Advanced sort algorithms implementation
Due May 6th, 11:59 pm
Task1(20pts) Merge Sort implementation
Based on the slides of Advanced sorting algorithm slides

- Write a Python script with the following functions:
  - def shell_sort(a_list):
  - def gap_insertion_sort(a_list, start, gap):
  -
- Run 3 test cases with random lists in size 20, 100 and 10000
- Ensure the program sorts three lists in ascending order

Task2(20pts) Quick sort Implementation
Based on the slides of Advanced sorting algorithm slides

- Write a Python script with the following three functions:
  - quick_sort(a_list)
  - quick_sort_helper(a_list, first, last)
  - partition(a_list, first, last)
  - call the functions properly

Task3(20pts) Merge Sort Implementation
Based on the slides of Advanced sorting algorithm slides

- Write a Python script with the following three functions:
  - merge_sort(a_list)
  - merge(left, right)
-
  - Call merge function in the merge_sort

Task 4(20pts) Test() function

- create 3 random lists with size 20, 100 and 10000, value range from 1 to 100
- Use the above sorting functions to sort the 3 lists:
  - shell_sort()
  - quick_sort()
  - merge_sort()
- output instruction:
-
  - Original Random List:
    - Display the randomly generated list before sorting.
  - Sort Function Used:
    - Clearly indicate which sorting algorithm is being tested
  - Ordered List:
    - Print the resulting list after sorting.
  - Running Time:
    - Print the time taken to sort the list, in seconds, with at least 6 decimal places of precision.

Sample output for size 20
Testing list size: 20
Original List: [34, 87, 12, 55, 76, 23, 45, 67, 89, 3, 90, 21, 8, 44, 66, 10, 5, 39, 71, 18]

Sorted List: [3, 5, 8, 10, 12, 18, 21, 23, 34, 39, 44, 45, 55, 66, 67, 71, 76, 87, 89, 90]
Sort Function: Shell Sort
Time Taken: 0.000024 seconds
Sort Function: Quick Sort
Time Taken: 0.000011 seconds
Sort Function: Merge Sort
Time Taken: 0.000015 seconds
Task 5(20pts) Analyze and Modify the Quicksort Algorithm

- Define a new partition function, partition_median. The pivot is selected as the median of the first, middle, and last elements of the sublist.
- Define the quick_sort_median
- Use the new partition_median function in the quick_sort_median implementation.

Answers

```
import random
import time

# Task 1: Shell Sort Implementation
def shell_sort(a_list):
    sublist_count = len(a_list) // 2
    while sublist_count > 0:
        for start_position in range(sublist_count):
            gap_insertion_sort(a_list, start_position, sublist_count)
        sublist_count = sublist_count // 2

def gap_insertion_sort(a_list, start, gap):
    for i in range(start + gap, len(a_list), gap):
        current_value = a_list[i]
        position = i
        while position >= gap and a_list[position - gap] > current_value:
            a_list[position] = a_list[position - gap]
            position -= gap
        a_list[position] = current_value

# Task 2: Quick Sort Implementation
def quick_sort(a_list):
    quick_sort_helper(a_list, 0, len(a_list) - 1)

def quick_sort_helper(a_list, first, last):
    if first < last:
```

```
        split_point = partition(a_list, first, last)
        quick_sort_helper(a_list, first, split_point - 1)
        quick_sort_helper(a_list, split_point + 1, last)

def partition(a_list, first, last):
    pivot_value = a_list[first]
    left_mark = first + 1
    right_mark = last

    done = False
    while not done:
        while left_mark <= right_mark and a_list[left_mark] <= pivot_value:
            left_mark += 1
        while a_list[right_mark] >= pivot_value and right_mark >= left_mark:
            right_mark -= 1
        if right_mark < left_mark:
            done = True
        else:
            a_list[left_mark], a_list[right_mark] = a_list[right_mark], a_list[left_mark]

    a_list[first], a_list[right_mark] = a_list[right_mark], a_list[first]
    return right_mark

# Task 3: Merge Sort Implementation
def merge_sort(a_list):
    if len(a_list) <= 1:
        return a_list
    mid = len(a_list) // 2
    left_half = merge_sort(a_list[:mid])
    right_half = merge_sort(a_list[mid:])
    return merge(left_half, right_half)

def merge(left, right):
    result = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    result.extend(left[i:])
    result.extend(right[j:])
```

```
    return result

# Task 5: Quicksort with Median-of-Three Pivot
def partition_median(a_list, first, last):
    mid = (first + last) // 2
    pivot_candidates = [(a_list[first], first), (a_list[mid], mid), (a_list[last], last)]
    pivot_candidates.sort()
    pivot_value, pivot_index = pivot_candidates[1]
    a_list[first], a_list[pivot_index] = a_list[pivot_index], a_list[first]
    return partition(a_list, first, last)

def quick_sort_median(a_list):
    def helper(lst, first, last):
        if first < last:
            split_point = partition_median(lst, first, last)
            helper(lst, first, split_point - 1)
            helper(lst, split_point + 1, last)
    helper(a_list, 0, len(a_list) - 1)

# Task 4: Testing Function
def Test():
    sizes = [20, 100, 10000]
    for size in sizes:
        print(f"\nTesting list size: {size}\n")
        rand_list = [random.randint(1, 100) for _ in range(size)]
        print("Original List:", rand_list if size == 20 else "[List too large to display]")

        for sort_name, sort_func in [
            ("Shell Sort", shell_sort),
            ("Quick Sort", quick_sort),
            ("Merge Sort", lambda lst: merge_sort(lst))
        ]:
            list_copy = rand_list[:]
            start_time = time.time()
            if sort_name == "Merge Sort":
                list_copy = sort_func(list_copy)
            else:
                sort_func(list_copy)
            end_time = time.time()
            print(f"\nSort Function: {sort_name}")
            if size == 20:
                print("Sorted List:", list_copy)
            print("Time Taken: {:.6f} seconds".format(end_time - start_time))
Test()
```