

Subject Code · Subject Name

Semester / Session

Lecture 1: Title of the Topic

Lecturer: Name of the Professor

Compiled: October 24, 2026

Learning objectives

After this lecture you should be able to:

- state and apply the divide-and-conquer paradigm;
- analyze recurrences using recursion trees and induction;
- implement merge sort and Dijkstra's algorithm.

Contents

1	Introduction	1
2	Mathematical Foundations	2
3	Algorithms	2
4	Code	2
5	Figures and Plots	4
6	Exercises	4

1 Introduction

Divide and conquer solves a problem by splitting it into smaller subproblems, solving them recursively, and combining the results. Inline math such as $T(n) = 2T(n/2) + \mathcal{O}(n)$ and inline code such as `merge_sort(A)` sit comfortably together. Cross-references work as usual: see Section 2, Algorithm 1, Listing 1, and Figure 1.

Key idea. Many efficient algorithms follow one pattern: *split*, *recurse*, *merge*. The cost is determined by how the work at each level of the recursion adds up.

2 Mathematical Foundations

Definition 2.1: Big-O notation

Let $f, g : \mathbb{N} \rightarrow \mathbb{R}_{\geq 0}$. We write $f(n) = \mathcal{O}(g(n))$ if there exist $c > 0$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq c g(n)$ for all $n \geq n_0$.

Theorem 2.2: Merge sort recurrence

Let $T(n) = 2T(n/2) + n$ with $T(1) = 1$ for n a power of two. Then $T(n) = n \log_2 n + n = \Theta(n \log n)$.

Proof. We argue by induction on k where $n = 2^k$. For $k = 0$, $T(1) = 1 = 1 \cdot 0 + 1$. Assume the claim for $n/2$. Then

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2\left(\frac{n}{2} \log_2 \frac{n}{2} + \frac{n}{2}\right) + n \\ &= n(\log_2 n - 1) + n + n \\ &= n \log_2 n + n. \end{aligned}$$

■

Example 2.3: Gradient descent step

For a differentiable $f : \mathbb{R}^d \rightarrow \mathbb{R}$ the update with step size $\eta > 0$ is

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta \nabla f(\mathbf{x}_t), \quad \mathbf{x}^* \in \arg \min_{\mathbf{x}} f(\mathbf{x}).$$

Note. Use `\begin{theorem}{Title}{label}` and refer to it with `Theorem \ref{thm:label}`. For instance, Theorem 2.2 above.

A small table with booktabs:

Table 1: Common running times.

Algorithm	Time	Space
Merge sort	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$
Quicksort (avg)	$\mathcal{O}(n \log n)$	$\mathcal{O}(\log n)$
Dijkstra (heap)	$\mathcal{O}((V + E) \log V)$	$\mathcal{O}(V)$

3 Algorithms

Lemma 3.1: Correctness invariant

When a vertex u is extracted from Q , $d[u]$ equals the true shortest-path distance $\delta(s, u)$.

4 Code

Algorithm 1 Dijkstra's shortest paths**Input:** Graph $G = (V, E)$, weights $w : E \rightarrow \mathbb{R}_{\geq 0}$, source $s \in V$ **Output:** Shortest-path distances $d[v]$ from s to every $v \in V$

```

1:  $d[v] \leftarrow \infty$  for all  $v \in V$ ;  $d[s] \leftarrow 0$ 
2:  $Q \leftarrow$  priority queue containing all  $v \in V$  keyed by  $d[v]$ 
3: while  $Q \neq \emptyset$  do
4:    $u \leftarrow \text{EXTRACTMIN}(Q)$ 
5:   for all edges  $(u, v) \in E$  do
6:     if  $d[u] + w(u, v) < d[v]$  then
7:        $d[v] \leftarrow d[u] + w(u, v)$ 
8:        $\text{DECREASEKEY}(Q, v, d[v])$ 
9: return  $d$ 

```

```

1 def merge_sort(a):
2     """Return a sorted copy of list 'a' in O(n log n) time."""
3     if len(a) <= 1:
4         return a
5     mid = len(a) // 2
6     left, right = merge_sort(a[:mid]), merge_sort(a[mid:])
7     out, i, j = [], 0, 0
8     while i < len(left) and j < len(right): # merge step
9         if left[i] <= right[j]:
10            out.append(left[i]); i += 1
11        else:
12            out.append(right[j]); j += 1
13    return out + left[i:] + right[j:]

```

Listing 1: Merge sort in Python.

```

1 #include <vector>
2
3 // Returns the index of x in sorted vector a, or -1 if absent.
4 int binary_search(const std::vector<int>& a, int x) {
5     int lo = 0, hi = (int)a.size() - 1;
6     while (lo <= hi) {
7         int mid = lo + (hi - lo) / 2;
8         if (a[mid] == x) return mid;
9         if (a[mid] < x) lo = mid + 1;
10        else hi = mid - 1;
11    }
12    return -1;
13 }

```

Listing 2: Binary search in C++.

Pitfall. Writing $(lo + hi) / 2$ can overflow for large indices; use $lo + (hi - lo) / 2$ as in Listing 2.

5 Figures and Plots

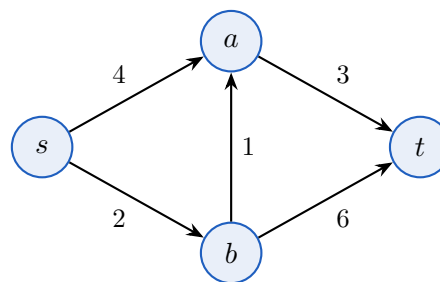


Figure 1: A small weighted digraph. The shortest s - t path is $s \rightarrow b \rightarrow a \rightarrow t$ with cost 6.

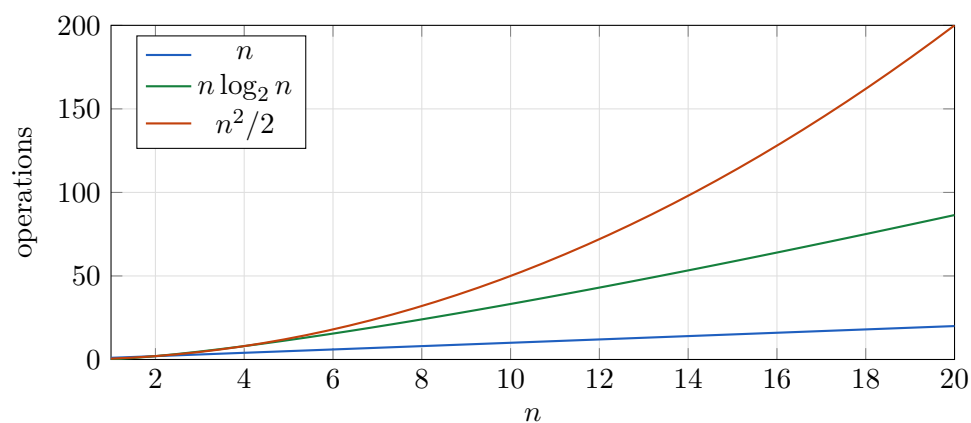


Figure 2: Growth rates of common running times.

6 Exercises

Exercise 6.1: Recurrence

Solve $T(n) = 3T(n/2) + n$ with $T(1) = 1$ using a recursion tree, and give a tight Θ -bound.

Exercise 6.2: Implementation

Modify Listing 1 to count the number of inversions in $\mathcal{O}(n \log n)$ time.

Further Reading

- T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Introduction to Algorithms*, MIT Press.
- J. Kleinberg, É. Tardos. *Algorithm Design*, Pearson.