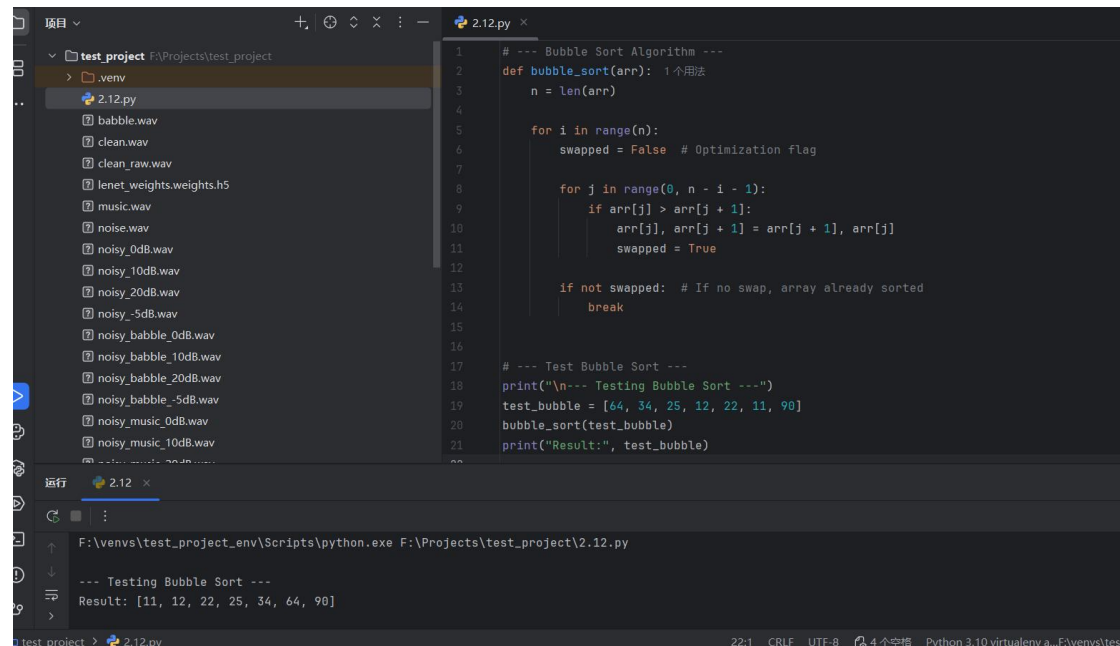


Problem 1 — Sorting Algorithms

1. write own bad $O(n^2)$ sorting (bubble sorting)



```
1 # --- Bubble Sort Algorithm ---
2 def bubble_sort(arr): 1个用法
3     n = len(arr)
4
5     for i in range(n):
6         swapped = False # Optimization flag
7
8         for j in range(0, n - i - 1):
9             if arr[j] > arr[j + 1]:
10                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
11                 swapped = True
12
13         if not swapped: # If no swap, array already sorted
14             break
15
16 # --- Test Bubble Sort ---
17 print("\n--- Testing Bubble Sort ---")
18 test_bubble = [64, 34, 25, 12, 22, 11, 90]
19 bubble_sort(test_bubble)
20 print("Result:", test_bubble)
```

运行 2.12 x

F:\venvs\test_project_env\Scripts\python.exe F:\Projects\test_project\2.12.py

```
--- Testing Bubble Sort ---
Result: [11, 12, 22, 25, 34, 64, 90]
```

Code

```
# --- Bubble Sort Algorithm ---
```

```
def bubble_sort(arr):
```

```
    n = len(arr)
```

```
    for i in range(n):
```

```
        swapped = False # Optimization flag
```

```
        for j in range(0, n - i - 1):
```

```
            if arr[j] > arr[j + 1]:
```

```
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
```

```
                swapped = True
```

```
        if not swapped: # If no swap, array already sorted
```

```
            break
```

```
# --- Test Bubble Sort ---
```

