*Owen Lindsey*

*Professor Demland, David*

*CST-201 Exercise 8*

*10/27/2024*

**Exercise 5.1 - 6**

*Algorithm Explanation*

1. *Input List:*

*{"E", "X", "A", "M", "P", "L", "E"}*

1. *MergeSort Process Core Concepts:*

***Divide:*** *Split array into two halves recursively until individual elements*

***Conquer:*** *Sort and merge smaller subarrays back together*

***Combine:*** *Merge sorted subarrays while maintaining order*

1. ***Division Process:***

**Level 0 (Initial Array):**  
[E, X, A, M, P, L, E]  
  
**Level 1 (Split into two halves):**  
Left half: [E, X, A, M] Right half: [P, L, E]  
  
**Level 2 (Split each half again):**  
Left quarter: [E, X] Right quarter: [A, M] Left quarter: [P, L] Right quarter: [E]  
  
**Level 3 (Individual elements):**  
[E] [X] [A] [M] [P] [L] [E]

*Merging Process:*

**Level 1 (Merging pairs):**  
[E] [X] → **Compare** E,X → [E, X]  
[A] [M] → **Compare** A,M → [A, M]  
[P] [L] → **Compare** P,L → [L, P]  
[E] **remains single** → [E]  
  
**Level 2 (Merging groups of 2):**  
[E, X] merged with [A, M]:  
- **Compare** E with A: A is smaller → A  
- **Compare** E with M: E is smaller → E  
- **Compare** X with M: M is smaller → M  
- X **remains** → X  
**Result:** [A, E, M, X]  
  
[L, P] merged with [E]:  
- **Compare** L with E: E is smaller → E  
- L **remains** → L  
- P **remains** → P  
**Result:** [E, L, P]  
  
Level 3 (Final merge):  
[A, E, M, X] merged with [E, L, P]:  
- **Compare** A with E: A is smaller → A  
- **Compare** E with E: Equal, take both → E, E  
- **Compare** M with L: L is smaller → L  
- **Compare** M with P: M is smaller → M  
- **Compare** X with P: P is smaller → P  
- X **remains** → X  
**Final Result:** [A, E, E, L, M, P, X]

**Comparison Rules:**

*When merging, always compare the front elements of each subarray. Choose smaller element to place in merged array. If equal, maintain relative order (stability), continue until all elements are processed.*

**Exercise 5.1 - 8**

**Analyzing MergeSort's**

**A) Worst-Case Key Comparison Analysis**

***Initial Setup:***

*Let's understand why we get T(n) = 2T(n/2) + (n-1) during each merge operation of two n/2 sized arrays we must*  *compare elements until we exhaust at least one subarray. In worst case, we alternate between arrays, requiring (n-1)*  *comparisons.*

***Example with [3,5,7] and [2,4,6]:***

***Compare*** *3:2 → take 2 (1 comparison)*

***Compare*** *3:4 → take 3 (2 comparisons)*

***Compare*** *5:4 → take 4 (3 comparisons)*

**Solution:**

*T(n) = 2T(n/2) + (n-1), where n = 2ᵏ*

**Expanding one level:**  
*T(2ᵏ) = 2[2T(2ᵏ⁻²) + 2ᵏ⁻¹-1] + 2ᵏ-1*  
 *= 4T(2ᵏ⁻²) + 2ᵏ + 2ᵏ⁻¹-2*  
  
**Expanding further:**  
**Level 1:** *2ᵏ + (2ᵏ⁻¹-2)*  
**Level 2:** *2ᵏ + 2ᵏ⁻¹ + (2ᵏ⁻²-4)*  
**Level k:** *Sum becomes n log₂n - n + 1*

**B) Best-Case Analysis**

**Consider merging [1,2,3] and [4,5,6]:**

*Only need n/2 comparisons because, after comparing first elements (1:4) we know all elements in first array < second*  *array can directly copy remaining elements.*

**Progression:**

*B(n) = 2B(n/2) + n/2*  
**For n** = *2ᵏ:*  
**Level 1:** *2ᵏ⁻¹*  
**Level 2:** *2ᵏ⁻¹ + 2(2ᵏ⁻²)*  
**Level k:** *Results in (n/2)log₂n*

**C) Key Moves**

**Why moves are consistent:**

*During the merger phase each element must be copied to an auxiliary array, then copied back to the original array.*

**Analysis:**

*M(n) = 2M(n/2) + n*  
*Solving for n = 2ᵏ:*  
**Level 1:** *n moves*  
**Level 2:** *n moves*  
  
