Merge Sort

Jianguo Lu

January 22, 2025

Overview

Divide and Conquer

Merge sort

Divide and Conquer

Divide If the input size is smaller than a certain threshold (say, one or two elements), solve the problem directly using a straightforward method and return the solution so obtained. Otherwise, divide the input data into two or more disjoint subsets.

Conquer Recursively solve the subproblems associated with the subsets.

Combine Take the solutions to the subproblems and merge them into a solution to the original problem

Merge sort

Merge sort German folk dance (youTube)

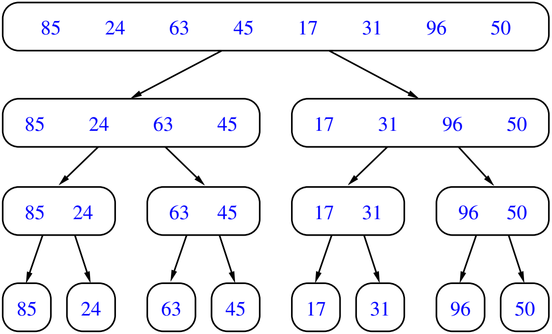
Illustration of Merge sort algorithm

6 5 3 1 8 7 2 4

(Animation to be viewed with Adobe Reader)

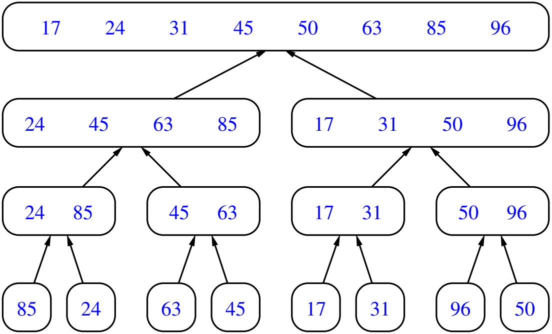
Overall Idea of Merge Sort

Divide

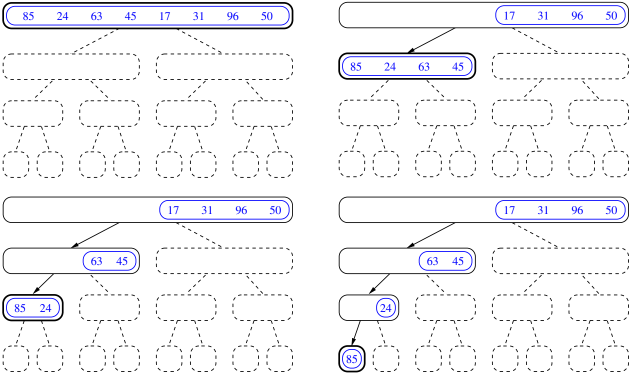


Overall Idea of Merge Sort

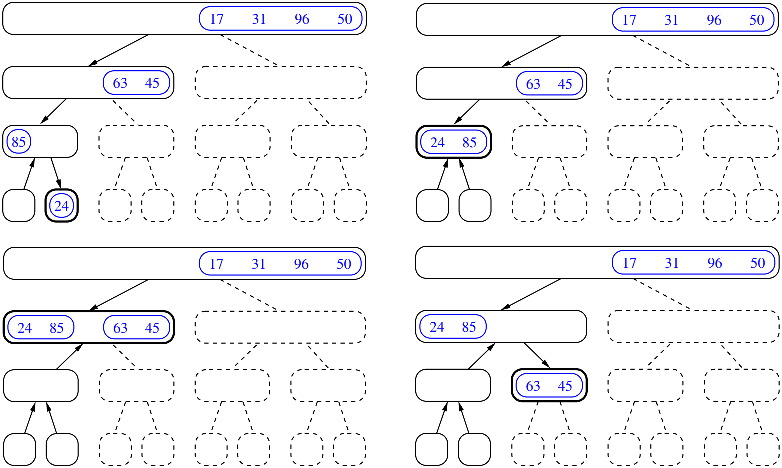
Combine

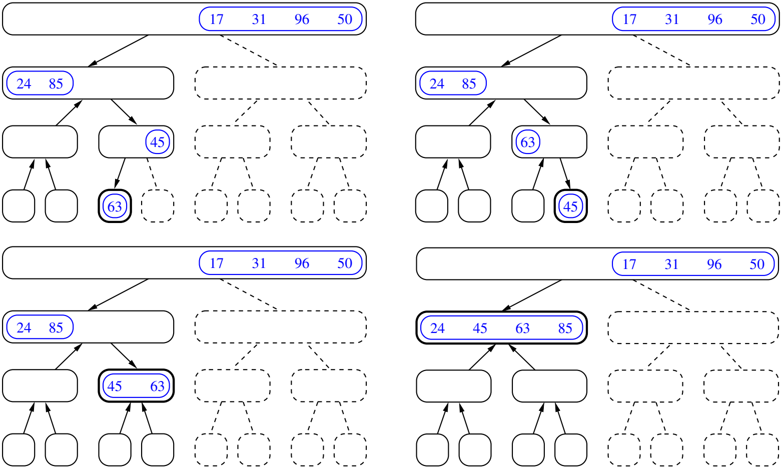


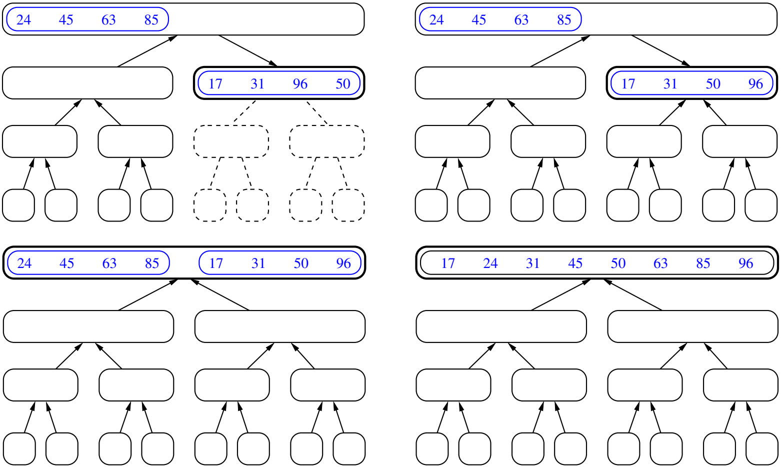
Merge-sort Example Step by Step



Merge-sort Example Step by Step




17 31 96 50
24 85
45
63
17 31 96 50
24 85
63
45
17 31 96 50
24 85
45 63
17 31 96 50
24 45 63 85

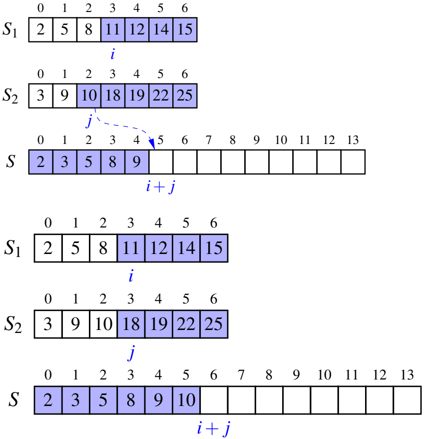

24 45 63 85
17 31 96 50
24 45 63 85
17 31 50 96
24 45 63 85 17 31 50 96
17 24 31 45 50 63 85 96

Merge sort in Array

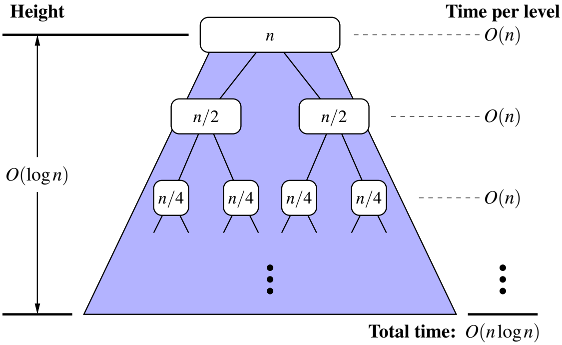
```
public static<K>void mergeSort(K[] S,Comparator<K> comp){
    int n = S.length;
    if (n < 2) return;
    int mid = n/2;
    K[] S1 = Arrays.copyOfRange(S, 0, mid);
    K[] S2 = Arrays.copyOfRange(S, mid, n);
    mergeSort(S1, comp);
    mergeSort(S2, comp);
    merge(S1, S2, S, comp);
}
```

Merge operation

```