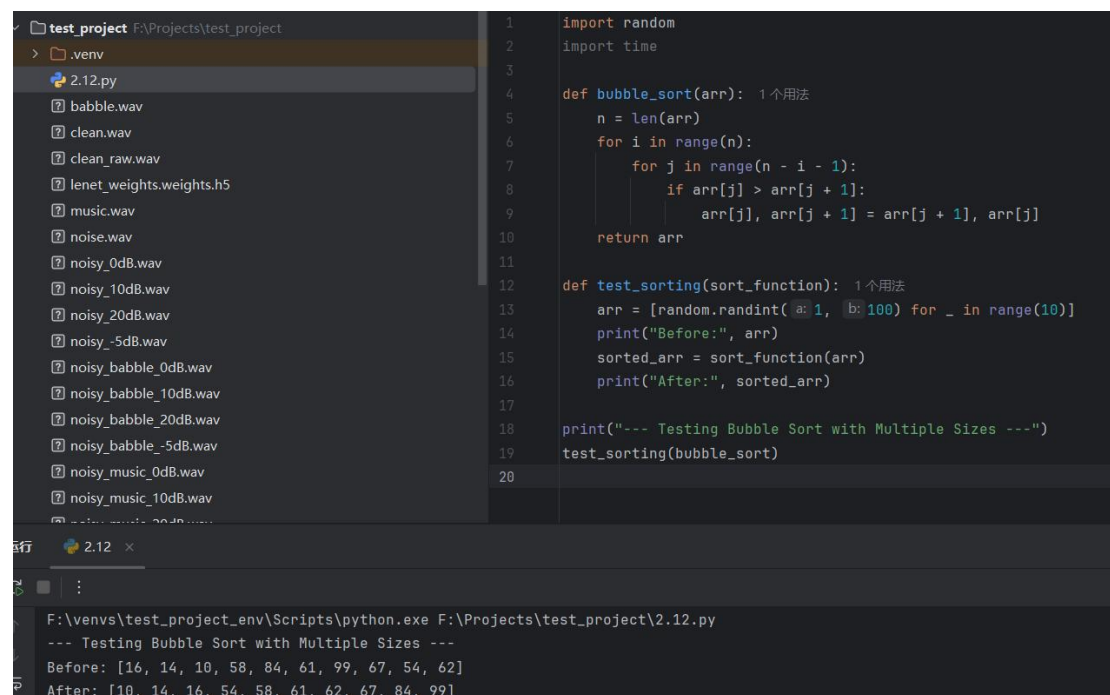
```
print("\n--- Testing Bubble Sort ---")
```

```
test_bubble = [64, 34, 25, 12, 22, 11, 90]
```

```
bubble_sort(test_bubble)
```

```
print("Result:", test_bubble)
```

2. Tests of Various Sizes



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'test_project' with a subdirectory '.venv' and a file '2.12.py'. The code editor shows the following Python code:

```
1 import random
2 import time
3
4 def bubble_sort(arr):
5     n = len(arr)
6     for i in range(n):
7         for j in range(n - i - 1):
8             if arr[j] > arr[j + 1]:
9                 arr[j], arr[j + 1] = arr[j + 1], arr[j]
10    return arr
11
12 def test_sorting(sort_function):
13     arr = [random.randint(1, 100) for _ in range(10)]
14     print("Before:", arr)
15     sorted_arr = sort_function(arr)
16     print("After:", sorted_arr)
17
18 print("--- Testing Bubble Sort with Multiple Sizes ---")
19 test_sorting(bubble_sort)
20
```

The terminal output at the bottom shows the execution of the code:

```
F:\venvs\test_project_env\Scripts\python.exe F:\Projects\test_project\2.12.py
--- Testing Bubble Sort with Multiple Sizes ---
Before: [16, 14, 10, 58, 84, 61, 99, 67, 54, 62]
After: [10, 14, 16, 54, 58, 61, 62, 67, 84, 99]
```

