AUBURN

UNIVERSITY

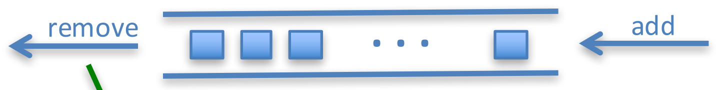
SAMUEL GINN
COLLEGE OF ENGINEERING

Binary Heaps

Motivation: Priority Queue

Conceptually similar to a stack or a queue.

A **priority queue** chooses the next element to delete based on priority.



The element returned by the remove operation will be the one with the most **extreme priority** (max or min, depending on how the priority queue is configured).

Priority is some value associated with the element that could represent importance, cost, or some other problem-specific concept.

Applications:

Interrupt handling, bandwidth management, simulation, sorting, graph algorithms, selection algorithms, compression algorithms

Priority Queue

PQ Method	Unsorted List	Sorted List	Balanced BST
add	O(1)	O(N)	
remove	O(N)	O(1)	
peek	O(N)	O(1)	

Priority Queue

PQ Method	Unsorted List	Sorted List	Balanced BST
add	O(1)	O(N)	
remove	O(N)	O(1)	
peek	O(N)	O(1)	

?

Q. What is the worst-case for each PQ if using a balanced BST?

	A	B	C	[✓ D]
add	O(N)	O(N)	O(logN)	O(logN)
remove	O(N)	O(1)	O(logN)	O(logN)
peek	O(N)	O(1)	O(1)	O(logN)

Priority Queue

PQ Method	Unsorted List	Sorted List	Balanced BST	**Binary Heap**
add	O(1)	O(N)	O(log N)	O(log N)
remove	O(N)	O(1)	O(log N)	O(log N)
peek	O(N)	O(1)	O(log N)	O(1)
	Nodes or arrays	*Nodes or arrays*	*AVL, R-B, etc.*	*Nodes or arrays*

But…

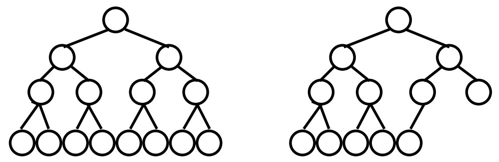
An array-based implementation is the most common and preferred.

The term "heap" usually implies an array.

Binary heaps

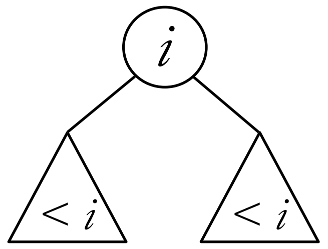
Binary heaps

A binary heap is a **complete binary tree** …

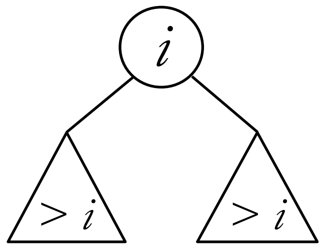


Height is O(log N)

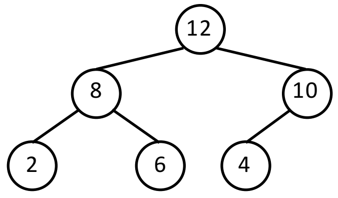
… in which each node obeys a **partial order property**.

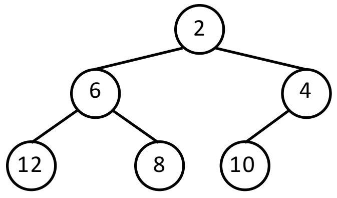


or



max heap **min heap**





Array-based implementation

Binary heaps are usually implemented as an array because:

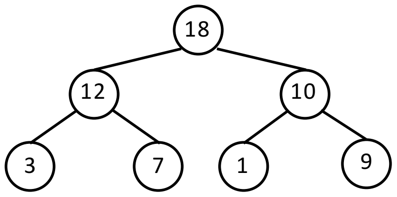
Acceptably space efficient (complete shape).

Easy traversal: parent to child via multiplication, child to parent via division

Many ways to map a hierarchy onto a linear array, but this is the one that we will use:

```
- Store the root at index 0.
- For a node stored at index i, store its left child at index
  2i+1 and its right child at index 2i+2.  Thus, the parent of
  a node stored at index i will be at index (i-1)/2.
```

Conceptually:



Implemented:

18	12	10	3	7	1	9
0	1	2	3	4	5	6

Adding values

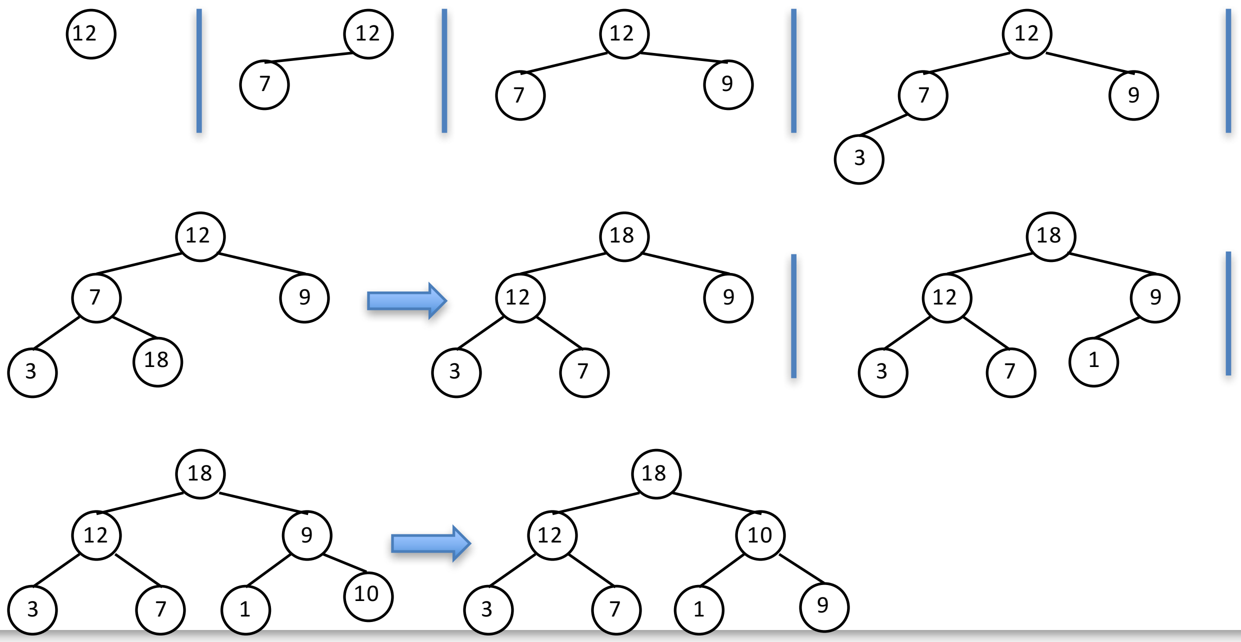
1. Insert the new element in the one and only one location that will maintain the complete shape.
2. Swap values as necessary on a leaf-to-root path to maintain the partial order.

