

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 17

Sorting 19

Selection 21

Searching 23

Dynamic Programming 25

Traversal on Trees 27

Bibliography 31

List of Algorithms

1	Horner rule for polynomial evaluation.	17
2	Integer Multiplication.	17
3	Bubblesort (Bubble the smallest element each time.).	19
4	Bubblesort (Bubble the largest element each time.).	19
5	Bubblesort (An improved version; from wiki).	20
6	Selection Sort.	20
7	Iterative binary search.	23
8	Recursive binary search.	24
9	Computing $\binom{n}{k}$ recursively.	25
10	Computing $\binom{n}{k}$ by dynamic programming.	26
11	Max-sum subarray.	26
12	Max-sum subarray (Implementation Simplified).	26
13	Calculate the sum of depths of all nodes of a tree T .	27
14	Count the number of nodes in T at depth K .	27
15	Check whether a tree T has any leaf at an even depth.	28
16	Calculate the sum of contents of nodes of a tree T at each depth.	28
17	Count the number of nodes of a tree T at each depth.	29

List of Figures

- 1 3-element sorting algorithm represented by a decision tree. 19
- 2 3-element median selection algorithm represented by a decision tree. 21
- 3 Binary stride. 23
- 4 Calculate $\binom{4}{2}$ recursively. 25
- 5 Pascal triangle for binomial coefficients. 25

List of Tables

*Dedicated to those who appreciate \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( $y, z$ )  $\triangleright y, z \geq 0; y, z \in \mathbb{Z}$ 
2:   if  $z = 0$  then
3:     return 0
4:   return INT-MULT( $cy, \lfloor \frac{z}{c} \rfloor$ ) +  $y(z \bmod c)$ 
```

Sorting

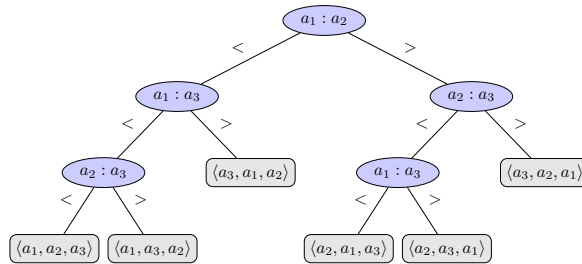


Figure 1: 3-element sorting algorithm represented by a decision tree.

Algorithm 3 Bubblesort (Bubble the smallest element each time.).

```

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

```

Algorithm 4 Bubblesort (Bubble the largest element each time.).

```

1: procedure BUBBLESORT( $A[1 \dots n]$ )
2:   for  $i \leftarrow n$  downto  $2$  do
3:     for  $j \leftarrow 1$  to  $i - 1$  do
4:       if  $A[j] > A[j + 1]$  then
5:         SWAP( $A[j], A[j + 1]$ )

```

Algorithm 5 Bubblesort (An improved version; from wiki).

```
1: procedure BUBBLESORT( $A[1 \cdots n]$ )
2:   repeat
3:      $newn \leftarrow 0$ 
4:     for  $i \leftarrow 1$  to  $n - 1$  do
5:       if  $A[i - 1] > A[i]$  then
6:         SWAP( $A[i - 1], A[i]$ )
7:          $newn \leftarrow i$ 
8:      $n \leftarrow newn$ 
9:   until  $n = 0$ 
```

Algorithm 6 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

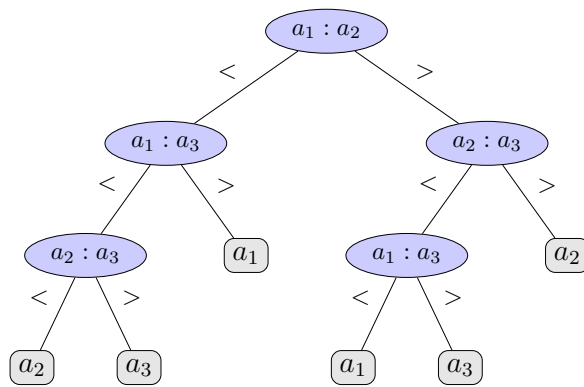


Figure 2: 3-element median selection algorithm represented by a decision tree.

Searching

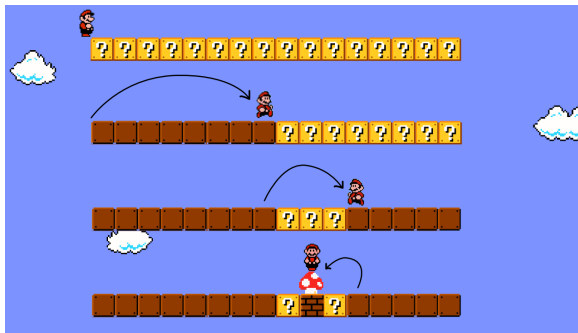


Figure 3: Binary stride.

Algorithm 7 Iterative binary search.

