Quiz 5 Practice Questions

Quiz 5 Problems

Problem 1: Divide and Conquer Time Analysis

```
def process_data(arr):
    if len(arr) == 0:
        return 0
    if len(arr) == 1:
        return 1 if arr[0] > 0 else 0
    mid = len(arr) // 2
    left = process_data(arr[:mid])
    right = process_data(arr[mid:])
    return left + right
```

Recurrence: ____

Question: Based on how the array is divided and how long the combine step takes, write a recurrence for the running time of this function.

Problem 3: Divide and Conquer Time Analysis

```
def process_data(arr):
    if len(arr) < 3:
        return arr
    a = process_data(arr[:len(arr)//2])
    b = process_data(arr[len(arr)//2:len(arr)//2 + 1])
    c = process_data(arr[len(arr)//2 + 1:])
    return a + b + c
```

Recurrence: ____

Question: Based on how the array is divided and how long the combine step takes, write a recurrence for the running time of this function.

Problem 4: Divide and Conquer Time Analysis

```
def process_data(arr):
    if len(arr) <= 2:
        return arr
    mid = len(arr) // 2
    left = process_data(arr[:mid])
    right = process_data(arr[mid:])
    return left + right
```

Recurrence: ____

Question: Based on how the array is divided and how long the combine step takes, write a recurrence for the running time of this function.

Problem 5: Divide and Conquer Time Analysis

```
def process_data(arr):
    if len(arr) <= 1:
        return arr
    third = len(arr) // 3
    a = process_data(arr[:third])
    b = process_data(arr[third:2*third])
    c = process_data(arr[2*third:])
    return a + b + c
```

Recurrence: ____

Question: Based on how the array is divided and how long the combine step takes, write a recurrence for the running time of this function.

Problem 6: Divide and Conquer Time Analysis

```
def process_data(points):
    if len(points) <= 3:
        return brute_force(points)
    mid = len(points) // 2
    left = process_data(points[:mid])
    right = process_data(points[mid:])
    cross = find_cross_pairs(points, mid)
    return left + right + cross
```

Recurrence: ____

Question: Based on how the array is divided and how long the combine step takes, write a recurrence for the running time of this function. Assume that $find\_cross\_pairs$ runs in $O(n)$ time.

Problem 8: Divide and Conquer Time Analysis

```
def process_data(arr):
    if len(arr) <= 1:
        return arr[0]
    mid = len(arr) // 2
    left = process_data(arr[:mid])
    right = process_data(arr[mid:])
    return 0.7 * left + 0.3 * right
```

Recurrence: ____

Question: Based on how the array is divided and how long the combine step takes, write a recurrence for the running time of this function.

Problem 9: Divide and Conquer Time Analysis

```
def process_data(data):
    if len(data) <= 1:
        return dict(data)
    mid = len(data) // 2
    left = process_data(data[:mid])
    right = process_data(data[mid:])
    for key in right:
        left[key] = right[key]
    return left
```

Recurrence: ____

Question: Based on how the array is divided and how long the combine step takes, write a recurrence for the running time of this function.

Problem 10: Counting Sort Example

```
arr = [4, 2, 2, 8, 3, 3, 1]
```

Question: Show how counting sort would sort this list step by step. What is the running time of counting sort in terms of n (length of the array) and k (maximum key value)? What assumptions must be true about the input data for counting sort to be efficient?

Problem 11: When to Use Counting Sort

```
arr = [15, 14, 16, 14, 15]
```

Question: Can counting sort be used efficiently on this input? What is the running time in terms of n and k? What assumptions about the data must hold for counting sort to be efficient?

Problem 12: Radix Sort on Strings

```
arr = ['bca', 'cab', 'abc', 'acb']
```

Question: Show the steps radix sort would follow when sorting this list of 3-letter strings. What is the running time of radix sort in terms of n (number of strings) and d (number of characters per string)? What assumptions must be true about the input for radix sort to be efficient?

Problem 13: Bucket Sort on Decimals

```
arr = [0.13, 0.25, 0.22, 0.45, 0.21, 0.24]
```

Question: Show how bucket sort would process this list. What is the running time of bucket sort in terms of n? What assumptions about the distribution of input data must hold for bucket sort to be efficient?

Problem 14: Counting Sort - Characters

```
arr = ['d', 'a', 'c', 'b', 'a']
```

Question: Show how counting sort would sort this list of lowercase letters assuming they are converted to ASCII codes. What is the running time in terms of n (number of characters) and k (range of ASCII codes)? What assumptions must be true for counting sort to be efficient?

Problem 15: When Not to Use Counting Sort

```
arr = [100, 50000, 30000, 20000]
```

Question: Explain why counting sort is a poor choice for this input. What would k be and how does it affect the running time? What assumptions about the input data make counting sort inefficient in this case?

Problem 16: Radix Sort - Integers

```
arr = [329, 457, 657, 839, 436, 720, 355]
```

Question: Show how radix sort processes this list of integers. What is the running time in terms of n (number of integers) and d (number of digits)? What assumptions must be true about the input for radix sort to be efficient?

Problem 17: Bucket Sort - Uniform Input

```
arr = [0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68]
```

Question: Perform bucket sort on this input. What is the running time in terms of n? How does the uniform distribution of the input values over [0,1) affect the efficiency of bucket sort? What assumptions must hold for bucket sort to be efficient?

Problem 18: Why Comparison-Based Sorting is $\Omega(n \log n)$

Question: Explain why any sorting algorithm that uses only comparisons must take at least $\Omega(n \log n)$ time in the worst case. Use the concept of decision trees in your explanation.

Problem 19: Sorting Lower Bound Application

Question: Suppose you are designing a new sorting algorithm that only compares elements and does not use any assumptions about the input. Can this algorithm ever beat the $\Omega(n \log n)$ lower bound? Justify your answer.