



THE PUBLIC IS MORE FAMILIAR WITH BAD DESIGN THAN GOOD DESIGN. IT IS, IN EFFECT, CONDITIONED TO PREFER BAD DESIGN, BECAUSE THAT IS WHAT IT LIVES WITH. THE NEW BECOMES THREATENING, THE OLD REASSURING.

PAUL RAND

A DESIGNER KNOWS THAT HE HAS ACHIEVED PERFECTION NOT WHEN THERE IS NOTHING LEFT TO ADD, BUT WHEN THERE IS NOTHING LEFT TO TAKE AWAY.

ANTOINE DE SAINT-EXUPÉRY

...THE DESIGNER OF A NEW SYSTEM MUST NOT ONLY BE THE IMPLEMENTOR AND THE FIRST LARGE-SCALE USER; THE DESIGNER SHOULD ALSO WRITE THE FIRST USER MANUAL...IF I HAD NOT PARTICIPATED FULLY IN ALL THESE ACTIVITIES, LITERALLY HUNDREDS OF IMPROVEMENTS WOULD NEVER HAVE BEEN MADE, BECAUSE I WOULD NEVER HAVE THOUGHT OF THEM OR PERCEIVED WHY THEY WERE IMPORTANT.

DONALD E. KNUTH

HENGFENG WEI

# COLLECTION OF ALGO- RITHMS PSEUDOCODE

ANT

Copyright © 2018 Hengfeng Wei

PUBLISHED BY ANT

TUFTE-LATEX.GOOGLECODE.COM

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

*First printing, April 2018*

# *Contents*

*Basic Iterative and Recursive Algorithms*      15

*Sorting*      17

*Selection*      19

*Dynamic Programming*      21

*Traversal on Trees*      23

*Bibliography*      27



## *List of Figures*

- 1 3-element sorting algorithm represented by a decision tree. 17
- 2 3-element median selection algorithm represented by a decision tree. 19
- 3 Calculate  $\binom{4}{2}$  recursively. 21
- 4 Pascal triangle for binomial coefficients. 21





## *List of Tables*



*Dedicated to those who appreciate  $\text{\LaTeX}$   
and the work of Edward R. Tufte and Donald E. Knuth.*



# *Introduction*

This is a collection of psuedocode for classic algorithms.



## *Basic Iterative and Recursive Algorithms*

---

**Algorithm 1** Horner rule for polynomial evaluation.

---

```
1: procedure HORNER( $A[0 \dots n], x$ )  $\triangleright A : \{a_0 \dots a_n\}$ 
2:    $p \leftarrow A[n]$ 
3:   for  $i \leftarrow n - 1$  downto 0 do
4:      $p \leftarrow px + A[i]$ 
5:   return  $p$ 
```

---

---

**Algorithm 2** Integer Multiplication.

---

```
1: procedure INT-MULT( $y, z$ )
2:   if  $z = 0$  then
3:     return 0
4:   return INT-MULT( $cy, \lfloor \frac{z}{c} \rfloor$ ) +  $y(z \bmod c)$ 
```

---





# Sorting

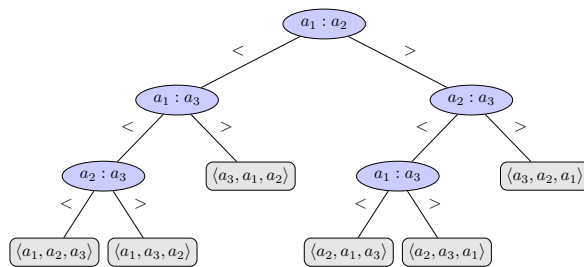


Figure 1: 3-element sorting algorithm represented by a decision tree.

---

## Algorithm 3 Selection Sort.

---

```

1: procedure SELECTION-SORT( $A, n$ )
2:   for  $i \leftarrow 1$  to  $n - 1$  do
3:     for  $j \leftarrow i + 1$  to  $n$  do
4:       if  $A[j] < A[i]$  then
5:         SWAP( $A[j], A[i]$ )

```

---



## *Selection*

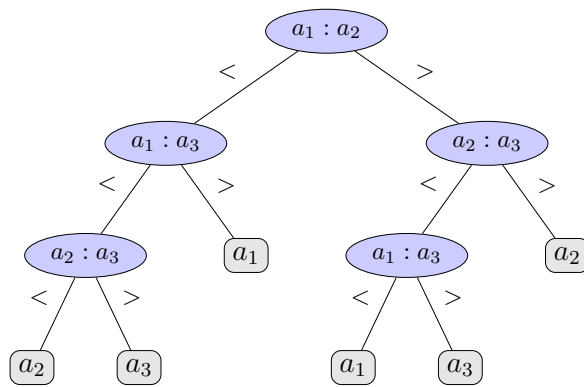


Figure 2: 3-element median selection algorithm represented by a decision tree.



# Dynamic Programming

---

**Algorithm 4** Computing  $\binom{n}{k}$  recursively.

---

```

1: procedure BINOM( $n, k$ ) ▷ Required:  $n \geq k \geq 0$ 
2:   if  $k = 0 \vee n = k$  then
3:     return 1
4:   return BINOM( $n - 1, k$ ) + BINOM( $n - 1, k - 1$ )

```

---

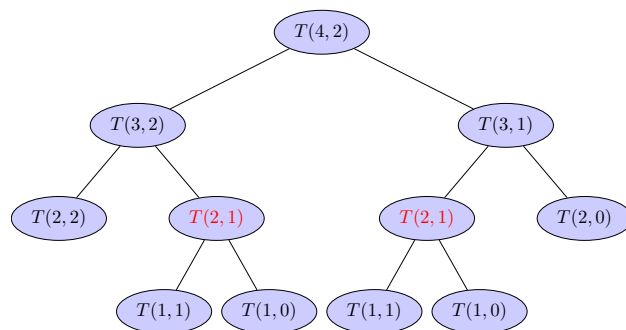


Figure 3: Calculate  $\binom{4}{2}$  recursively.