```

1: procedure BINARYSEARCH( $A[0 \cdots n - 1], x$ )
2:    $L \leftarrow 0$ 
3:    $R \leftarrow n - 1$ 
4:   while  $L \leq R$  do
5:      $m \leftarrow L + (R - L) / 2$ 
6:     if  $A[m] < x$  then
7:        $L \leftarrow m + 1$ 
8:     else if  $A[m] > x$  then
9:        $R \leftarrow m - 1$ 
10:    else
11:      return  $m$ 
12:  return  $-1$ 

```

Algorithm 8 Recursive binary search.

```
1: procedure BINARYSEARCH( $A, L, R, x$ )  
2:   if  $R < L$  then  
3:     return  $-1$   
  
4:    $m \leftarrow L + (R - L) / 2$   
5:   if  $A[m] = x$  then  
6:     return  $m$   
7:   else if  $A[m] > x$  then  
8:     return BINARYSEARCH( $A, L, m - 1, x$ )  
9:   else  
10:    return BINARYSEARCH( $A, m + 1, R, x$ )
```

Dynamic Programming

Algorithm 9 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( $n - 1, k - 1$ )

```

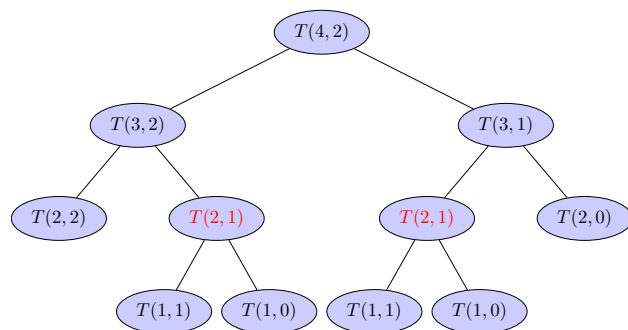


Figure 4: Calculate $\binom{4}{2}$ recursively.

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

Figure 5: Pascal triangle for binomial coefficients.

Algorithm 10 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  $B[n][k]$ 

```

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

Algorithm 11 Max-sum subarray.

```

1: procedure MSS( $A[1 \dots n]$ )
2:    $MSS[0] \leftarrow 0$ 
3:   for  $i \leftarrow 1$  to  $n$  do
4:      $MSS[i] \leftarrow \max \{MSS[i - 1] + A[i], 0\}$ 
5:   return  $\max_{1 \leq i \leq n} MSS[i]$ 

```

Algorithm 12 Max-sum subarray (Implementation Simplified).

```

1: procedure MSS( $A[1 \dots n]$ )
2:    $mss \leftarrow 0$ 
3:    $MSS \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $n$  do
5:      $MSS \leftarrow \max \{MSS + A[i], 0\}$ 
6:      $mss \leftarrow \max \{mss, MSS\}$ 
7:   return  $mss$ 

```

Traversal on Trees

DFS on Trees

Algorithm 13 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 14 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 15 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 16 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 17 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