public static<K> void merge(K[] S1,K[] S2,K[] S,Comparator<K>comp)
  int i = 0, j = 0;
  while (i + j < S.length) {
    if (j==S2.length||(i<S1.length && comp.compare(S1[i], S2[j])<0))
        S[i+j] = S1[i++];
    else  S[i+j] = S2[j++];
  }
```

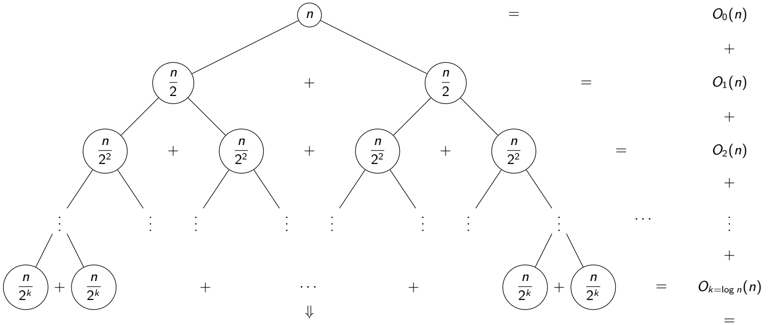


Analysis of merge sort: recursion tree



- The height h of the merge-sort tree is $O(\log n)$

Recursion tree



$$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{k} n\right) = O(k \cdot n) \Leftrightarrow O(n \cdot \log n)$$

Analysis using the substitution method

1. Write the recurrence relation

$$T(n) = \begin{cases} 1, & \text{if n=1} \\ 2T(n/2) + n, & \text{if } n > 1 \end{cases} \tag{1}$$

2. Guess its closed-form upper bound

$$T(n) = O(n \log n) \tag{2}$$

   i.e., there exists a constant c such that

$$T(n) \leq cn \log n \tag{3}$$

3. Prove it using mathematical induction

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

It is equivalent to proving that there exists a constant c such that

$$T(n) \leq cn \log n \tag{4}$$

Proof using mathematical induction

- Base case: when n=1. it is trivially true: $T(1) = O(1)$.
- Induction step: suppose that it is true for n/2. i.e.,

$$T(n/2) \leq c(n/2)\log(n/2) \tag{5}$$

We need to prove that it is also true for n. i.e.,

$$T(n) \leq cn\log(n) \tag{6}$$

The proof goes as follows:

$$\begin{aligned}
T(n) &= 2T(n/2) + n && (7)\\
&\leq 2c(n/2)\log(n/2) + n && \text{(by induction assumption)}\\
&= cn(\log n - log2) + n && (\log(n/2) = \log n - log2)\\
&= cn\log n - cn + n && (\log 2 = 1)\\
&= cn\log n - n(c-1) && \\
&\leq cn\log n && (\text{when } c \geq 1)
\end{aligned}$$

- Readings: Goodrich et al. Chapter 12.1. P.532-540