CSCI 2300 — Introduction to Algorithms
Spring 2020 Exam 2 (April 23, 2020)
SOLUTIONS (document version 1.1)

1. **(2 POINTS)**
   SOLUTION: (d) H — for the make-up: (h) H

2. **(2 POINTS)**
   SOLUTION: (d) H — for the make-up (h) H

3. **(2 POINTS)**
   SOLUTION: (b) E — for the make-up (f) E

4. **(2 POINTS)**
   SOLUTION: (f) V — for the make-up (c) V

5. **(2 POINTS)**
   SOLUTION: (c) 2 — for the make-up (d) 2

6. **(2 POINTS)**
   SOLUTION: (e) 4 — for the make-up (b) 4

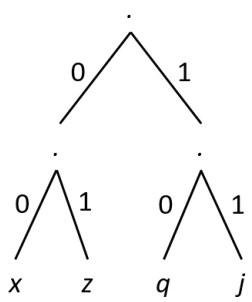
7. **(12 POINTS)**

(a) **(5 POINTS)**

SOLUTION: A valid Huffman encoding tree is shown below. Other variations are fine, as long as the four symbols are each encoded with 2 bits.

NOTE: The make-up exam has q and x frequencies swapped.

→ 1pt for a valid tree structure
→ 1pt for showing the four symbols as leaf nodes
→ 1pt for showing 0/1 on each edge
→ 2pts for a valid encoding for each symbol



q: 10
j: 11
z: 01
x: 00

(b) **(2 POINTS)**

SOLUTION: For a file with 1,000,000 symbols and given frequencies, we have the number of bits in the encoded file calculated as follows:

→ 2pts: $250,000 \times 2 + 300,000 \times 2 + 220,000 \times 2 + 230,000 \times 2 = 2,000,000$
(NO PARTIAL CREDIT)

(c) **(5 POINTS)**

SOLUTION: Removing symbol j leaves only three symbols, so while x and z are still encoded as 2 bits, q can now be encoded using only 1 bit.

→ 1pt: The algorithm takes as input the original encoded file from part (b).

→ 2pts: As it decodes the input using the original codewords from part (a), the new codewords are written **directly** to the new file.

→ 1pt: The output of the algorithm is then the newly encoded file that contains codewords for only q, x, and z.

→ 1pt: The runtime is linear in the size of the input file.

8. **(12 POINTS)**

SOLUTION: First note that each vertex $u \in U$ must have at least one neighbor in $V - U$ (or else the problem has no solution).

Also, if $T = (X, Y)$ is the optimal LST, then $X - U$ must be a minimum spanning tree of $G' = (V - U, E')$, where $E' \subset E$ is the set of edges incident to U.

The greedy algorithm is as follows.

$\rightarrow$ 2pts: Given as input graph $G = (V, E)$ and subset $U \subset V$.

$\rightarrow$ 2pts: Use Kruskal's (or Prim's) algorithm to find an MST of $G' = (V - U, E')$ (i.e., of the graph that does not contain vertices or incident edges of U). Call this MST subgraph T.

$\rightarrow$ 2pts: For each vertex $u \in U$, add the lightest edge between u and G' to T.

$\rightarrow$ 2pts: If subgraph T contains all vertices of G, then the output of the algorithm is subgraph T; otherwise, no such LST exists.

$\rightarrow$ 1pt: Constructing G' is linear.

$\rightarrow$ 2pts: The runtime of the MST algorithm on G' to produce T is $O(|E| \log |V|)$.

$\rightarrow$ 1pt: Adding the lightest edges from U runs in linear time $O(|E|)$.

Other variations are certainly possible, though the algorithm **must** be a greedy algorithm and must be clearly described (no code).

9. **(12 POINTS)** Determine the runtime of the four divide-and-conquer algorithms described below. More specifically, write a recurrence relation for each, then express the runtime complexity using Big $O()$ notation. If you use the Master theorem, be sure to clearly indicate coefficients a, b, and d.

(a) **(3 POINTS)**
SOLUTION:
→ 1pt: The recurrence relation is $T(n) = 3T(n/3) + O(n^3)$.
→ 1pt: From this, we can use the Master Theorem with $a = 3$, $b = 3$, and $d = 3$.
→ 1pt: The Master Theorem tells us $d > \log_b a$ $(3 > 1)$, so $T(n) = O(n^3)$ (or $\Theta(n^3)$).

(b) **(3 POINTS)**
SOLUTION:
→ 1pt: The recurrence relation is $T(n) = 2T(n/4) + O(n^0)$.
→ 1pt: From this, we can use the Master Theorem with $a = 2$, $b = 4$, and $d = 0$.
→ 1pt: The Master Theorem has $d < \log_b a$ $(0 < 0.5)$, so $T(n) = O(n^{\log_4 2})$ or $O(n^{0.5})$ (or $\Theta()$).

(c) **(3 POINTS)**
SOLUTION:
→ 1pt: The recurrence relation is $T(n) = 4T(n-2) + n$, which we can expand as:

$$T(n) = 4(4T(n-4) + n) + n$$

$$T(n) = 4(4(4T(n-6) + n) + n) + n$$

$$T(n) = 4(4(4(\cdots) + n) + n) + n$$

→ 1pt: Observe here that the number of subproblems quadruples $n/2$ times and each subproblem uses $O(n)$ time, with the dominating term being the exponential term $4^{n/2}$, which simplifies to 2^n, so we have:
→ 1pt: $T(n) = O(2^n)$ (or $\Theta()$); also acceptible is $T(n) = O(4^n)$.

