Sorting Algorithm

1) Bobble sort ($O(n^2)$)

```
def bubble_sort (arr):
    n = len (arr)
    for i in range (n):
        for j in range (0, n+i -1):
            if arr [j] > arr [j+1]:
                arr [j], arr [j+1] = arr [j+1], arr [j]

return arr
```

2) Test (different sizes):

```
Import random
def test_algorithm (sort_func):
    for size in [5, 10, 50]:
        arr = [random.randint (1, 100) for _ in range (size)]
        print ("size:", size, "Correct", sorted_arr == sorted (a
test_algorithm (bubble_sort)
```

3) quick sort

a) Q Pivot

```
Import random
def quick_sort_random (arr):
    if len (arr) <= 1:
        return arr
pivot = random.choice (arr)
left = [x for x in arr if x < pivot]
middle = [x for x in arr if x == pivot]
```

=>

```
right = [x for x in arr
if x > pivot)

return quick_sort_ran
(left) + middle + arr
```

(b) Average pivot (first + middle + last) / 3

```
def quick_sort_average(arr):
    if len(arr) <= 1:
        return arr

    first = arr[0]
    middle = arr[len(arr)//2]
    last = arr[-1]
    pivot = (first + middle + last) // 3
```

// // //

4) Merge Sort

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
```

5) Heap Sort

```
def heapify(arr, n, i):
    largest = i
    left = 2*i + 1
    right = 2*i + 2
# If left < n and arr[left] > arr[largest]:
    largest = left
# If right < n and arr[right] > arr[largest]:
    largest = right
# if largest != i
    arr[i], arr[largest] = arr[largest], arr[i]
    ⇒ heapify(arr, n, largest)
```

Problem 2: Time & Space

1) bubble sort

Time = $O(n^2)$

Space = $O(1)$

2) Quick Sort

Recurrence: $T(n) = T(n/2) + T(n/2) + O(n)$

Av Time = $O(n \log n)$

Worst Time = $O(n^2)$

Space = $O(\log n)$

3) merge Sort

Recurrence: $T(n) = 2T(n/2) + O(n)$

Time: $O(n \log n)$

Space: $O(n)$

4) heap Sort

Build heap: $O(n)$

Sorting: $O(n \log n)$

Total time: $O(n \log n)$

Space: $O(1)$