

# Algorithm

<b>1</b>	<b>Foundations</b>	<b>2</b>
1.1	Computational Problems and Algorithms . . . . .	2
1.2	Asymptotic Notations . . . . .	3
<b>2</b>	<b>Sorting</b>	<b>4</b>
2.1	Insertion Sort . . . . .	4
2.2	Heapsort . . . . .	6
<b>3</b>	<b>Divide and Conquer</b>	<b>7</b>
3.1	Selection . . . . .	7
<b>10</b>	<b>Shortest Paths</b>	<b>8</b>
10.1	Single-Source Shortest Paths . . . . .	8

# 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 implementation 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.

## 1.2 Asymptotic Notations

**Definition 1.4.** Let  $f(n)$  and  $g(n)$  be functions.

- We write  $f(n) = O(g(n))$  if there exists positive constants  $c$  and  $n_0$  such that

$$0 \leq f(n) \leq cg(n)$$

holds for any integer  $n \geq n_0$ .

- We write  $f(n) = \Omega(g(n))$  if  $g(n) = O(f(n))$ .
- We write  $f(n) = \Theta(g(n))$  if  $f(n) = O(g(n))$  and  $g(n) = O(f(n))$ .
- We write  $f(n) = o(g(n))$  if for any real number  $c > 0$ , there exists a positive constant  $n_0$  such that

$$0 \leq f(n) < cg(n)$$

holds for any integer  $n \geq n_0$ .

- We write  $f(n) = \omega(g(n))$  if  $g(n) = o(f(n))$ .

# Chapter 2

## Sorting

### 2.1 Insertion Sort

In this chapter, we focus on the sorting problem. An algorithm that solves the sorting problem is usually called a sorting algorithm.

**Problem 2.A (Sorting Problem).**

- Input: An array  $A[1..n]$  of numbers.
- Output: A permutation of  $A$  that is nondecreasing.

**Algorithm 2.1.** INSERTION-SORT is an efficient sorting algorithm if the size of input array is small.

INSERTION-SORT( $A[1..n]$ )

```
1  for  $i \leftarrow 2$  to  $n$ 
2       $\tau \leftarrow A[i]$ 
3       $j \leftarrow i$ 
4       $\phi \leftarrow \text{TRUE}$ 
5      while  $\phi$ 
6          if  $j = 1$  or  $A[j - 1] \leq \tau$ 
7               $\phi \leftarrow \text{FALSE}$ 
8          else
9               $A[j] \leftarrow A[j - 1]$ 
10              $j \leftarrow j - 1$ 
11      $A[j] \leftarrow \tau$ 
```

**Theorem 2.2.** The algorithm INSERTION-SORT correctly solves the sorting problem.

*Proof.* We prove the loop invariant that at the start of each iteration of the **for** loop of lines 1 – 11, the subarray  $A[1..i - 1]$  is a nondecreasing permutation of the elements originally in  $A[1..i - 1]$ . The loop invariant is trivially true for  $i = 2$ , and we show that each iteration maintains the loop invariant.

First, we set  $\tau \leftarrow A[i]$  and  $j \leftarrow i$ . Then the **while** loop of lines 5 – 10 maintains the loop invariant that at the start of each iteration,  $A[1..j - 1]$  remains unchanged, and the elements in  $A[j + 1..i]$  are the elements originally in  $A[j..i - 1]$ , each at its corresponding position. It can be shown that when the **while** loop of lines 5 – 10

terminates, each element in  $A[1 \dots j - 1]$  is less than or equal to  $A[j]$ , and each element in  $A[j + 1 \dots i]$  is greater than  $A[j]$ . Thus, after we set  $A[j] \leftarrow \tau$ , the subarray  $A[1 \dots i]$  becomes a sorted permutation of the elements originally in  $A[1 \dots i]$ , implying that the loop invariant holds after the increment of  $i$ .

When the **for** loop of lines 1 – 11 terminates, we have  $i = n + 1$ . Due to the loop invariant, the entire array is a nondecreasing permutation of the original input array, which completes the proof.  $\square$

**Theorem 2.3.** The worst-case running time of INSERTION-SORT is  $\Theta(n^2)$ .

*Proof.* It is easy to verify that the **while** loop of lines 5 – 10 takes  $O(i)$  time. Thus, the overall running time is  $O(n^2)$ .

However, if the input array is strictly decreasing, then the **while** loop of lines 5 – 10 will take  $\Omega(i)$  time. In this case, the overall running time is  $\Omega(n^2)$ . Thus, the worst-case running time of INSERTION-SORT is  $\Theta(n^2)$ .  $\square$

## 2.2 Heapsort

**Definition 2.4.** A **binary heap** is a complete binary tree such that the value of each node is not less than the values of its children.

