Algorithm

1 Foundations **2**
1.1 Computational Problems and Algorithms 2

2 Sorting **3**
2.1 Insertion Sort 3
2.2 Heap Sort 4

3 Divide and Conquer **5**
3.1 Selection 5

10 Shortest Paths **6**
10.1 Single-Source Shortest Paths 6

Chapter 1

Foundations

1.1 Computational Problems and Algorithms

Definition 1.1. A **computational problem** is a relation

$$P \subseteq X \times Y,$$

where X is called the set of **instances** and Y is called the sets of **solutions**.

Definition 1.2. We will assume the **random-access machine (RAM)** model of computation as our implementaion technology for most of this note. In this model, we have an infinite sequence of w-bit words, and we assume $w = \lceil c \lg n \rceil$ for some constant $c \geq 1$, where n is the input size. We can perform some basic operations on these words, including

- arithmetic operations (e.g., addition, subtraction, multiplication, division),
- data movement operations (e.g., load, store, copy), and
- control operations (e.g., branch, subroutine call, return).

Definition 1.3. Given a computational model, an **algorithm** is defined as a finite sequence of basic operations that transforms a given input into a unique output.

- We say that an algorithm **solves** a computational problem $P \subseteq X \times Y$ if it transforms every instance $x \in X$ into a solution $y \in Y$ such that $(x, y) \in P$.
- The **running time** of an algorithm on a specific input is defined as the number of basic operations performed.

Chapter 2

Sorting

2.1 Insertion Sort

Problem 2.A (Sorting Problem).

- Input: An array A of n numbers.
- Output: A permutation of A that is nondecreasing.

```
Insertion-Sort(A)
n ← |A|
for i ← 2 to n
    τ ← A[i]
    j ← i
    while j > 1 and A[j − 1] > τ
        A[j] ← A[j − 1]
        j ← j − 1
    A[j] ← τ
```

2.2 Heap Sort

Definition 2.1. An array A of n numbers can be viewed as a complete binary tree, where each node of the tree corresponds to an element of A.

- $A[1]$ is the root.
- The left child of $A[i]$ is $A[2i]$ (if $2i \leq n$).
- The right child of $A[i]$ is $A[2i+1]$ (if $2i+1 \leq n$).

We say that A is **heapified** if

$$A\left[\left\lfloor \frac{i}{2} \right\rfloor\right] \geq A[i]$$

holds for each $i \in \{2, \ldots, n\}$.

```
HEAPIFY-DOWN(A, i)
n ← |A|
φ ← TRUE
while φ
    ℓ ← 2i
    r ← 2i + 1
    j ← i
    if ℓ ≤ n and A[ℓ] > A[j]
        j ← ℓ
    if r ≤ n and A[r] > A[j]
        j ← r
    if j = i
        φ ← FALSE
    else
        swap A[i] and A[j]
        i ← j
```

```
HEAPIFY-UP(A, i)
while i > 1 and A[i] > A[⌊i/2⌋]
    swap A[i] and A[⌊i/2⌋]
    i ← ⌊i/2⌋
```

```
HEAP-SORT(A)
n ← |A|
for i ← ⌊n/2⌋ downto 1
    HEAPIFY-DOWN(A, i)
for j ← n downto 2
    swap A[1] and A[j]
    HEAPIFY-DOWN(A[1 . . j − 1], 1)
```

Chapter 3

Divide and Conquer

3.1 Selection

Problem 3.A (Selection Problem).

- Input: An array A of n numbers and an integer k with $1 \leq k \leq n$.
- Output: The kth smallest number of A.

```
PARTITION(A)
n ← |A|
i ← 1
for j ← 1 to n − 1
    if A[j] ≤ A[n]
        swap A[i] and A[j]
        i ← i + 1
swap A[i] and A[n]
return i
```

```
SELECT(A, i)
n ← |A|
if n ≤ 5
    INSERTION-SORT(A)
else
    ℓ ← ⌊n/5⌋
    for i ← 1 to ℓ
        INSERTION-SORT(A[(5i − 4) . . 5i])
        swap A[i] and A[5i − 2]
    m ← ⌈ℓ/2⌉
    SELECT(A[1 . . ℓ], m)
    swap A[m] and A[n]
    j ← PARTITION(A)
    if j > i
        SELECT(A[1 . . j − 1], i)
    elseif j < i
        SELECT(A[j + 1 . . n], i − j)
```

Chapter 10

Shortest Paths

10.1 Single-Source Shortest Paths

```
BELLMAN-FORD(G, w, s)
n ← |V(G)|
for each u ∈ V(G)
    u.d ← ∞
    u.π ← NIL
s.d ← 0
for i ← 1 to n − 1
    for each u ∈ V(G)
        for each v ∈ N_G(u)
            if v.d > u.d + w(u, v)
                v.d ← u.d + w(u, v)
                v.π ← u
for each u ∈ V(G)
    for each v ∈ N_G(u)
        if v.d > u.d + w(u, v)
            return FALSE
return TRUE
```

```
DIJKSTRA(G, w, s)
for each u ∈ V(G)
    u.d ← ∞
    u.π ← NIL
s.d ← 0
Q ← V(G)
while Q ≠ ∅
    u ← EXTRACT-MIN(Q)
    for each v ∈ N_G(u)
        if v.d > u.d + w(u, v)
            v.d ← u.d + w(u, v)
            v.π ← u
```