Code

```
import random
```

```
import time
```

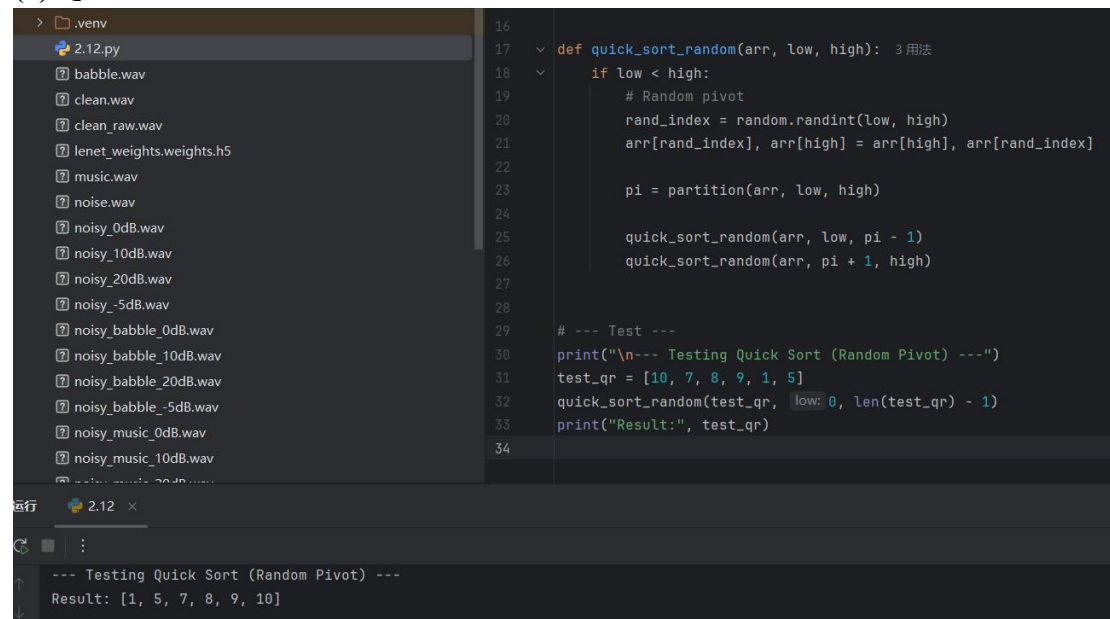
```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

```
def test_sorting(sort_function):
    arr = [random.randint(1, 100) for _ in range(10)]
    print("Before:", arr)
    sorted_arr = sort_function(arr)
    print("After:", sorted_arr)
```

```
print("--- Testing Bubble Sort with Multiple Sizes ---")
test_sorting(bubble_sort)
```

3. Quick Sort

(a) Quick Sort with Random Pivot



```
16
17 def quick_sort_random(arr, low, high): 3用法
18     if low < high:
19         # Random pivot
20         rand_index = random.randint(low, high)
21         arr[rand_index], arr[high] = arr[high], arr[rand_index]
22
23         pi = partition(arr, low, high)
24
25         quick_sort_random(arr, low, pi - 1)
26         quick_sort_random(arr, pi + 1, high)
27
28
29 # --- Test ---
30 print("\n--- Testing Quick Sort (Random Pivot) ---")
31 test_qr = [10, 7, 8, 9, 1, 5]
32 quick_sort_random(test_qr, low=0, len(test_qr) - 1)
33 print("Result:", test_qr)
34
```

运行 2.12 x

```
--- Testing Quick Sort (Random Pivot) ---
Result: [1, 5, 7, 8, 9, 10]
```

```
import random
```

```
def partition(arr, low, high):
```

```
    pivot = arr[high]
```

```
    i = low - 1
```

```
    for j in range(low, high):
```

```
        if arr[j] <= pivot:
```

```
            i += 1
```

```
            arr[i], arr[j] = arr[j], arr[i]
```

```
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
```

```
    return i + 1
```

```
def quick_sort_random(arr, low, high):
```

```
    if low < high:
```

```
        # Random pivot
```

```
        rand_index = random.randint(low, high)
```

```
        arr[rand_index], arr[high] = arr[high], arr[rand_index]
```

```
        pi = partition(arr, low, high)
```

```
        quick_sort_random(arr, low, pi - 1)
```

```
        quick_sort_random(arr, pi + 1, high)
```

```
# --- Test ---
print("\n--- Testing Quick Sort (Random Pivot) ---")
test_qr = [10, 7, 8, 9, 1, 5]
quick_sort_random(test_qr, 0, len(test_qr) - 1)
print("Result:", test_qr)
```

(b) Quick Sort with Median-of-Three Pivot