We can use an array to represent a complete binary tree, such that  $A[1]$  is the root of the tree, and  $A[2i]$  and  $A[2i + 1]$  are the left child and the right child of  $A[i]$ .

**Algorithm 2.5.** Suppose that  $A[1..n]$  is an array representing a complete binary tree. If the subtrees rooted at  $A[2i]$  and  $A[2i + 1]$  are already heapified, then we can use HEAPIFY-DOWN to heapify the subtree rooted at  $A[i]$ .

HEAPIFY-DOWN( $A[1..n], i$ )

```
1   $\phi \leftarrow \text{TRUE}$ 
2  while  $\phi$ 
3       $\ell \leftarrow 2i$ 
4       $r \leftarrow 2i + 1$ 
5       $j \leftarrow i$ 
6      if  $\ell \leq n$  and  $A[\ell] > A[j]$ 
7           $j \leftarrow \ell$ 
8      if  $r \leq n$  and  $A[r] > A[j]$ 
9           $j \leftarrow r$ 
10     if  $j = i$ 
11          $\phi \leftarrow \text{FALSE}$ 
12     else
13         swap  $A[i]$  and  $A[j]$ 
14          $i \leftarrow j$ 
```

HEAPIFY-UP( $A[1..n], i$ )

```
1   $j \leftarrow \lfloor i/2 \rfloor$ 
2  while  $j \geq 1$  and  $A[i] > A[j]$ 
3      swap  $A[i]$  and  $A[j]$ 
4       $i \leftarrow j$ 
5       $j \leftarrow \lfloor j/2 \rfloor$ 
```

HEAPSORT( $A[1..n]$ )

```
1  for  $i \leftarrow \lfloor n/2 \rfloor$  downto 1
2      HEAPIFY-DOWN( $A, i$ )
3  for  $j \leftarrow n$  downto 2
4      swap  $A[1]$  and  $A[j]$ 
5      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  $k$ th smallest number of  $A$ .

PARTITION( $A$ )

```
1   $n \leftarrow |A|$ 
2   $i \leftarrow 1$ 
3  for  $j \leftarrow 1$  to  $n - 1$ 
4      if  $A[j] \leq A[n]$ 
5          swap  $A[i]$  and  $A[j]$ 
6           $i \leftarrow i + 1$ 
7  swap  $A[i]$  and  $A[n]$ 
8  return  $i$ 
```

SELECT( $A, i$ )

```
1   $n \leftarrow |A|$ 
2  if  $n \leq 5$ 
3      INSERTION-SORT( $A$ )
4  else
5       $\ell \leftarrow \lfloor n/5 \rfloor$ 
6      for  $i \leftarrow 1$  to  $\ell$ 
7          INSERTION-SORT( $A[(5i - 4) \dots 5i]$ )
8          swap  $A[i]$  and  $A[5i - 2]$ 
9       $m \leftarrow \lceil \ell/2 \rceil$ 
10     SELECT( $A[1 \dots \ell], m$ )
11     swap  $A[m]$  and  $A[n]$ 
12      $j \leftarrow \text{PARTITION}(A)$ 
13     if  $j > i$ 
14         SELECT( $A[1 \dots j - 1], i$ )
15     elseif  $j < i$ 
16         SELECT( $A[j + 1 \dots n], i - j$ )
```

# Chapter 10

## Shortest Paths

### 10.1 Single-Source Shortest Paths

BELLMAN-FORD( $G, w, s$ )

```
1   $n \leftarrow |V(G)|$ 
2  for each  $u \in V(G)$ 
3       $u.d \leftarrow \infty$ 
4       $u.\pi \leftarrow \text{NIL}$ 
5   $s.d \leftarrow 0$ 
6  for  $i \leftarrow 1$  to  $n - 1$ 
7      for each  $u \in V(G)$ 
8          for each  $v \in N_G(u)$ 
9              if  $v.d > u.d + w(u, v)$ 
10                  $v.d \leftarrow u.d + w(u, v)$ 
11                  $v.\pi \leftarrow u$ 
12 for each  $u \in V(G)$ 
13     for each  $v \in N_G(u)$ 
14         if  $v.d > u.d + w(u, v)$ 
15             return FALSE
16 return TRUE
```

DIJKSTRA( $G, w, s$ )

```
1  for each  $u \in V(G)$ 
2       $u.d \leftarrow \infty$ 
3       $u.\pi \leftarrow \text{NIL}$ 
4   $s.d \leftarrow 0$ 
5   $Q \leftarrow V(G)$ 
6  while  $Q \neq \emptyset$ 
7       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
8      for each  $v \in N_G(u)$ 
9          if  $v.d > u.d + w(u, v)$ 
10              $v.d \leftarrow u.d + w(u, v)$ 
11              $v.\pi \leftarrow u$ 
```