1. Answer the following questions (circle True or False and fill in the blank).

(a) True or False: The average-case behavior of an algorithm is always better than its worst-case behavior.

(b) True or False: The amount of work done by an algorithm usually depends on the size of the input.

(c) True or False: The size of the input to the problem always corresponds to the size of an array.

(d) Suppose the running time of a divide-and-conquer algorithm is given by the following recurrence:

\[ T(n) = 3T\left(\frac{n}{4}\right) + \Theta(n) \]

This algorithm partitions the original problem into \[ \text{subproblems, each of size } \] \[ \text{time to } \]

(e) True or False: \[ f(n) = 1000n \log^9 n + n^3 + n^2 \log^5 n + \frac{1}{1000}n^4 \log^2 n \] is \( \Omega(n^3) \).

(f) True or False: \[ f(n) = 1000n \log^9 n + n^3 + n^2 \log^5 n + \frac{1}{1000}n^4 \log^2 n \] is \( O(n^3) \).

(g) True or False: \[ f(n) = 1000n \log^9 n + n^3 + n^2 \log^5 n + \frac{1}{1000}n^4 \log^2 n \] is \( O(3^n) \).

(h) True or False: Every comparison-based sorting algorithm must take at least \( \Omega(n \log n) \) time in the best-case.

(i) True or False: If the input is a sorted array of \( n \) elements, the selection problem can be solved in \( O(n) \) time.

2. For each part, circle asymptotic bounds that apply.

(a) What is the worst-case time for MAX-HEAPIFY on an array of \( n \) distinct elements?
\[ O(1) \quad O(\log n) \quad O(n) \quad O(n \log n) \]

(b) What is the worst-case time for Counting sort on \( n \) elements, each in the range 1 to \( \log n \)?
\[ O(1) \quad O(n) \quad O(n \log n) \quad O(n^2) \]

(c) What is the best-case time for Counting sort on \( n \) distinct elements, each in the range 1 to \( n^2 \)?
\[ O(1) \quad O(n) \quad O(n \log n) \quad O(n^2) \]

(d) What is the amount of extra storage used by Heapsort on \( n \) distinct elements?
\[ O(1) \quad O(n) \quad O(n \log n) \quad O(n^2) \]

(e) What is the worst-case time for BUILD-MAX-HEAP on an unordered array of \( n \) distinct elements?
\[ O(1) \quad O(n) \quad O(n \log n) \quad O(n^2) \]

3. Consider the following comparison-based sorting algorithm.

\text{Fast-Sort}(array L of integers)
\begin{align*}
\text{if } & (L \text{ has less than 3 elements}) \\
& \text{sort } L \text{ with Insertion-Sort and return } L \\
\text{else} & \\
& L1 = \text{Fast-Sort}(1\text{st quarter of } L) \\
& L2 = \text{Fast-Sort}(2\text{nd quarter of } L) \\
& L3 = \text{Fast-Sort}(3\text{rd quarter of } L) \\
& L4 = \text{Fast-Sort}(4\text{th quarter of } L) \\
& L = \text{4-Way-Merge}(L1, L2, L3, L4) \\
& \text{return } L
\end{align*}
Assume the complexity of 4-way-Merge($L_1, L_2, L_3, L_4$) is $O(n_1 + n_2 + n_3 + n_4)$, where $n_i$ is the number of elements in $L_i$, $i = 1, 2, 3, 4$. Write down the the recurrence relation for the asymptotic running time of Fast-Sort:

4. Consider the following comparison-based algorithm for the selection problem.

```java
New-Select(A, p, r, i)
1 if p == r
2 then return A[p]
3 m = Magic-Median(A, p, r) // returns median of A, m
4 q = Magic-Partition(A, p, r, m) // partitions A around m
5 k = q - p + 1
6 if i <= k
7 then return New-Select(A, p, q, i)
8 else return New-Select(A, q + 1, r, i - k)
```

(a) Assume the running time of Magic-Median($L$) is $O(1)$ and the running time of Magic-Partition($L$) is $O(\log n)$. State the recurrence relation for the asymptotic running time of New-Select.

(b) Solve the recurrence you derived in part (a); show the the tightest bound you can. Show your work and clearly state any assumptions you make.

(c) Is it possible for the algorithm to run in the amount of time you determined in (b)? Why or Why not?

5. Consider inserting the following keys, in this order, into a hash table of length $m = 11$.

keys to insert (in this order): 3, 4, 14, 15, 25, 26

(a) Suppose you use chaining with the hash function $h(k) = k \mod 11$. Illustrate the result of inserting the keys above using chaining.

(b) Suppose you use open addressing with the primary hash function $h_1(k) = k \mod 11$. Illustrate the result of inserting the keys above using linear probing, i.e., $h(k, i) = (h_1(k) + i) \mod 11$.

(c) Suppose you use open addressing with hash functions $h_1(k) = k \mod 11$ and $h_2(k) = (k \mod 10)$. Illustrate the result of inserting the keys above using double hashing, i.e., $h(k, i) = (h_1(k) + (h_2(k) \cdot i)) \mod 11$. 