Max heap example: 12, 7, 9, 3, 18, 1, 10

Adding values

1. Insert the new element in the one and only one location that will maintain the complete shape.
2. Swap values as necessary on a leaf-to-root path to maintain the partial order.

Max heap example: 12, 7, 9, 3, 18, 1, 10



Adding values

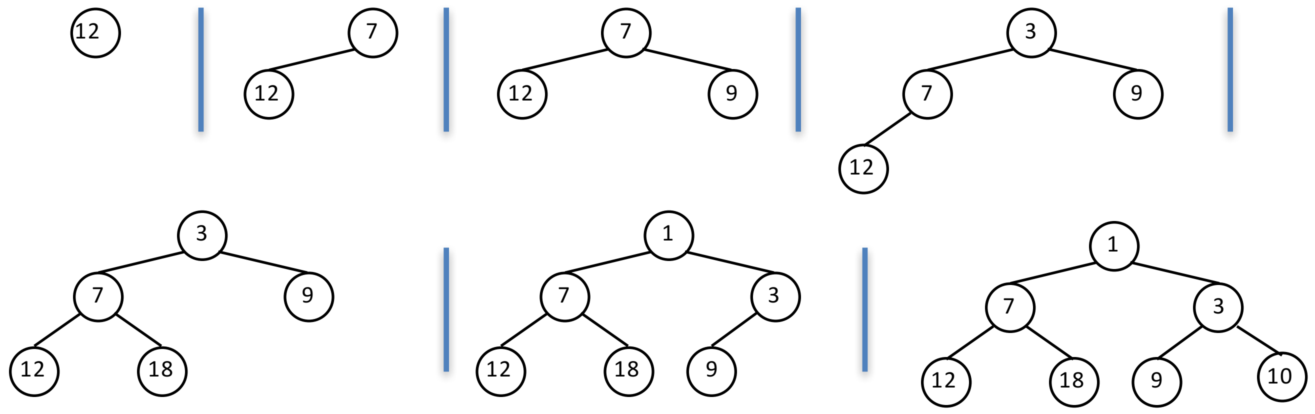
1. `Insert the new element in the one and only one location that will maintain the complete shape.`
2. `Swap values as necessary on a leaf-to-root path to maintain the partial order.`

<u>Min heap example</u>: 12, 7, 9, 3, 18, 1, 10

Adding values

1. Insert the new element in the one and only one location that will maintain the complete shape.
2. Swap values as necessary on a leaf-to-root path to maintain the partial order.

<u>Min heap example</u>: 12, 7, 9, 3, 18, 1, 10

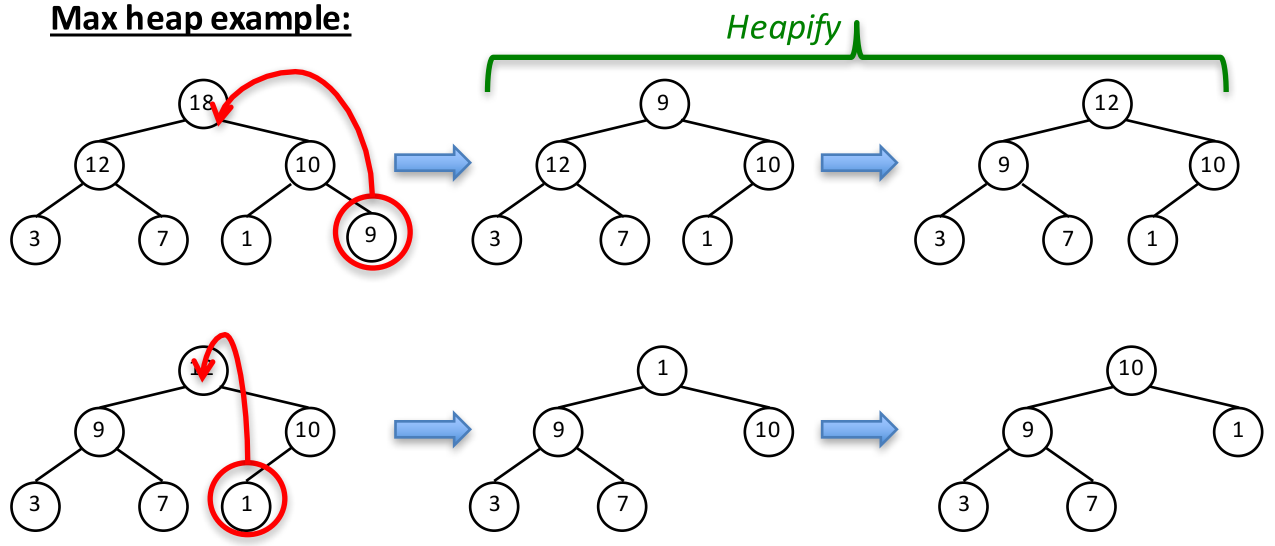


Removing values

Delete and return the element with the extreme (max/min) priority.

1. Maintain the complete shape by replacing the root value with the value in the lowest, right-most leaf. Then delete that leaf.
2. Swap values as necessary on a root-to-leaf path to maintain the partial order.

Max heap example:



Application: sorting

Heapsort

Heapsort works in two phases: (1) The initial arbitrary order of the array is transformed into a partial order, and then (2) the partial order is transformed into a total order.

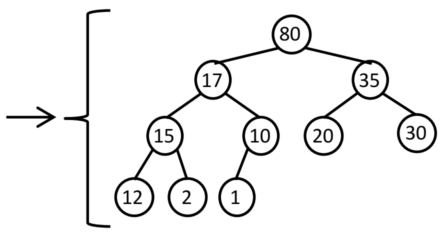
```
1. Rearrange the array elements into max heap order.
2. Repeatedly move the maximum element to its final sorted place
   toward the end of the array, and heapify the remaining elements.
```

Example:

20	12	35	15	10	80	30	17	2	1
0	1	2	3	4	5	6	7	8	9

O(N log N) *Actually, O(N)*

80	17	35	15	10	20	30	12	2	1
0	1	2	3	4	5	6	7	8	9

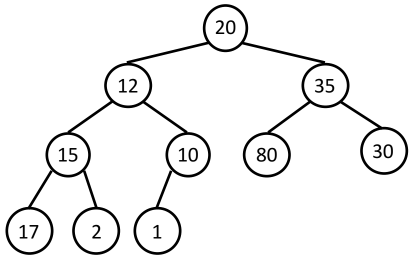


O(N log N)