(d) **(3 POINTS)**
SOLUTION:
→ 1pt: The recurrence relation is $T(n) = 9T(n/3) + O(\log_3 n)$.
→ 1pt: From this, the Master Theorem has $a = 9$, $b = 3$, but we do not have a clear coefficient d. The Master Theorem requires us to compare $\log_b a$ to d, so we know at least that $\log_b a = \log_3 9 = 2$. Since the last term $O(\log_3 n)$ is certainly asymptotically less than $O(n^2)$ (i.e., if $d = 2$), we can conclude from the Master Theorem that:
→ 1pt: $T(n) = O(n^{log_b a}) = O(n^2)$

10. **(16 POINTS)**

SOLUTION:

→ 2pts: Must state that the input is unsorted array $Z[1 \ldots n]$ of n **distinct** integers.

→ 2pts: Algorithm must be or at least use a divide-and-conquer algorithm (e.g., median of medians).

→ 6pts: One possible algorithm is to use the median of medians divide-and-conquer algorithm to first identify the median.

Next, in one pass through Z, divide Z into two subarrays $Z_<$ and $Z_\geq$ that contain values less than the median or greater than or equal to the median, respectively.

Next, merge to form output array S by adding alternating elements from $Z_\geq$ and $Z_<$ at each level.

Another solution is to recursively divide array Z into subproblems of size $n/2$ until the subproblems are of size $n = 2$ at most, then merging these smaller subproblems similar to above. This approach requires the additional work of considering the "conquer" or merge cases where the one or both of the subproblems to merge contain lists of odd length or duplicate values exist.

→ 2pt: Must state that the output is the zigzag array (e.g., array S in the above algorithm).

→ 4pts: Must state and show that the algorithm runs in linear time $O(n)$.

11. **(18 POINTS)**

SOLUTION:

Subproblems:

→ 5pts: Define subproblem $A(i, j)$ for $1 \leq i$ and $j \leq n$ to represent the probability that team C is the first to win n games after $m = i + j$ games have been played and C has won i games.

Algorithm and Recursion:

→ 4pts: Initialize A by looking at the "end" possibilities in which team C has won n games and team L has not won n games. In this case, we set $A(n, j) = 1$ for $j \neq n$ and $A(i, n) = 0$ for all i.

As an example with $n = 4$, we would have the matrix shown to the right:

	j=1	j=2	j=3	j=4
i=1				0
i=2				0
i=3				0
i=4	1	1	1	0

→ 5pts: The other subproblems can be solved incrementally in *decreasing order* of $i+j$ using the recurrence relation:

$$A(i, j) = \frac{1}{2}(A(i, j + 1) + A(i + 1, j))$$

The recursion is such that we compute $A(i, j)$ based on the outcome of the $(i+j+1)$th game. The two possible outcomes here (i.e., win or loss) are equally likely with probability $\frac{1}{2}$.

More specifically, if C wins, then it wins the game with probability $A(i+1, j)$. If L wins, C has probability of winning $A(i, j + 1)$.

Continuing the example with $n = 4$, we would have the matrix shown to the right:

	j=1	j=2	j=3	j=4
i=1	0.5	.3125	0.125	0
i=2	.6875	0.5	0.25	0
i=3	0.875	0.75	0.5	0
i=4	1	1	1	0

Runtime:

→ 4pts: Total runtime is $O(n^2)$. If we wish to find solutions to all subproblems, we have $O(n^2)$ subproblems, each requiring $O(1)$ time.

(Extra note, not required: If we instead are only interested in a specific subproblem $A(i, j)$, we can stop after we have solved $O((n - (i + j))^2)$ subproblems, still $O(n^2)$.)

12. **(18 POINTS)**

SOLUTION:

Subproblems:

→ 4pts: Suppose the given matrix is matrix M with n rows and n columns numbered $1 \dots n$. Define subproblem $A(i, j)$ for $1 \leq i \leq n$ and $1 \leq j \leq n$ to represent the maximum length of a valid path ending at $M(i, j)$.

Algorithm and Recursion:

→ 4pts: Initialize $A(i, j) = 1$ for all i and j since all individual cells of the matrix represent a path of length 1. Subproblem $A(1, 1) = 1$ is the only solution guaranteed to be correct at this point.

Alternatively, depending on how you might have interpreted the question, we can initialize all non-entry points $A(i, j) = 0$, so every cell that is not the first row and/or column (or perhaps just the cell $A(1, 1) = 1$.

→ 4pts: The other subproblems can be solved incrementally in *increasing order* of i and j, meaning we process row by row or column by column using the recurrence relation:

$$A(i, j) = \max \begin{cases} A(i-1, j) + 1 & \text{if } i > 1 \text{ and } |M(i-1, j) - M(i, j)| \leq 1 \\ A(i, j-1) + 1 & \text{if } j > 1 \text{ and } |M(i, j) - M(i, j-1)| \leq 1 \end{cases}$$

The recursion here is such that we compute $A(i, j)$ by adding 1 to the maximum length thus far for either the cell above or to the left of the current cell.

Alternatively, depending on how you might have interpreted the question, we can change the above recurrence relation to add $A(i, j)$ instead of 1.

(And not required for points, but we would also keep track of the previous cell that got us to the maximum thus far for each $A(i, j)$ entry.)

→ 2pts: Once matrix A is completely filled in, we can select a maximum path by finding a cell with the maximum total length (i.e., the maximum $A(i, j)$ for all i and j).

Alternatively, depending on how you might have interpreted the question, we can limit our selection to only the lower right-hand corner $A(n, n)$ or require that a cell be selected from the rightmost column or bottom row.

Runtime:

→ 4pts: Total runtime is $O(n^2)$ (or $\Theta(n^2)$ since we are finding the maximum over all possible solutions). More specifically, we have $O(n^2)$ subproblems, each requiring $O(1)$ time.

Any additional pre- or post-processing is, at most, $O(n^2)$, so overall, our total runtime is $O(n^2)$.