No position l exists such that $a_l \neq b_l \neq c_l$ Therefore, there must exist position i and j such that:

$$a_i \neq b_i, but b_i = c_i or a_i = c_i$$
$$b_j \neq c_j, but a_j = b_j and b_j \neq c_j$$

We first consider strings a and b. They differ in position i, and we fix all other positions of the table w.r.t string a and b. Using the same proof as case 1, we can easily say that:

$$P((\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \;\&\; (\bigoplus_{k=1}^{n} T_{k,b_k} = v2) = \frac{1}{M^2}$$

Now we consider string b and c. They differ in position j. We also know that either $c_j = b_j$ or $c_j = a_j$ therefore T_{j,c_j} is fixed

We assume we are given $(\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \;\&\; (\bigoplus_{k=1}^{n} T_{k,b_k} = v2$. Also $a_i = c_i$ or $b_i = c_i$, we know $T_{i,c_i} is fixed.$ Now we fix all other values of c other than j.

Therefore, only one value is random, which is T_{j,c_j}

$$P(h(c) = v3|((\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \;\&\; (\bigoplus_{k=1}^{n} T_{k,b_k} = v2))) = P((\bigoplus_{k=1}^{n} T_{k,c_k} = v2|((\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \;\&\; (\bigoplus_{k=1}^{n} T_{k,b_k} = v2)))$$

Xoring both sides with the fixed values

$$= P(T_{j,c_j} = c3|((\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \;\&\; (\bigoplus_{k=1}^{n} T_{k,b_k} = v2)))$$

Now using the same result proved above, we can easily say

$$P(T_{j,c_j} = c3|((\bigoplus_{k=1}^{n} T_{k,a_k} = v1) \ \ \& \ \ (\bigoplus_{k=1}^{n} T_{k,b_k} = v2))) = \frac{1}{M}$$

Therefore,

$$P((T_{i,a_i} = c1) \ \ \& \ \ (T_{i,b_i} = c2) \ \ \& \ \ (T_{i,c_i} = c3))$$
$$= P((T_{i,a_i} = c1)|(T_{i,b_i} = c2) \ \ \& \ \ (T_{i,c_i} = c3)) * P((T_{i,b_i} = c2) \ \ \& \ \ (T_{i,c_i} = c3))$$
$$= \frac{1}{M} * \frac{1}{M^2} = \frac{1}{M^3}$$

Q.E.D