1	2	10	12	15	17	20	30	35	80
0	1	2	3	4	5	6	7	8	9

Heapsort

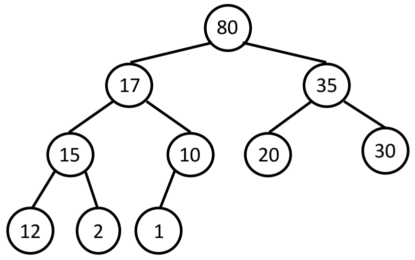
20	12	35	15	10	80	30	17	2	1
0	1	2	3	4	5	6	7	8	9



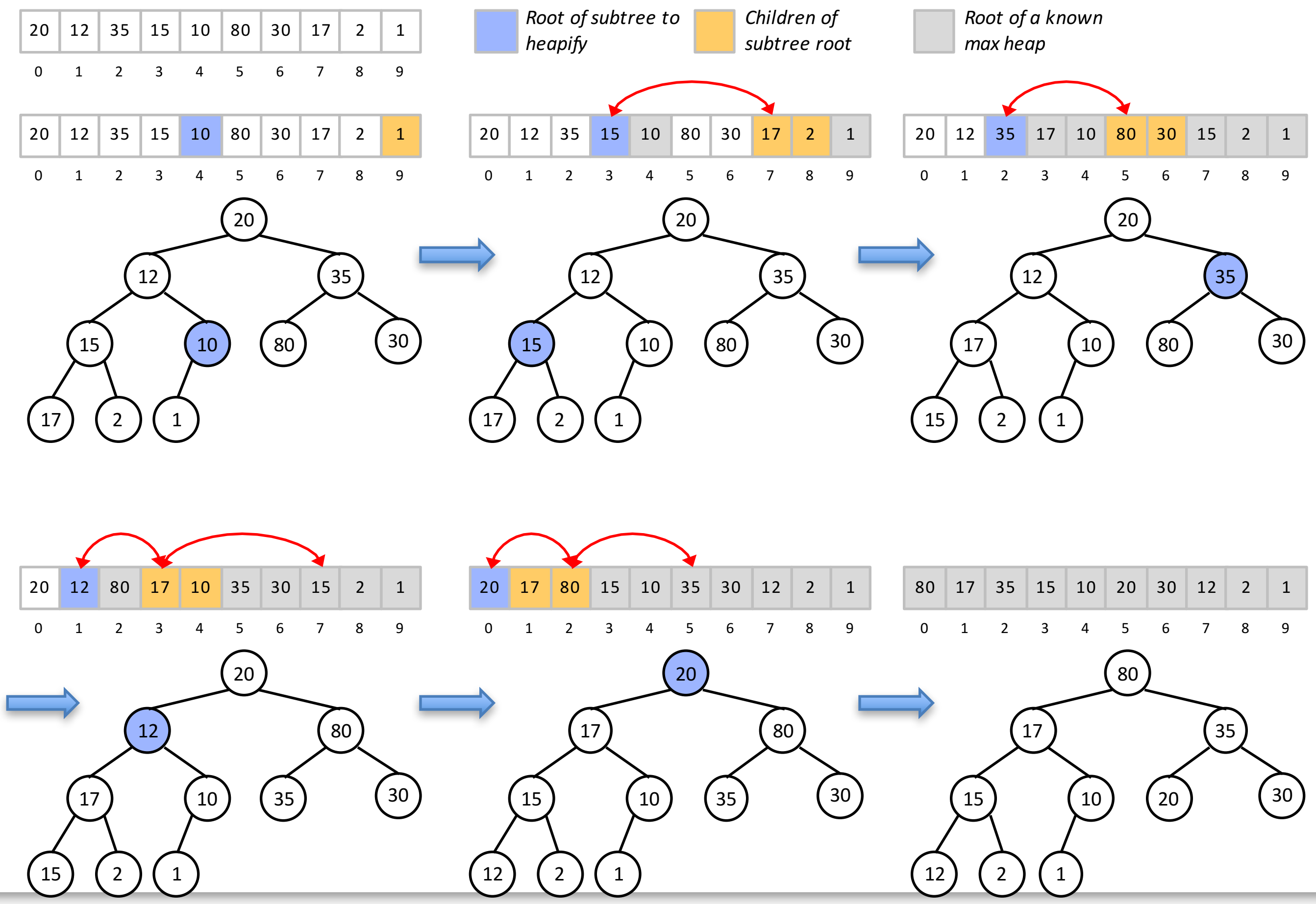
1. Rearrange the array elements into max heap order.

Beginning with the lowest, right-most parent and continuing to the root, heapify each subtree.

80	17	35	15	10	20	30	12	2	1
0	1	2	3	4	5	6	7	8	9

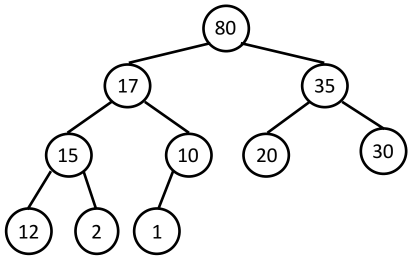


Heapsort


Root of subtree to heapify
Children of subtree root
Root of a known max heap

Heapsort

80	17	35	15	10	20	30	12	2	1
0	1	2	3	4	5	6	7	8	9



```
2.  Repeatedly move the maximum element to its final
    sorted place toward the end of the array, and
    heapify the remaining elements.

    Set last to a.length - 1
    Swap a[0] and a[last]
    last--
    Heapify a[0 .. last]
    Repeat until last == 0
```