```

32
33
34 def quick_sort_median(arr, low, high): 3 用法
35     if low < high:
36         pi = partition(arr, low, high)
37         quick_sort_median(arr, low, pi - 1)
38         quick_sort_median(arr, pi + 1, high)
39
40
41 # 测试
42 print("---- Testing Quick Sort (Median-of-Three) ----")
43
44 test_qm = [random.randint(a=1, b=100) for _ in range(10)]
45 print("Before:", test_qm)
46
47 quick_sort_median(test_qm, low=0, len(test_qm) - 1)
48
49 print("After:", test_qm)
50

```

```

F:\venvs\test_project_env\Scripts\python.exe F:\Projects\test_project\2.12.py
--- Testing Quick Sort (Median-of-Three) ---
Before: [86, 77, 80, 55, 66, 43, 73, 42, 85, 17]
After: [17, 42, 43, 55, 66, 73, 77, 80, 85, 86]

```

code

```
import random

# 选择三个数的中位数作为 pivot
def median_of_three(arr, low, high):
    mid = (low + high) // 2

    if arr[low] > arr[mid]:
        arr[low], arr[mid] = arr[mid], arr[low]
    if arr[low] > arr[high]:
        arr[low], arr[high] = arr[high], arr[low]
    if arr[mid] > arr[high]:
        arr[mid], arr[high] = arr[high], arr[mid]

    # 把中位数放到 high 位置
    arr[mid], arr[high] = arr[high], arr[mid]
    return arr[high]
```

★ 这个就是你缺少的函数

```
def partition(arr, low, high):
    pivot = median_of_three(arr, low, high)
    i = low - 1

    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1
```

```
def quick_sort_median(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        quick_sort_median(arr, low, pi - 1)
        quick_sort_median(arr, pi + 1, high)
```

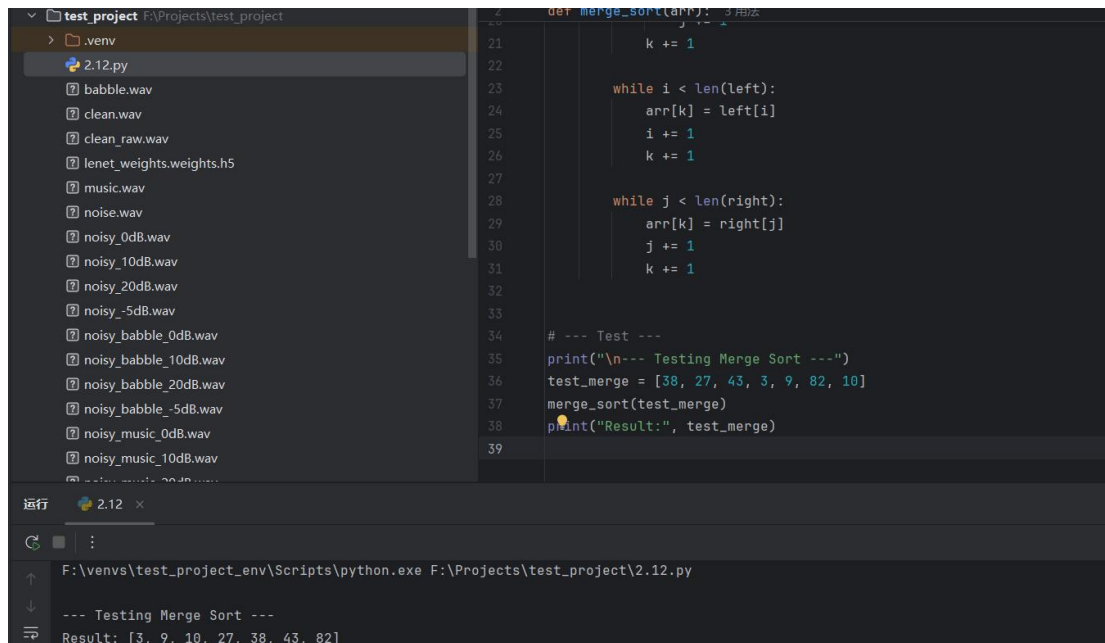
```
# test
print("--- Testing Quick Sort (Median-of-Three) ---")

