Exercise: Scheduling Problem with Min Weighted Completion Time

Scheduling Problem

Input: Given are n jobs each $i \in [n]$ has a weight (or the importance) w_i and the length (or the time required) l_i. We define the completion time c_i of job i to be the sum of the lengths of jobs in the ordering up to and including l_i.

Output: An ordering of jobs that minimizes the weighted sum of completion times $\sum_{i \in [n]} w_i c_i$.

- Example: Given are 5 jobs with the following weights and lengths:

	1	2	3	4	5
weight	2	6	5	4	2
length	5	4	10	8	3

CSE 431/531: Algorithm Analysis and Design (Fall 2023)
Divide-and-Conquer

Lecturer: Kelin Luo

Department of Computer Science and Engineering
University at Buffalo

Outline

Greedy Algorithm

- mainly for combinatorial optimization problems
- trivial algorithm runs in exponential time
- greedy algorithm gives an efficient algorithm
- main focus of analysis: correctness of algorithm

Greedy Algorithm

- mainly for combinatorial optimization problems
- trivial algorithm runs in exponential time
- greedy algorithm gives an efficient algorithm
- main focus of analysis: correctness of algorithm

Divide-and-Conquer

- not necessarily for combinatorial optimization problems
- trivial algorithm already runs in polynomial time
- divide-and-conquer gives a more efficient algorithm
- main focus of analysis: running time

Divide-and-Conquer

- **Divide**: Divide instance into many smaller instances
- **Conquer**: Solve each of smaller instances recursively and separately
- **Combine**: Combine solutions to small instances to obtain a solution for the original big instance

Divide-and-Conquer

- **Divide**: Divide instance into many smaller instances
- **Conquer**: Solve each of smaller instances recursively and separately
- **Combine**: Combine solutions to small instances to obtain a solution for the original big instance

Running time analysis

- recursive programs: recurrence

merge-sort(A, n)

if $n = 1$ **then**
    return A
else
    $B \leftarrow$ merge-sort$\Big(A\big[1..\lfloor n/2\rfloor\big], \lfloor n/2\rfloor\Big)$
    $C \leftarrow$ merge-sort$\Big(A\big[\lfloor n/2\rfloor + 1..n\big], \lceil n/2\rceil\Big)$
    return merge$(B, C, \lfloor n/2\rfloor, \lceil n/2\rceil)$

merge-sort(A, n)

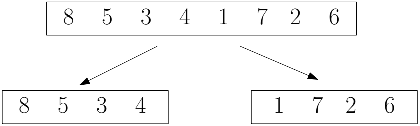
1: **if** $n = 1$ **then**
2:     **return** A
3: **else**
4:     $B \leftarrow \text{merge-sort}\Big(A\big[1..\lfloor n/2\rfloor\big], \lfloor n/2\rfloor\Big)$
5:     $C \leftarrow \text{merge-sort}\Big(A\big[\lfloor n/2\rfloor + 1..n\big], \lceil n/2\rceil\Big)$
6:     **return** merge$(B, C, \lfloor n/2\rfloor, \lceil n/2\rceil)$

- Divide: trivial
- Conquer: 4, 5
- Combine: 6

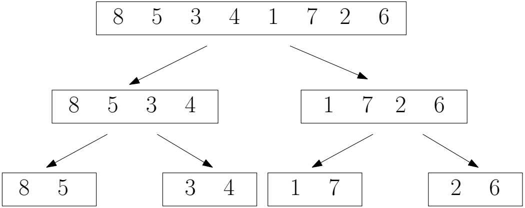
merge-sort()

8	5	3	4	1	7	2	6

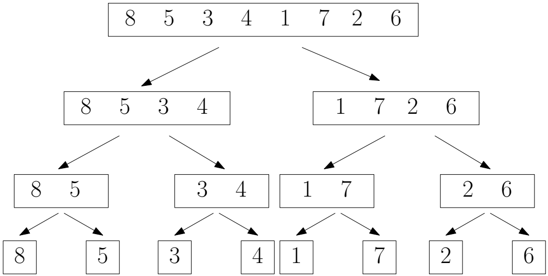
merge-sort()



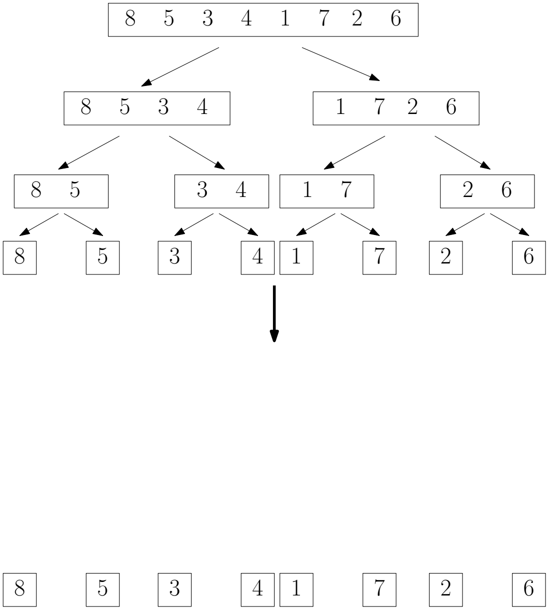
merge-sort()



