1 Problem 1 (25 points): Multiple Choice

Indicate which of the following are true. You do not need to give a proof, nor do you need to give counterexamples.

(a) Let $T(n) = 3T(n - 2)$, with $T(1) = T(2) = 1$. Circle whichever of the following are true (there may be more than one).

\[
\begin{align*}
T(n) &= \Theta(3^{n/2}) \\
T(n) &= \Theta(2^{n/3}) \\
T(n) &= \Theta(3^n) \\
T(n) &= \Theta(2^n)
\end{align*}
\]

(b) $(\log(n^3))^5 = O(2^{5\log\log n})$

true \hspace{1cm} false

(c) On every input, the expected running time of randomized quicksort is $O(n \log n)$

true \hspace{1cm} false

(d) The median-of-medians algorithm (BPFRT) runs in $O(n)$ time for median-finding, but $\omega(n)$ time for selecting the $k$-th smallest element.

true \hspace{1cm} false

(e) The sorting lower bound implies that no sorting algorithm can sort $n$ integers between 1 and $n^{10}$ in $o(n \log n)$ time (in the worst case).

true \hspace{1cm} false

(f) The best list-based Union-Find data structure we discussed in class has an amortized cost in $O(\log n)$ for Make-Set, $O(1)$ for Union, and $O(1)$ for find.

true \hspace{1cm} false

(g) If we use trees with union-by-rank and path compression, the amortized cost of both Union and Find is $O(\log^* n)$.

true \hspace{1cm} false

(h) Let $H$ be a universal family of hash functions from $U$ to $[M]$. Then for any sequence of $L$ insert, delete, and lookup operation, the expectation of the total running time (when $h$ is drawn uniformly from $H$) is $O(L)$.

true \hspace{1cm} false

(i) Let $H$ be a universal family of hash functions. Then for every $h \in H$, if we choose two elements $x, y \in U$ uniformly at random, the probability that $h(x) = h(y)$ is at most $1/M$

true \hspace{1cm} false

(j) If we build a B-tree with $t = \Theta(\log n)$, then the running time of Insert is $O(\log n)$

true \hspace{1cm} false
2 Problem 2 (25 points): Data Structures

(a) Draw all possible 2-3-4 trees on the keys \{1, 2, 3, 4, 5\}.

Solution:

```
2
/   \
1    345

3
/   \
12   45

4
/   \
123   5

24
/   \
1    3    5
```
(b) For each of the trees in the previous part, draw a corresponding red-black tree (use a dashed line for red and a solid line for black).

Solution:
(c) Draw all possible binary min-heaps on the keys \{1, 2, 3, 4, 5\}.

**Solution:**

```
1
/   \
2   3
|    |
4   5

1
/   \
2   3
|    |
5   4

1
/   \
2   4
|    |
3   5

1
/   \
2   4
|    |
5   3

1
/   \
2   5
|    |
3   4

1
/   \
3   2
|    |
5   4

1
/   \
3   2
|    |
4   5

1
/   \
3   4
|    |
5   3
```


(d) How many different binomial heaps are there on the keys \( \{1, 2, 3, 4, 5\} \)? Justify your answer. Hint: think about the structure of a binomial heap. How many ways are there to fill in that structure legally?

**Solution:** 15. A binomial heap on \( \{1, 2, 3, 4, 5\} \) has the following structure:

![Diagram of a binomial heap](image)

There are 5 ways of filling in node A. For each choice, B is fixed (it must be the smallest of the remaining elements). Then there are three ways of filling in D, and for each such choice, C and E are fixed. Thus overall there are \( 5 \times 3 = 15 \) possible binomial heaps.
3 Problem 3 (25 points): Amortized Analysis

In this problem we have two stacks $A$ and $B$. In what follows, we will use $n$ to denote the number of elements in stack $A$ and use $m$ to denote the number of elements in stack $B$. Suppose that we use these stacks to implement the following operations:

- **PushA($x$)**: Push element $x$ onto $A$.
- **PushB($x$)**: Push element $x$ onto $B$.
- **MultipopA($k$)**: Pop min($n, k$) elements from $A$.
- **MultipopB($k$)**: Pop min($m, k$) elements from $B$.

We are using the stacks as a black box. You may assume that **PushA**, **PushB**, **MultipopA(1)**, and **MultipopB(1)** each take exactly one unit of time (i.e. it takes one time step to push or pop a single element).

Justify all of your answers.

(a) Consider the potential function $\Phi = n + m$. Under this potential function, what is the amortized cost of **MultipopA($k$)**, and what is the amortized cost of **MultipopB($k$)**?