test_qm = [random.randint(1, 100) for _ in range(10)]
print("Before:", test_qm)

quick_sort_median(test_qm, 0, len(test_qm) - 1)

print("After:", test_qm)
```

4. Merge Sort



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'test_project' with a subdirectory '.venv' containing a file '2.12.py'. The code editor shows the following Python code:

```
def merge_sort(arr):  
    if len(arr) > 1:  
        mid = len(arr) // 2  
        left = arr[:mid]  
        right = arr[mid:]  
        merge_sort(left)  
        merge_sort(right)  
        i = j = k = 0  
        while i < len(left) and j < len(right):  
            if left[i] < right[j]:  
                arr[k] = left[i]  
                i += 1  
            else:  
                arr[k] = right[j]  
                j += 1  
            k += 1  
        while i < len(left):  
            arr[k] = left[i]  
            i += 1  
            k += 1  
        while j < len(right):  
            arr[k] = right[j]  
            j += 1  
            k += 1  
    # --- Test ---  
    print("\n--- Testing Merge Sort ---")  
    test_merge = [38, 27, 43, 3, 9, 82, 10]  
    merge_sort(test_merge)  
    print("Result:", test_merge)
```

The output of the code is shown in the bottom panel:

```
--- Testing Merge Sort ---  
Result: [3, 9, 10, 27, 38, 43, 82]
```

code

--- Merge Sort ---