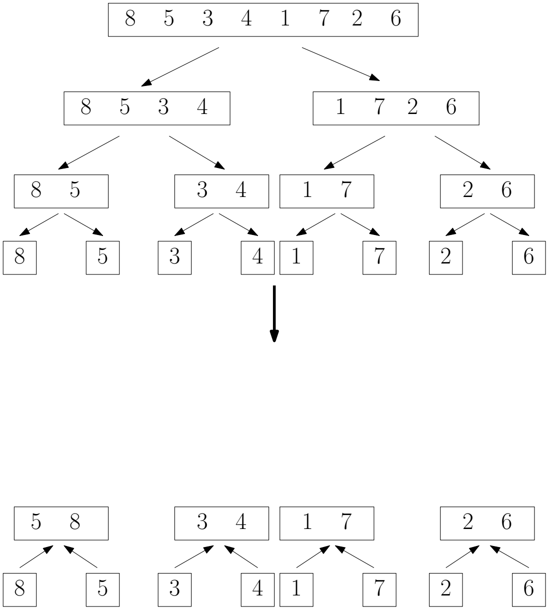
merge-sort()



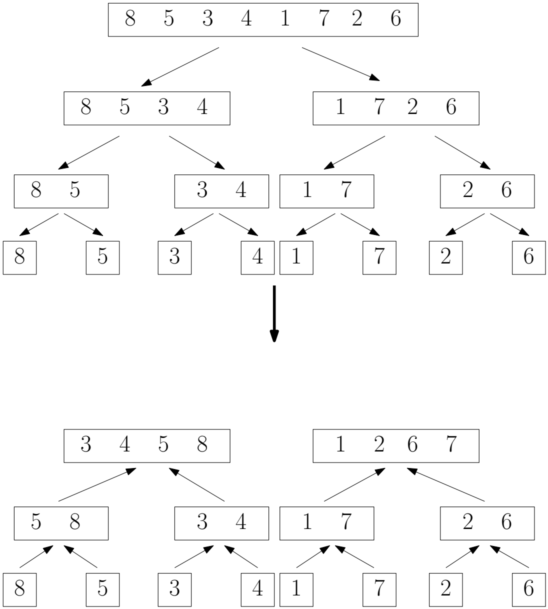
merge-sort()



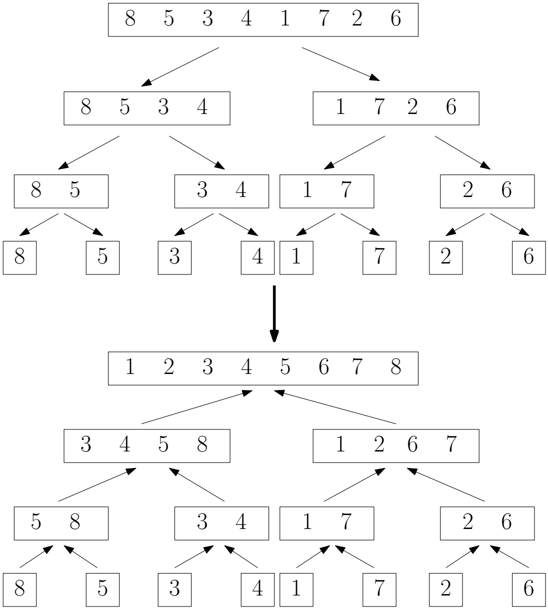
merge-sort()



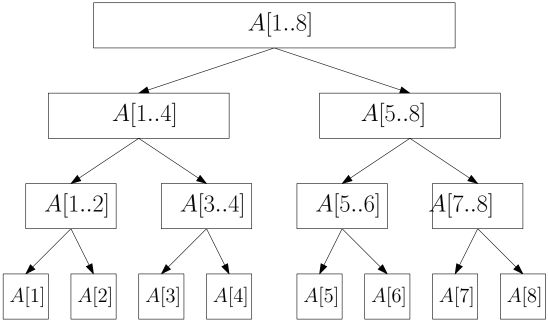
merge-sort()


8 5 3 4 1 7 2 6
8 5 3 4
1 7 2 6
8 5
3 4
1 7
2 6
8
5
3
4
1
7
2
6
3 4 5 8
1 2 6 7
5 8
3 4
1 7
2 6
8
5
3
4
1
7
2
6

merge-sort()


8 5 3 4 1 7 2 6
8 5 3 4
1 7 2 6
8 5
3 4
1 7
2 6
8
5
3
4
1
7
2
6
1 2 3 4 5 6 7 8
3 4 5 8
1 2 6 7
5 8
3 4
1 7
2 6
8
5
3
4
1
7
2
6

Running Time for Merge-Sort



- Each level takes running time $O(n)$
- There are $O(\lg n)$ levels
- Running time $= O(n \lg n)$
- Better than insertion sort

Running Time for Merge-Sort

Implementation

- Divide $A[a, b]$ by $q = \lfloor (a + b)/2 \rfloor$: $A[a, q]$ and $A[q + 1, b]$; or $A[a, q - 1]$ and $A[q, b]$?

Running Time for Merge-Sort

Implementation

- Divide $A[a, b]$ by $q = \lfloor (a + b)/2 \rfloor$: $A[a, q]$ and $A[q + 1, b]$; or $A[a, q - 1]$ and $A[q, b]$?
- Speed-up: avoid the constant copying from one layer to another and backward

Running Time for Merge-Sort

Implementation

- Divide $A[a,b]$ by $q = \lfloor (a+b)/2 \rfloor$: $A[a,q]$ and $A[q+1,b]$; or $A[a,q-1]$ and $A[q,b]$?
- Speed-up: avoid the constant copying from one layer to another and backward
- Speed-up: stop the dividing process when the sequence sizes fall below constant