**Level k**: *n moves*  
**Total:** *n × log₂n moves*

*The key moves do not change the algorithm's overall efficiency class, it remains O(n log n).*

**Exercise 5.2 - 1**

QuickSort process:

1. **Input List:**

{"E", "X", "A", "M", "P", "L", "E"}

1. **Quicksort Core Concepts:**

**Partition:** *Choose pivot and arrange elements around it.*

**Recursion:** *Apply quicksort to subarrays independently.*

**In-place:** *Sort occurs within original array space.*

1. **Partitioning Rules:**

**Select Pivot:** *Choose first element.*

**Compare:** *Each element against pivot.*

**Swap:** *Move elements ≤ pivot to left, > pivot to right.*

**Place Pivot:** *Put pivot between partitions.*

1. **Step-by-Step:** *First Partition:*

**Initial:**

[E, X, A, M, P, L, E]

**Compare** E with X:

X > E, keep X right

**Compare** E with A:

A < E, swap → [A, X, E, M, P, L, E]

**Compare** E with M:

M > E, keep M right

**Compare** E with P:

P > E, keep P right

**Compare** E with L:

L > E, keep L right

**Compare** E with E:

E = E, place next to first E

**Result:**

[A, E, E] | [X, M, P, L]

**Second Level Partitions:**

**Left:** [A, E, E] → [A] | [E, E] (already sorted)

**Right**: [X, M, P, L] → [M, P, L] | [X]

**Third Level Partition:**

[M, P, L] → [L] | [M] | [P]

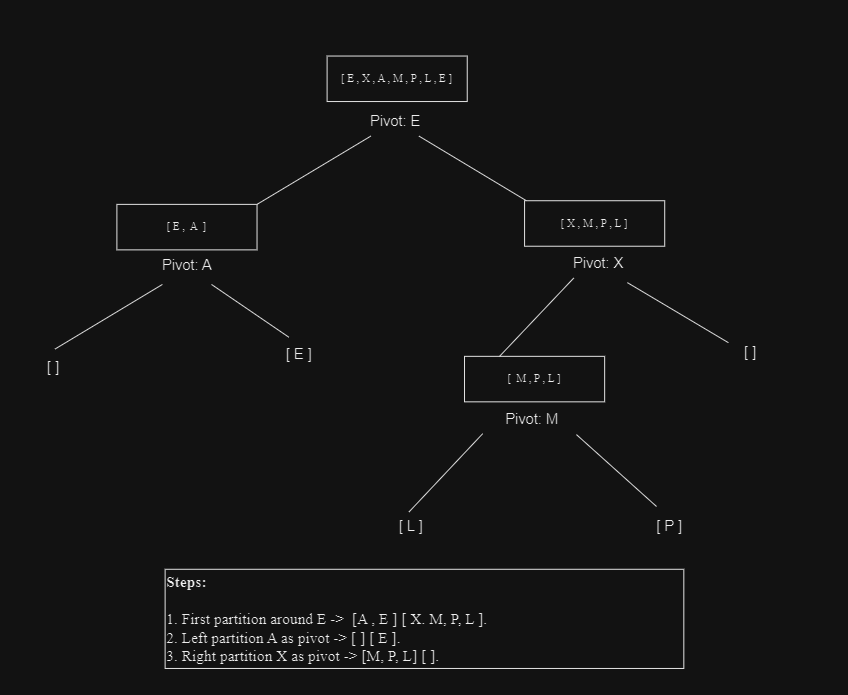
**Final Assembly Process:**

**Combine** [A] + [E, E]

**Combine** result with [L] + [M] + [P] + [X]

**Result:** [A, E, E, L, M, P, X]

**Exercise 5.2 - 1**

QuickSort tree diagram:   


**Exercise 5.2 - 5**

Quicksort Case Analysis:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Type | Case analysis | Reason | Example | Performance |
| All Equal Elements | BEST CASE | - Partitions create balanced splits  - s ≈ n/2 at each step  - Optimal recursion depth | [E,E,E,E] → [E,E] : [E,E] → [E] : [E] : [E] : [E] | O(n log n) |
| Strictly Decreasing | WORST CASE | - Most unbalanced splits  - One element in right partition  - Maximum recursion depth | [5,4,3,2,1] → [4,3,2,1] : [5] → [3,2,1] : [4] : [5] | O(n²) |

**Final Answers:**

1. Arrays of all equal elements: BEST CASE.

* Creates balanced partitions every time.
* Achieves optimal log n recursion levels.
* Each level requires n operations.

1. Strictly decreasing arrays: WORST CASE.

* Creates maximally unbalanced partitions.
* Requires n levels of recursion.
* Each level still requires n operations.