Name:Majed Hameed Alodhaylah

ID:432105522

Solution for the Quiz2

Problem 1: Heapsort

1. Convert the array into a Max Heap (Step by Step)

Given array: [9, 5, 2, 11, 7, 6, 14, 1, 3]

Heapify Process:

1. **Heapify index 2: [9, 5, 14, 11, 7, 6, 2, 1, 3]**
2. **Heapify index 1: [9, 11, 14, 5, 7, 6, 2, 1, 3]**
3. **Heapify index 0: [14, 11, 9, 5, 7, 6, 2, 1, 3]**

Max Heap: [14, 11, 9, 5, 7, 6, 2, 1, 3]

---

2. Perform Heapsort (Step by Step)

Sorting Process:

1. **Swap 14 ↔ 3, Heapify → [11, 7, 9, 5, 3, 6, 2, 1, 14]**
2. **Swap 11 ↔ 1, Heapify → [9, 7, 6, 5, 3, 1, 2, 11, 14]**
3. **Swap 9 ↔ 2, Heapify → [7, 5, 6, 2, 3, 1, 9, 11, 14]**
4. **Swap 7 ↔ 1, Heapify → [6, 5, 1, 2, 3, 7, 9, 11, 14]**
5. **Swap 6 ↔ 3, Heapify → [5, 3, 1, 2, 6, 7, 9, 11, 14]**
6. **Swap 5 ↔ 2, Heapify → [3, 2, 1, 5, 6, 7, 9, 11, 14]**
7. **Swap 3 ↔ 1, Heapify → [2, 1, 3, 5, 6, 7, 9, 11, 14]**
8. **Swap 2 ↔ 1, Heapify → [1, 2, 3, 5, 6, 7, 9, 11, 14]**

Sorted Array: [1, 2, 3, 5, 6, 7, 9, 11, 14]

---

3. Worst-case Time Complexity of Heapsort

O(nlogn)O(n \log n)

---

4. When is Heapsort Preferred Over Quicksort?

When worst-case performance matters (O(n log n) vs. QuickSort's worst-case $O(n^2)$)
When memory is limited (Heapsort is in-place, QuickSort uses recursion stack)
For real-time systems (more predictable execution time)

Problem 2: Counting Sort with Negative Numbers

1. Modify Counting Sort to Handle Negative Numbers and Sort the Given Array

Given array: [-5, -10, 0, -3, 8, 5, -1, 10]

Modified Counting Sort Approach:

1. **Find min = -10, max = 10, shift values by +10**
2. **Apply Counting Sort**
3. **Shift values back**

Sorted Array: [-10, -5, -3, -1, 0, 5, 8, 10]

---

2. Why is Counting Sort Inefficient When the Range is Too Large?

Consumes too much memory if the range is large

Example: Sorting numbers between -1,000,000 to 1,000,000 requires an array of size 2,000,001, which is impractical.

---

3. Is Counting Sort Suitable for 1 Million Integers Ranging from -100,000 to 100,000? Why?

No, because it requires too much extra space (200,001 slots).

Better alternatives: Radix Sort, Merge Sort, or QuickSort.

Problem 3: Radix Sort vs. Merge Sort

1. Why is Radix Sort Good for Sorting 1 Million 9-Digit Integers? Radix Sort runs in O(d(n+k))O(d(n + k)), making it efficient for fixed-length numbers.

For 9-digit integers, it processes them digit-by-digit, avoiding costly comparisons.

---

2. How Many Passes Are Needed in Base k=10k=10 vs. Base k=256k=256?

Base 10 → 9 passes
Base 256 → 3-4 passes

Larger bases reduce the number of passes but require more memory.

---

3. Sort [234, 455, 224, 323, 123] Using Radix Sort (Base 10) Step by Step

Step 1: Sort by 1s digit

[224, 455, 123, 234, 323]

Step 2: Sort by 10s digit

[123, 224, 234, 323, 455]

Step 3: Sort by 100s digit

[123, 224, 234, 323, 455]

Final Sorted Array: [123, 224, 234, 323, 455]

---

4. Which is Better for Sorting 100 Million Integers: Radix Sort or Merge Sort? Why?

Radix Sort is better for fixed-length numbers (digits-based sorting).
Merge Sort is better for large arbitrary data (works for variable-length numbers).

Conclusion: Radix Sort is faster when sorting large numbers with a fixed number of digits.
