CSCI3160: Midterm Exam Solutions

Problem 1.
1. T
2. F
3. F
4. T
5. T
6. T
7. F
8. T

Problem 2. $[1, 10]$, $[20, 30]$, $[40, 50]$, $[60, 70]$

Problem 3. Many solutions exist, e.g., bd, de, eg, cg, df, ae. Total cost = 9.

Problem 4. Many solutions exist, e.g., $a = 00100$, $b = 00101$, $c = 0011$, $d = 010$, $e = 011$, $f = 000$, $g = 10$, and $h = 11$. Here is another solution: $a = 0000$, $b = 0001$, $c = 001$, $d = 100$, $e = 101$, $f = 110$, $g = 111$, and $h = 01$.

Problem 5. If $k = 1$, simply return the maximum element in S in $O(n)$ time. Otherwise, spend $O(n)$ time finding the median e of S (i.e., the element with rank $n/2$ in S). Divide S into $S_1 = \{e' \in S \mid e' \le e\}$ and $S_2 = \{e' \in S \mid e' > e\}$, which can also be done in $O(n)$ time. Recursively find the $(k/2)$-split set T_1 of S_1 and the $(k/2)$-split set T_2 of S_2. Return $T_1 \cup T_2$.

To analyze the running time, denote by $f(n, k)$ the time of our algorithm on parameters n and k. It holds that $f(n, 1) = O(n)$ and $f(n, k) = O(n) + 2f(n/2, k/2)$. We can derive:

$$\begin{aligned}
f(n,k) &= O(n) + 2f(n/2, k/2) \\
&= O(n) + 2(O(n/2) + 2f(n/4, k/4)) = 2 \cdot O(n) + 4f(n/4, k/4) \\
&= 2 \cdot O(n) + 4(O(n/4) + 2f(n/8, k/8)) = 3 \cdot O(n) + 8f(n/8, k/8) \\
&... \\
&= \log_2 h \cdot O(n) + h \cdot f(n/h, k/h) \\
&... \\
&= \log_2 k \cdot O(n) + k \cdot f(n/k, 1) = O(n \log k)
\end{aligned}$$

Problem 6. 1. Consider $d_1 = 4$ and $d_2 = 3$. The algorithm is not optimal for $n = 6$.

2. Take an arbitrary optimal solution that uses x'_1, x'_2, and x'_3 coins of d_1, d_2, and d_3, respectively. Hence:

$$5x_1 + 2x_2 + x_3 = 5x'_1 + 2x'_2 + x'_3 \quad (1)$$

We will show

$$4x_1 + x_2 \ge 4x'_1 + x'_2. \quad (2)$$

Plugging (2) into (1) yields: $x_1 + x_2 + x_3 \le x'_1 + x'_2 + x'_3$, which indicates that $\{x_1, x_2, x_3\}$ is optimal.

To prove (2), first observe that $x_1 \ge x'_1$ (because otherwise $5x'_1 \ge 5(x_1 + 1) > n$). We distinguish two cases:

Case 1: $x_1 = x'_1$**.** We must have $x_2 \geq x'_2$ because otherwise $2x'_2 + x'_1 \geq 2(x_2 + 1) + x_1 > n$. It follows that (2) holds.

Case 2: $x_1 > x'_1$**.** It suffices to prove $x'_2 \leq 4$ because this will yield $4(x_1 - x'_1) + x_2 \geq 4 \geq x'_2$, which then gives (2). To prove $x'_2 \leq 4$, observe that if $x'_2 \geq 5$, we can replace 5 coins of 2 dollars with 2 coins of 5 dollars, contradicting the optimality of $\{x'_1, x'_2, x'_3\}$.

Problem 7. 1. a_1 is greater than $(n/2) - 1$ elements in A_1 and at most $(n/2) - 1$ elements in A_2; hence, its rank in S is at most $1 + (n/2) - 1 + (n/2) - 1 = n - 1$. b_1 is greater than the first $n/2$ elements in A_1 and the first $(n/2) - 1$ elements in A_2; hence, its rank in S is at least n.

2. We will deal with a more general problem. Let A be an array of size n, and B be an array of size m, where n and m are powers of 2. Each array is sorted in ascending order, and all the $n + m$ integers in $A \cup B$ are distinct. Given an integer $k \in [1, n + m]$, we will find the element with rank k in $A \cup B$ in $O(\log n + \log m)$ time. We will use the notation $A[i : j]$ to refer to the subarray storing $A[i]$, $A[i + 1]$, ..., $A[j]$; $B[i : j]$ is defined similarly.

If $n = 1$, then we compare $A[1]$ with $B[k]$. If $A[1] < B[k]$, return $B[k - 1]$; otherwise, return $B[k]$. The cost is $O(1)$. Similarly, the problem can also be solved in constant time if $m = 1$.

Next, we consider $n \geq 2$ and $m \geq 2$. Let $a = A[n/2]$ and $b = B[m/2]$. Assume, w.l.o.g., that $a < b$. By an argument similar to how we proved question 6(1), we know that the rank of a in $A \cup B$ is at most $\frac{n+m}{2}$, and that of b is at least $\frac{n+m}{2}$.

- If $k \leq (n + m)/2$, none of the elements in $B[\frac{m}{2} + 1 : m]$ can be the final answer. We recurse on A, $B[1 : m/2]$, and k (i.e., looking for the k-th smallest in $A \cup B[1 : m/2]$).

- If $k > (n + m)/2$, none of the elements in $A[1 : n/2]$ can be the final answer. We recurse on $A[1 + n/2 : n]$, B, and $k - n/2$ (i.e., looking for the $(k - n/2)$-th smallest in $A[1 + n/2 : 1] \cup B$).

In either case, we spend constant time before entering recursion. Each time we recurse, either A shrinks in half or B does. The recursion depth is therefore $O(\log n + \log m)$.