1	2	10	12	15	17	20	30	35	80
0	1	2	3	4	5	6	7	8	9

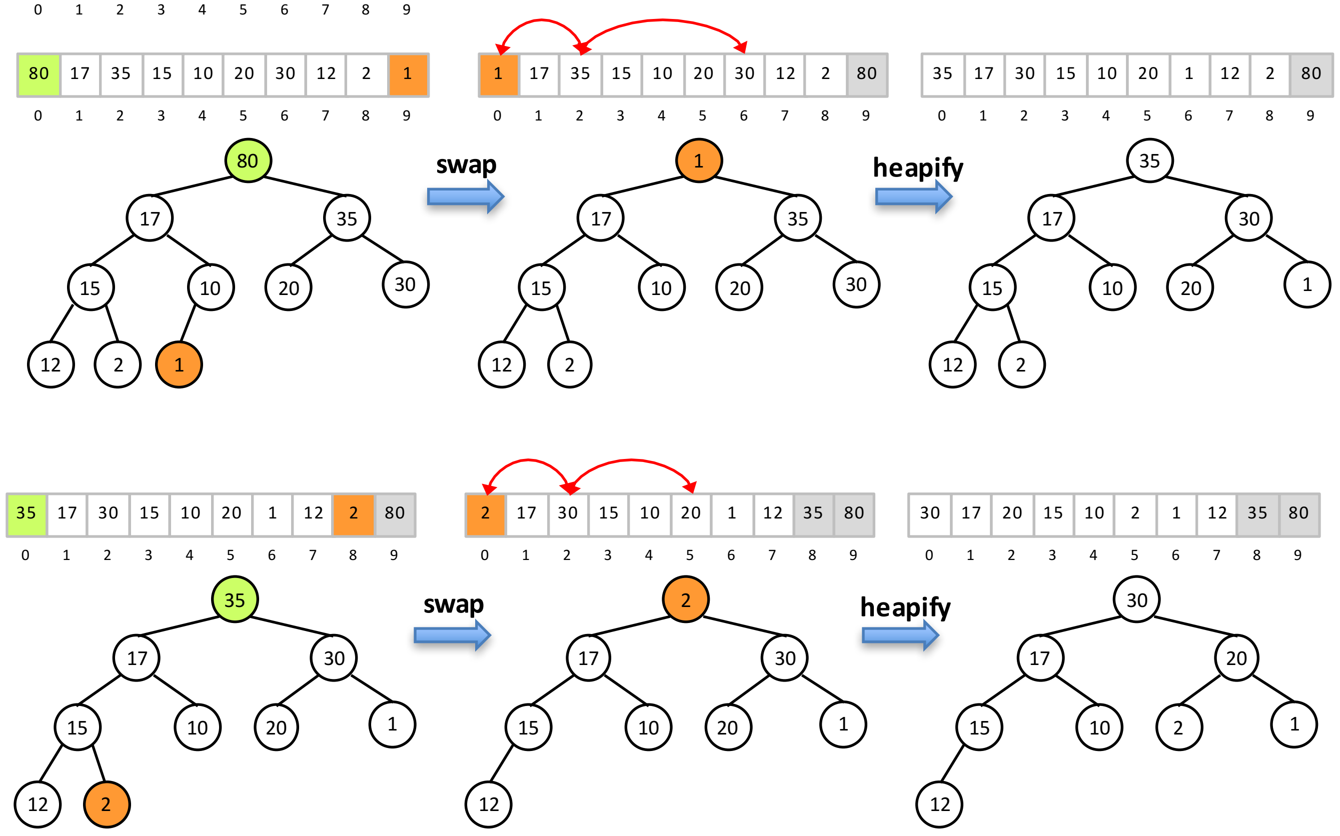
Heapsort

80	17	35	15	10	20	30	12	2	1
0	1	2	3	4	5	6	7	8	9

🟩 *Value to be swapped*

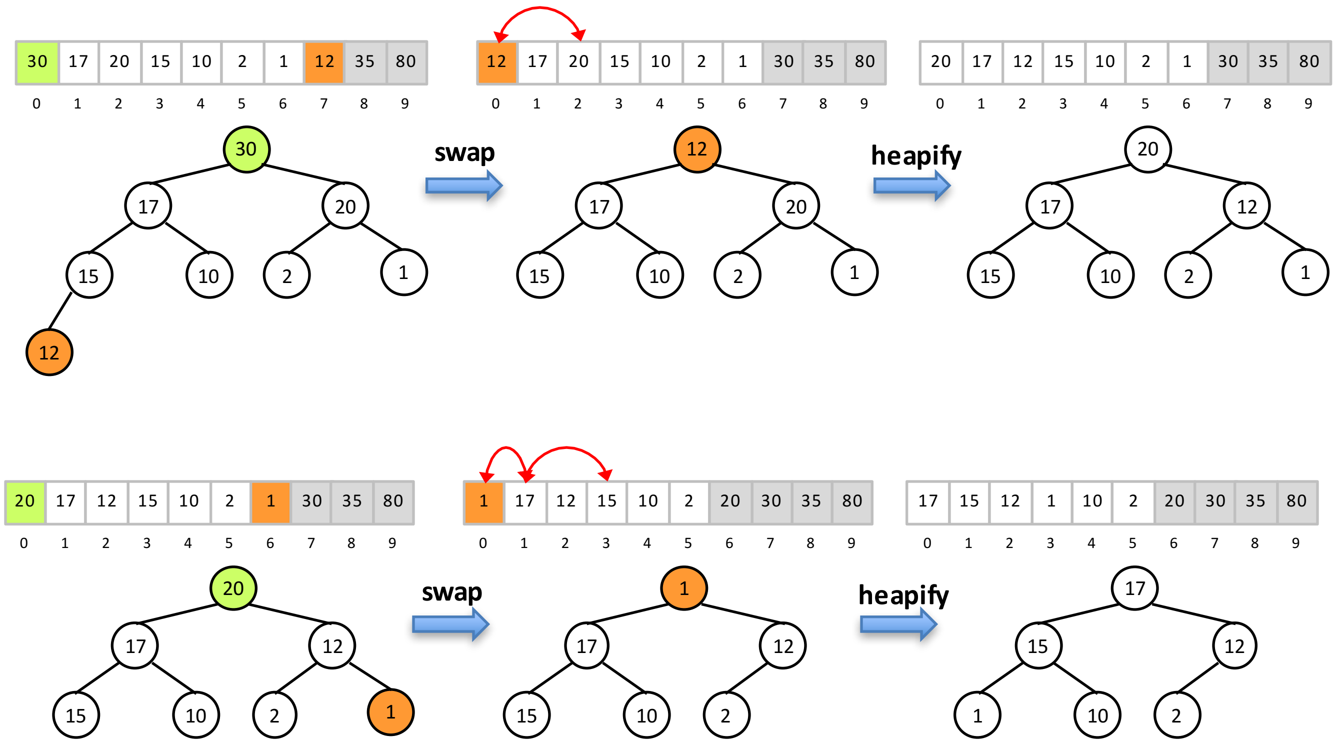
🟧 *Value to be swapped*

⬜ *In final sorted location*



Heapsort

🟩 *Value to be swapped*	🟧 *Value to be swapped*	▨ *In final sorted location*

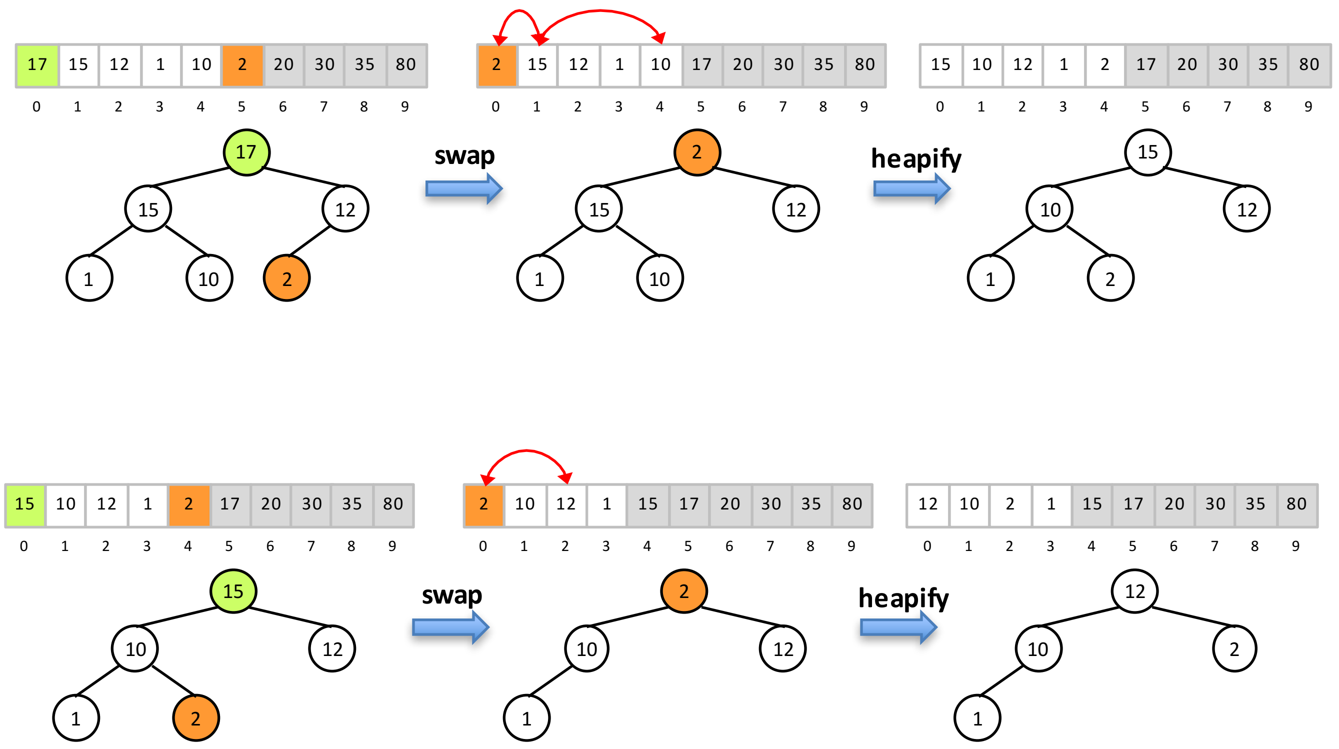


Heapsort

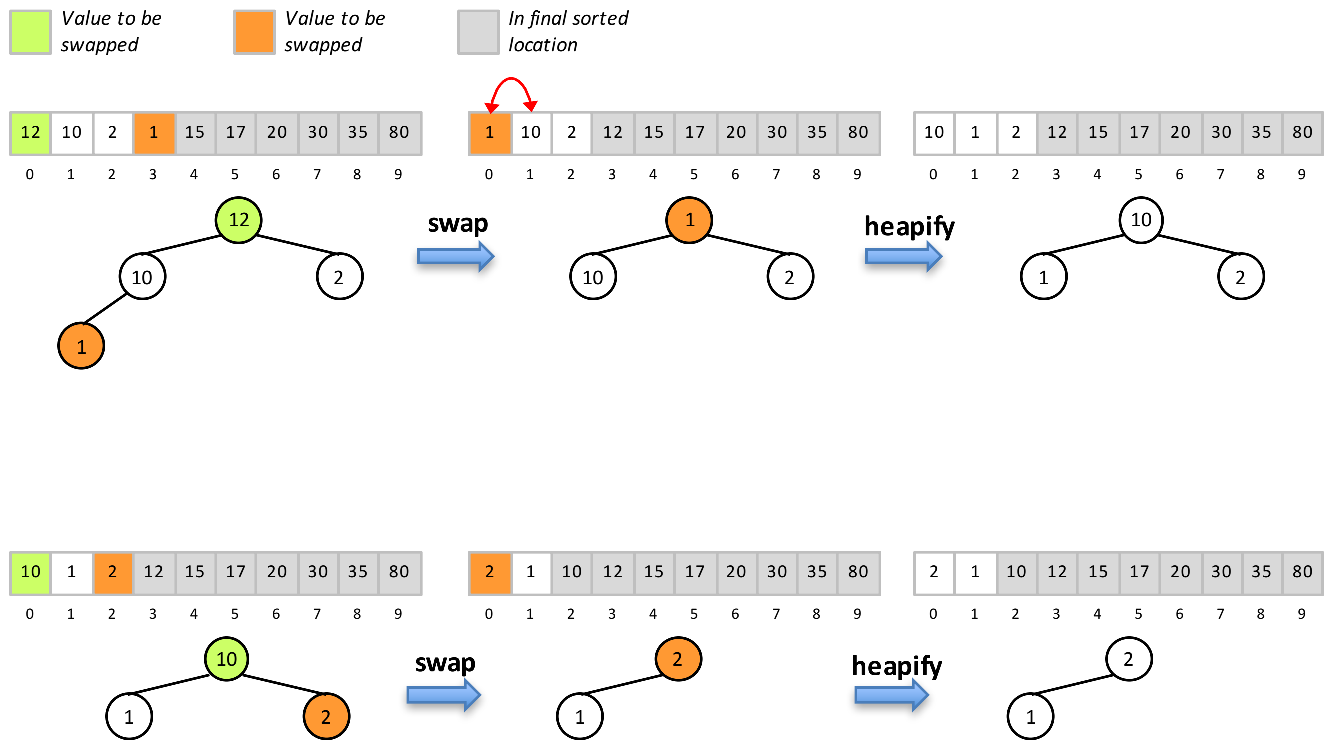
🟩 *Value to be swapped*

🟧 *Value to be swapped*

⬜ *In final sorted location*

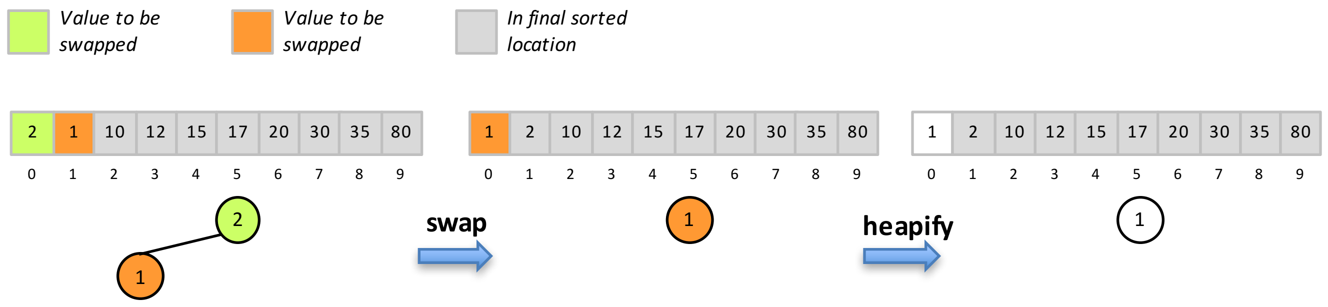


Heapsort



Value to be swapped
Value to be swapped
In final sorted location
12 10 2 1 15 17 20 30 35 80
0 1 2 3 4 5 6 7 8 9
swap
1 10 2 12 15 17 20 30 35 80
heapify
10 1 2 12 15 17 20 30 35 80
10 1 2 12 15 17 20 30 35 80
swap
2 1 10 12 15 17 20 30 35 80
heapify
2 1 10 12 15 17 20 30 35 80

Heapsort



Heapsort is an in-place sort with guaranteed NlogN worst-case performance.

Mergesort? No. (NlogN worst-case, but needs N extra space.)

Quicksort? No. (In-place, but N^2 in worst-case.)

But, heapsort is not stable and it typically has larger constant factors than quicksort.

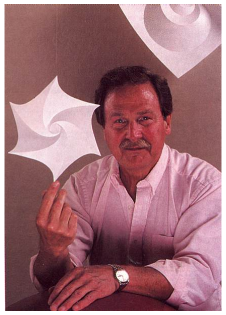
Application: Huffman’s algorithm

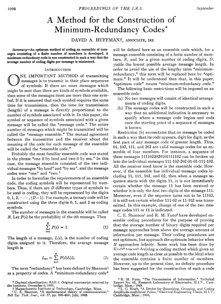
Huffman's algorithm

Huffman's algorithm generates a variable-length encoding for a given alphabet for the purposes of data compression.

Developed by David Huffman in 1951 as a class project at MIT, and published in 1952.

Widely used today as part of various compression utilities (PKZIP, MP3, JPEG).





1098 *PROCEEDINGS OF THE I.R.E.* *September*

A Method for the Construction of Minimum-Redundancy Codes*

DAVID A. HUFFMAN†, ASSOCIATE, IRE

Summary—An optimum method of coding an ensemble of messages consisting of a finite number of members is developed. A minimum-redundancy code is one constructed in such a way that the average number of coding digits per message is minimized.

INTRODUCTION