**Solution:** MultipopA($k$) decreases the potential function by min($n, k$) and has true cost of min($n, k$), so the amortized cost is $0 = O(1)$. Similarly, MultipopB($k$) decreases the potential function by min($m, k$) and has true cost of min($m, k$), so the amortized cost is $0 = O(1)$.
Now suppose we add a new operation $\text{TRANSFER}(k)$: Repeatedly pop one element from $A$ and push it into $B$, until either $k$ elements have been moved or $A$ is empty.

(b) What is the amortized cost of $\text{TRANSFER}(k)$ as a function of $n, m, k$ under our potential function $\Phi = n + m$?

**Solution:** Transfer$(k)$ decreases $n$ by $\min(n, k)$, but increases $m$ by the same amount. Thus the change in potential is 0, so the amortized cost is the true cost, which is $2 \min(n, k)$ ($\min(n, k)$ pops and $\min(n, k)$ pushes).
(c) Now consider the potential function $\Phi = 3n + m$. What is the amortized cost of $\text{TRANSFER}(k)$ under this potential function?

**Solution:** Let $x = \min(n, k)$. Then the potential before the operation is $3n + m$, while the potential after the operation is $3(n - x) + (m + x) = 3n + m - 2x$. Thus the change in potential is $-2x$. Since the true cost is $2x$ (a cost of $x$ for the pops and a cost of $x$ for the pushes), the amortized cost is $0 = O(1)$. 
4  Problem 4 (25 points): Dynamic Programming

A palindrome is a string which is equal to its own reversal, i.e., it is the same whether read backwards or forwards. Given a string \( X = (x_1, x_2, \ldots, x_n) \), a string \( Z = (z_1, z_2, \ldots, z_k) \) is a **palindromic subsequence** of \( X \) if \( Z \) is a palindrome and if \( Z \) is a subsequence of \( X \) (i.e., there are values \( i_1, i_2, \ldots, i_k \in [n] \) such that \( i_j < i_{j+1} \) for all \( j \in [k-1] \) and \( z_j = x_{i_j} \) for all \( j \in [k] \)). In this problem we will consider algorithms to find the longest palindromic subsequence of \( X \).

For \( 1 \leq i \leq j \leq n \), let \( S(i, j) \) be the length of the longest palindromic subsequence of \( (x_i, x_{i+1}, \ldots, x_j) \). So \( S(1, n) \) is the length of the longest palindromic subsequence of \( X \).

(a) Give a recursive formula (not algorithm) for \( S(i, j) \). Justify your answer.

**Solution:**

\[
S(i, j) = \begin{cases} 
0 & \text{if } i > j \\
1 & \text{if } i = j \\
\max(S(i+1, j), S(i, j-1)) & \text{if } i < j \text{ and } x_i \neq x_j \\
2 + S(i+1, j-1) & \text{if } i \neq j \text{ and } x_i = x_j 
\end{cases}
\]

For justification, first note that the base cases of \( i = j \) and \( i > j \) are clearly correct by definition. Now consider some \( i < j \). If \( x_i \neq x_j \) then the longest palindromic subsequence of \( (x_i, \ldots, x_j) \) cannot start with \( x_i \) and end with \( x_j \), so the optimal solution must not include at least one of these, and so the optimal solution must be \( \max(S(i+1, j), S(i, j-1)) \). On the other hand, if \( x_i = x_j \), then the longest palindromic subsequence of \( (x_i, \ldots, x_j) \) starts with \( x_i \) and ends with \( x_j \) and in between consists of the LPS of \( (x_{i+1}, \ldots, x_{j-1}) \) (there might be other palindromic subsequence which do not have this structure which are equally long, but at least one of the longest palindromic subsequences does). Thus the length of the longest palindromic subsequence would be \( 2 + S(i+1, j-1) \).
(b) Consider the recursive algorithm *without memoization* (not the dynamic programming algorithm) defined by your formula from the previous part. What would the running time of this algorithm be? Justify your answer.

**Solution:** At worst, each call to the algorithm results in two recursive calls on an instance of size one smaller, plus a constant amount of work. Thus the running time is at most $T(n) = 2T(n - 1) + O(1)$, which is $O(2^n)$. On the other hand, if every character of $X$ is unique (so the largest palindromic subsequence has size 1), every call to the algorithm will result in two recursive calls on an instance that has one less character, and thus the running time is $\Omega(2^n)$. 
(c) Now consider the dynamic programming algorithm (either top-down or bottom-up) defined by your answer to part (a). What is its running time? Justify your answer.

**Solution:** The DP table will have $\Theta(n^2)$ entries, and each entry take $O(1)$ time to fill in. Thus the running time is $O(n^2)$. 