$$\begin{array}{cccccc}
 & & & & \binom{0}{0} & & & & \\
 & & & & \binom{1}{0} & & \binom{1}{1} & & \\
 & & & \binom{2}{0} & & \binom{2}{1} & & \binom{2}{2} & \\
 & & \binom{3}{0} & & \binom{3}{1} & & \binom{3}{2} & & \binom{3}{3} \\
 & \binom{4}{0} & & \binom{4}{1} & & \binom{4}{2} & & \binom{4}{3} & & \binom{4}{4} \\
 \binom{5}{0} & & \binom{5}{1} & & \binom{5}{2} & & \binom{5}{3} & & \binom{5}{4} & & \binom{5}{5}
 \end{array}$$

Figure 4: Pascal triangle for binomial coefficients.

---

**Algorithm 5** Computing  $\binom{n}{k}$  by dynamic programming.

---

```

1: procedure BINOM( $n, k$ )                                ▷ Required:  $n \geq k \geq 0$ 
2:   for  $i \leftarrow 0$  to  $n - k$  do
3:      $B[i][0] \leftarrow 1$ 
4:   for  $i \leftarrow 1$  to  $k$  do
5:      $B[i][i] \leftarrow 1$ 
6:   for  $j \leftarrow 1$  to  $k$  do
7:     for  $d \leftarrow 1$  to  $n - k$  do
8:        $i \leftarrow j + d$ 
9:        $B[i][j] \leftarrow B[i - 1][j] + B[i - 1][j - 1]$ 
10:  return  $B[n][k]$ 

```

---

$$(n - k + 1) + (k) + k(n - k) = nk - k^2 + n + 1$$

# Traversal on Trees

## DFS on Trees

---

**Algorithm 6** Calculate the sum of depths of all nodes of a tree  $T$ .

---

```
1: procedure SUM-OF-DEPTHS()
2:   return SUM-OF-DEPTHS( $T, 0$ )

3: procedure SUM-OF-DEPTHS( $r, depth$ )           ▷  $r$ : root of a tree
4:   if  $r$  is a leaf then
5:     return  $depth$ 

6:    $sum \leftarrow depth$ 
7:   for all child vertex  $v$  of  $r$  do
8:      $sum \leftarrow sum + \text{SUM-OF-DEPTHS}(v, depth + 1)$ 
9:   return  $sum$ 
```

---

---

**Algorithm 7** Count the number of nodes in  $T$  at depth  $K$ .

---

```
1: procedure NODES-AT-DEPTH()
2:   return NODES-AT-DEPTH( $T, K$ )

3: procedure NODES-AT-DEPTH( $r, k$ )           ▷  $r$ : root of a tree
4:   if  $k = 0$  then
5:     return 1

6:   if  $r$  is a leaf then
7:     return 0

8:    $num \leftarrow 0$ 
9:   for all child vertex  $v$  of  $r$  do
10:     $num \leftarrow num + \text{NODES-AT-DEPTH}(v, k - 1)$ 
11:  return  $num$ 
```

---

---

**Algorithm 8** Check whether a tree  $T$  has any leaf at an even depth.

---

```

1: procedure LEAF-AT-EVEN-DEPTH()
2:   return LEAF-AT-DEPTH( $T, \text{even} = 0$ )

3: procedure LEAF-AT-DEPTH( $r, \text{parity}$ )            $\triangleright r$ : root of a tree
4:   if  $r$  is a leaf then
5:     return  $1 - \text{parity}$ 

6:    $\text{result} \leftarrow 0$ 
7:   for all child vertex  $v$  of  $r$  do
8:      $\text{result} \leftarrow \text{result} \vee \text{LEAF-AT-DEPTH}(v, 1 - \text{parity})$ 
9:   return  $\text{result}$ 

```

---

### BFS on Trees

---

**Algorithm 9** Calculate the sum of contents of nodes of a tree  $T$  at each depth.

---

```

1: procedure SUM-AT-DEPTH( $r$ )            $\triangleright r$ : root of the tree  $T$ 
2:    $r.\text{depth} \leftarrow 0$ 

3:    $Q \leftarrow \emptyset$ 
4:   ENQUEUE( $Q, r$ )

5:   while  $Q \neq \emptyset$  do
6:      $u \leftarrow \text{DEQUEUE}(Q)$ 
7:      $\text{sumAtDepth}[u.\text{depth}] += u.\text{content}$ 

8:     for all child vertex  $v$  of  $u$  do
9:        $v.\text{depth} \leftarrow u.\text{depth} + 1$ 
10:    ENQUEUE( $Q, v$ )

```

---



---

**Algorithm 10** Count the number of nodes of a tree  $T$  at each depth.

---

```

1: procedure NODES-AT-DEPTH( $r$ )            $\triangleright r$ : root of the tree  $T$ 
2:    $r.depth \leftarrow 0$ 
3:    $Q \leftarrow \emptyset$ 
4:   ENQUEUE( $Q, r$ )
5:   while  $Q \neq \emptyset$  do
6:      $u \leftarrow$  DEQUEUE( $Q$ )
7:      $nodesAtDepth[u.depth] += 1$ 
8:     for all child vertex  $v$  of  $u$  do
9:        $v.depth \leftarrow u.depth + 1$ 
10:      ENQUEUE( $Q, v$ )
11:  return  $\text{argmax}_K nodesAtDepth[k]$ 

```

---



## *Bibliography*