ONE IMPORTANT METHOD of transmitting messages is to transmit in their place sequences of symbols. If there are more messages which might be sent than there are kinds of symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it. In this paper, the symbol or sequence of symbols associated with a given message will be called the "message code." The entire number of messages which might be transmitted will be called the "message ensemble." The mutual agreement between the transmitter and the receiver about the meaning of the code for each message of the ensemble will be called the "ensemble code."

Probably the most familiar ensemble code was stated in the phrase "one if by land and two if by sea." In this case, the message ensemble consisted of the two individual messages "by land" and "by sea", and the message codes were "one" and "two."

In order to formalize the requirements of an ensemble code, the coding symbols will be represented by numbers. Thus, if there are D different types of symbols to be used in coding, they will be represented by the digits $0, 1, 2, \cdots, (D-1)$. For example, a ternary code will be constructed using the three digits 0, 1, and 2 as coding symbols.

The number of messages in the ensemble will be called N. Let $P(i)$ be the probability of the ith message. Then

$$\sum_{i=1}^{N} P(i) = 1. \quad (1)$$

The length of a message, $L(i)$, is the number of coding digits assigned to it. Therefore, the average message length is

$$L_{av} = \sum_{i=1}^{N} P(i)L(i). \quad (2)$$

The term "redundancy" has been defined by Shannon[1] as a property of codes. A "minimum-redundancy code" will be defined here as an ensemble code which, for a message ensemble consisting of a finite number of members, N, and for a given number of coding digits, D, yields the lowest possible average message length. In order to avoid the use of the lengthy term "minimum-redundancy," this term will be replaced here by "optimum." It will be understood then that, in this paper, "optimum code" means "minimum-redundancy code."

The following basic restrictions will be imposed on an ensemble code:

(a) No two messages will consist of identical arrangements of coding digits.

(b) The message codes will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.