```
def merge_sort(arr):
```

```
    if len(arr) > 1:
```

```
        mid = len(arr) // 2
```

```
        left = arr[:mid]
```

```
        right = arr[mid:]
```

```
        merge_sort(left)
```

```
        merge_sort(right)
```

```
        i = j = k = 0
```

```
        while i < len(left) and j < len(right):
```

```
            if left[i] < right[j]:
```

```
                arr[k] = left[i]
```

```
                i += 1
```

```
            else:
```

```
                arr[k] = right[j]
```

```
                j += 1
```

```
            k += 1
```

```
        while i < len(left):
```

```
            arr[k] = left[i]
```

```

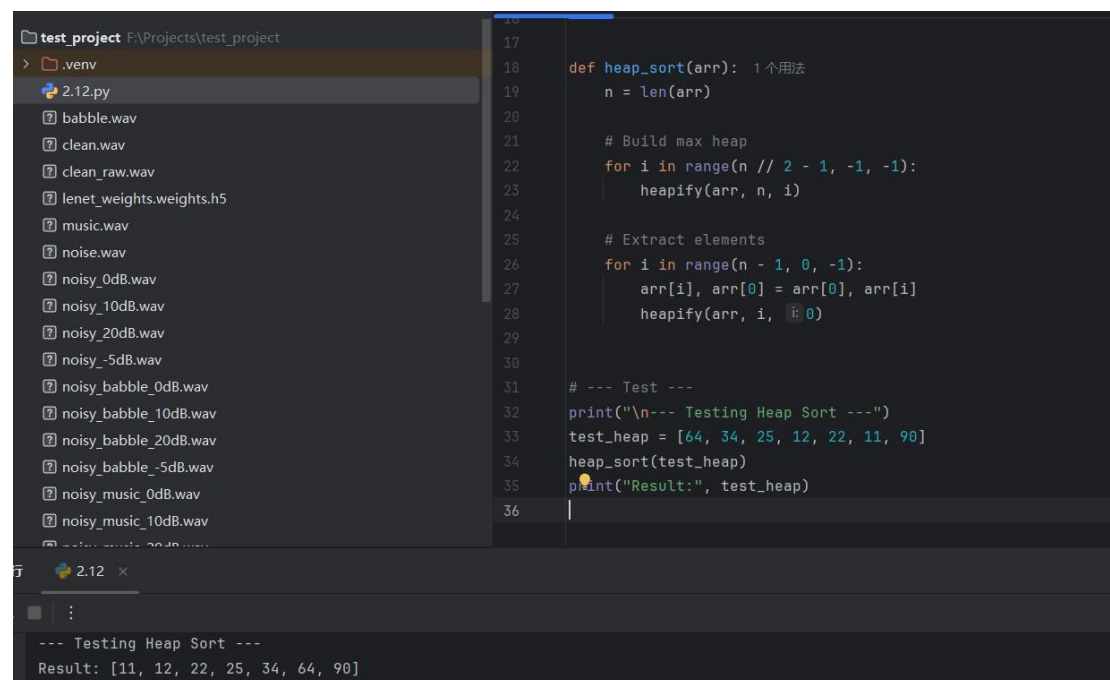
        i += 1
        k += 1

    while j < len(right):
        arr[k] = right[j]
        j += 1
        k += 1

# --- Test ---
print("\n--- Testing Merge Sort ---")
test_merge = [38, 27, 43, 3, 9, 82, 10]
merge_sort(test_merge)
print("Result:", test_merge)

```

5. Heap Sort



```

17
18 def heap_sort(arr): 1 个用法
19     n = len(arr)
20
21     # Build max heap
22     for i in range(n // 2 - 1, -1, -1):
23         heapify(arr, n, i)
24
25     # Extract elements
26     for i in range(n - 1, 0, -1):
27         arr[i], arr[0] = arr[0], arr[i]
28         heapify(arr, i, 0)
29
30
31 # --- Test ---
32 print("\n--- Testing Heap Sort ---")
33 test_heap = [64, 34, 25, 12, 22, 11, 90]
34 heap_sort(test_heap)
35 print("Result:", test_heap)
36

```

```

--- Testing Heap Sort ---
Result: [11, 12, 22, 25, 34, 64, 90]

```

code

```

# --- Heapify ---
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)

def heap_sort(arr):
    n = len(arr)

    # Build max heap
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    # Extract elements
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

# --- Test ---
print("\n--- Testing Heap Sort ---")
test_heap = [64, 34, 25, 12, 22, 11, 90]
heap_sort(test_heap)
print("Result:", test_heap)

```


Problem 2 (Analyse Sorting Algorithms). Analyse succinctly, for all sorting Algorithms above, time and space complexities using the master theorem where applicable

Problem 2. Analyse sorting Algorithms

Master Theorem

for a recurrence of the form $T(n) = aT(\frac{n}{b}) + O(n^d)$:

(a = number of subproblem)

case 1: If $d < \log_b(a)$, then $T(n) = O(n^{\log_b a})$

case 2: If $d = \log_b(a)$, then $T(n) = O(n^d \log n)$

case 3: If $d > \log_b(a)$, then $T(n) = O(n^d)$

1. Bubble Sort

Bubble Sort is not a divide-and-conquer algorithm, so

Master Theorem does not apply.

Time complexity:

Best case: $O(n)$ (already sorted, early break)

Average case: $O(n^2)$

Worst case: $O(n^2)$

space complexity:

$O(1)$, in-place sorting.

2. Quick Sort (Random Pivot)

Average case recurrence:

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

using Master Theorem:

$$a = 2$$

$$b = 2$$

$$d(n) = O(n)$$

Since $n \log_b a = n$, case 2 applies:

$$T(n) = O(n \log n)$$

worst case: $T(n) = T(n-1) + O(n) = O(n^2)$

space complexity:

Average: $O(\log n)$

Worst: $O(n)$

3. Quick Sort (Median-of-Three)

improves pivot selection, making balanced partitions more likely.

Average case: $O(n \log n)$

worst case: $O(n^2)$

space: $O(\log n)$ average

4. Merge Sort

Recurrence:

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

By Master Theorem:

$$T(n) = O(n \log n)$$

space complexity:

$O(n)$ extra temporary arrays.

5. Heap Sort

Building heap: $O(n)$

Extracting elements: $n \log n$

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

space: $O(1)$