Restriction (b) necessitates that no message be coded in such a way that its code appears, digit for digit, as the first part of any message code of greater length. Thus, 01, 102, 111, and 202 are valid message codes for an ensemble of four members. For instance, a sequence of these messages 1111022020101111102 can be broken up into the individual messages 111-102-202-01-01-111-102. All the receiver need know is the ensemble code. However, if the ensemble has individual message codes including 11, 111, 102, and 02, then when a message sequence starts with the digits 11, it is not immediately certain whether the message 11 has been received or whether it is only the first two digits of the message 111. Moreover, even if the sequence turns out to be 11102, it is still not certain whether 111-02 or 11-102 was transmitted. In this example, change of one of the two message codes 111 or 11 is indicated.

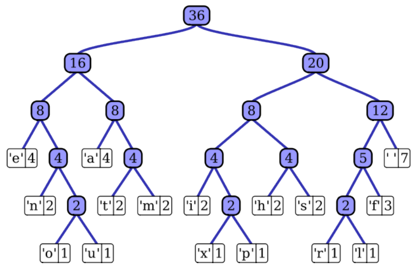
C. E. Shannon[1] and R. M. Fano[2] have developed ensemble coding procedures for the purpose of proving that the average number of binary digits required per message approaches from above the average amount of information per message. Their coding procedures are not optimum, but approach the optimum behavior when N approaches infinity. Some work has been done by Kraft[3] toward deriving a coding method which gives an average code length as close as possible to the ideal when the ensemble contains a finite number of members. However, up to the present time, no definite procedure has been suggested for the construction of such a code

* Decimal classification: R531.1. Original manuscript received by the Institute, December 6, 1951.
† Massachusetts Institute of Technology, Cambridge, Mass.
[1] C. E. Shannon, "A mathematical theory of communication," *Bell Sys. Tech. Jour.*, vol. 27, pp. 398–403; July, 1948.
[2] R. M. Fano, "The Transmission of Information," Technical Report No. 65, Research Laboratory of Electronics, M.I.T., Cambridge, Mass.; 1949.
[3] L. G. Kraft, "A Device for Quantizing, Grouping, and Coding Amplitude-modulated Pulses," Electrical Engineering Thesis, M.I.T., Cambridge, Mass.; 1949.

Famous story:

In 1951, David A. Huffman and his MIT information theory classmates were given the choice of a term paper or a final exam. The professor, Robert M. Fano, assigned a term paper on the problem of finding the most efficient binary code. Huffman, unable to prove any codes were the most efficient, was about to give up and start studying for the final when he hit upon the idea of using a frequency-sorted binary tree and quickly proved this method the most efficient.

In doing so, the student outdid his professor, who had worked with information theory inventor Claude Shannon to develop a similar code. Huffman avoided the major flaw of the suboptimal Shannon-Fano coding by building the tree from the bottom up instead of from the top down.



Huffman's algorithm

ASCII American Standard Code for Information Interchange

Binary character encoding scheme: A sequence of 0s and 1s (bits) used to encode characters.

ASCII includes English alphabet, punctuation, digits, and "control" characters (e.g., newline, carriage return).

Binary	Decimal	Char
100 0000	64	@
100 0001	65	A
100 0010	66	B
100 0011	67	C
100 0100	68	D

The 95 printable characters in ASCII:

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

ASCII is a fixed length code. Each character is represented by the same number of bits.

In US-ASCII, each character is represented in one byte (8 bits).

```
% more  abcfile.txt
ABC
% ls  -l abcfile.txt
-rw-r--r--    1 User  User   4 Apr   2 09:18  abcfile.txt
%
```

8 bits = 1 parity bit and 7 bits to encode the character. 2^7 = 128 different characters

Huffman's algorithm

Text file:

BABACEDABCDABABACD
ADABCCABCA

28 characters

Binary ASCII form:

```
01000010010000010100
00100100000101000011
01000101010001000100
00010100001001000011
01000100010000010100
0010…
```

28 bytes

28 * 7 = 196 bits

Text compression stores the same information in fewer bytes.

We could compress this file by taking advantage of the fact that some characters appear more often than others.

Huffman's algorithm

Number of bits per character determined by the char's relative frequency of occurrence.

Most frequently occurring characters should use the fewest bits.

Text file:

BABACEDABCDABABACD
ADABCCABCA

Character frequency:

A-10, B-7, C-6, D-4, E-1

A variable length code:

A = 11
B = 10
C = 00
D = 011
E = 010

"Compressed" file:

10111011000100111110
0001111011101100011
1101111000001110001
1

Only 61 bits

Uncompressed file required 196 bits

This would compress the file to 31% of its original size.

Huffman's algorithm

A first attempt: Iterate over the alphabet in descending order of frequency. Assign the next smallest unique bit string to the current character, starting with '0'.

Character frequency:

A-10, B-7, C-6, D-4, E-1

The variable length code:

A = 0
B = 1
C = 01
D = 10
E = 11

The vlc must have the **prefix property**.

Text file:

BABACEDABCDABABACD
ADABCCABCA

$\downarrow$ *zip*

"Compressed" file:

1010011110010110…

$\downarrow$ *unzip* *Can't reconstruct the original!*

Does the file start with a B or a D??

The code for one character can't be a prefix of another character's code.

Huffman's algorithm

Binary trees in which the leaves contain the characters to be coded.

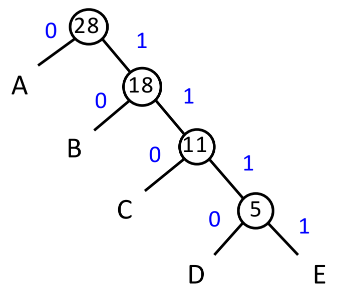
Interior nodes are just place-holders.

The root of every subtree is annotated with the cumulative frequency of all its descendent leaves.

Character codes are generated by root to leaf traversals.

Left branch = 0, Right branch = 1

A-10
B-7
C-6
D-4
E-1

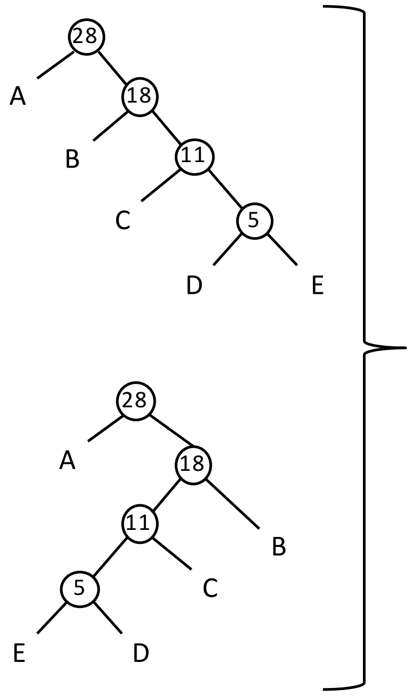


A = 0
B = 10
C = 110
D = 1110
E = 1111

No code is the prefix to another.

Huffman's algorithm

A-10, B-7, C-6, D-4, E-1

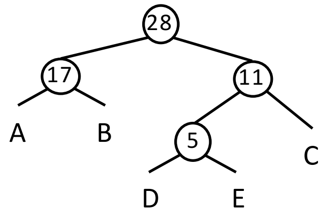


We would like to use the code tree with minimum **expected code length**.

$$L(C) = \sum_{i=1}^{N} w_i \times length(c_i)$$

This is just a weighted average of all possible character code lengths.

```
A: (10 ÷ 28) * 1 = 0.36
B: ( 7 ÷ 28) * 2 = 0.50
C: ( 6 ÷ 28) * 3 = 0.66
D: ( 4 ÷ 28) * 4 = 0.56
E: ( 1 ÷ 28) * 4 = 0.12
                   2.20
```



2.18

Huffman's algorithm generates a code tree with an expected code length that is at least as small as any other code tree that could be generated.

• • •

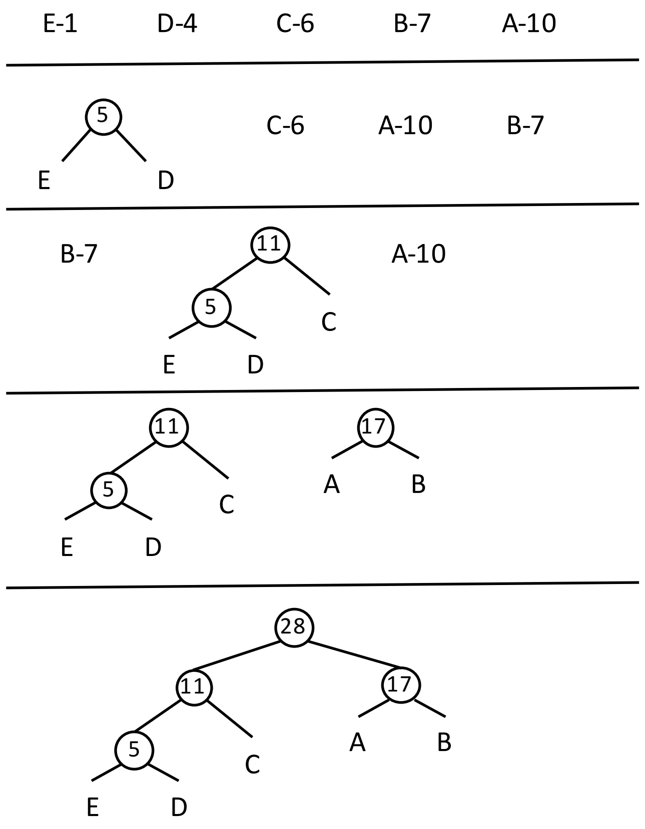
Huffman's algorithm

Generates a variable length code with the prefix property such that there is no other encoding with a smaller expected code length.

A-10, B-7, C-6, D-4, E-1

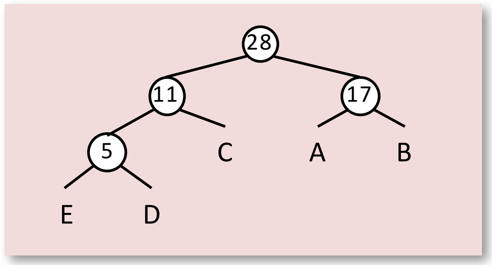
```
Create a single node code tree for
each character and insert each of
these trees into a priority queue (min
heap).

while (pq has more than one element) {
   c1 = pq.deletemin();
   c2 = pq.deletemin();
   c3 = new codetree(c1,c2);
   pq.add(c3);
}
```

Char	Encoding
A	10
B	11
C	01
D	001
E	000



Huffman's algorithm

A-10, B-7, C-6, D-4, E-1

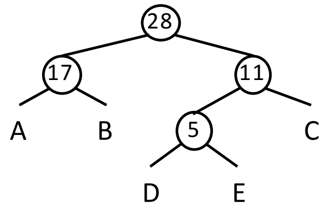


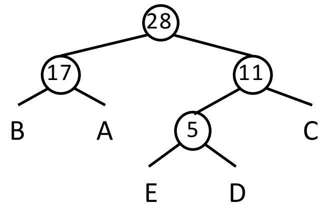
Char	Encoding
A	10
B	11
C	01
D	001
E	000

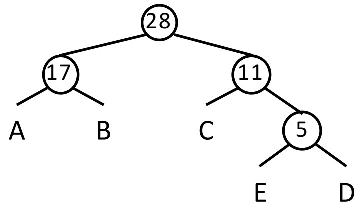
Expected code length:

```
A: (10 ÷ 28) * 2 = 0.71
B: ( 7 ÷ 28) * 2 = 0.50
C: ( 6 ÷ 28) * 2 = 0.43
D: ( 4 ÷ 28) * 3 = 0.43
E: ( 1 ÷ 28) * 3 = 0.11
                   2.18
```

This is not the only code tree with minimum expected code length.







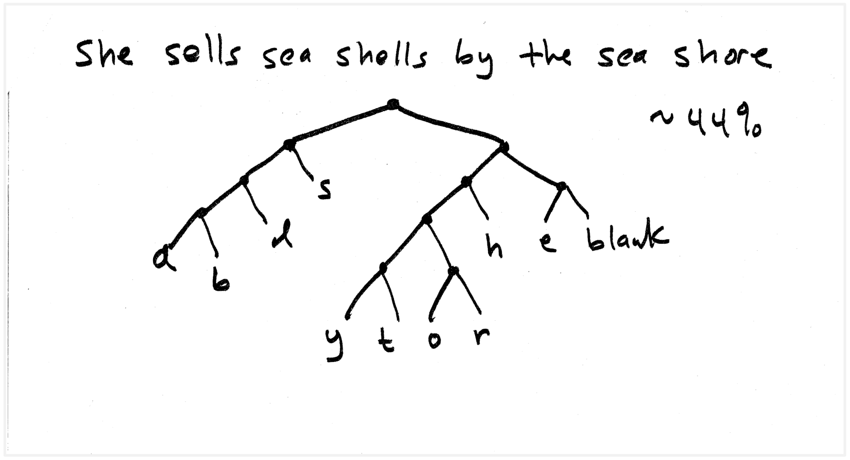
• • •

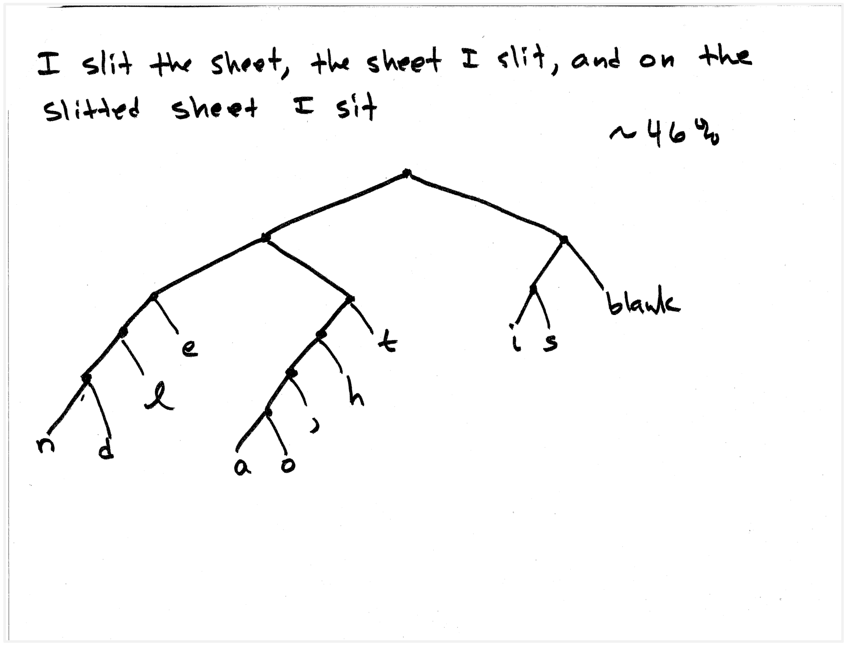
Char	Encoding
A	00
B	01
C	11
D	100
E	101

Char	Encoding
A	01
B	00
C	11
D	101
E	100

Char	Encoding
A	00
B	01
C	10
D	110
E	111

Huffman's algorithm




I slit the sheet, the sheet I slit, and on the slitted sheet I sit
~46%
blank
e
t
i
s
l
h
n
d
,
a
o