Binary Heaps

COMP 2210 – Dr. Hendrix

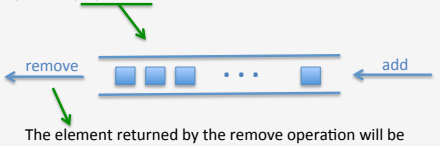
AUBURN UNIVERSITY

SAMUEL GINN COLLEGE OF ENGINEERING

Motivation: Priority Queue collection

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

More info:

http://en.wikipedia.org/wiki/Priority_queue

http://download.oracle.com/javase/6/docs/api/java/util/PriorityQueue.html

Implementing a 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)	
	Nodes or arrays	*Nodes or arrays*	*AVL, R-B, etc.*

Participation question

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)

Implementing a priority queue

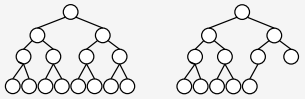
PQ Method	Unsorted List	Sorted List	Balanced BST	**Binary Heap**
add	O(1)	O(N)	O(log N)	O(log N)
remove	O(N)	O(1)	O(log N)	O(log N)
peek	O(N)	O(1)	O(log N)	O(1)
	Nodes or arrays	*Nodes or arrays*	*AVL, R-B, etc.*	*Nodes or arrays* But…

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

The term "heap" usually implies an array.

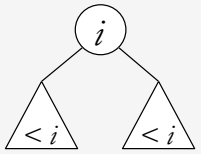
Binary Heap

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

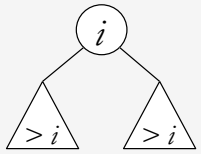


Height is O(log N)

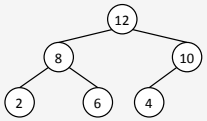
… in which each node obeys a **partial order property**.



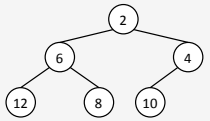
or



max heap



min heap



Array-Based Implementation

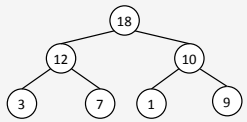
Binary heaps are almost always implemented as an array because:

- Acceptably space efficient (complete shape).
- Easy traversal: parent to child via multiplication, child to parent via division

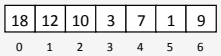
Many ways to map a hierarchy onto a linear array, but this is the one that we will use:

```
- 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:



Participation question

Q. Which of the following arrays stores its element in **max-heap** order?

A

2	4	6	8	10
0	1	2	3	4

B

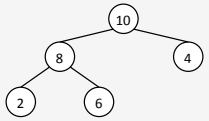
10	6	4	8	2
0	1	2	3	4

C

10	8	4	2	6
0	1	2	3	4

D

10	4	8	2	6
0	1	2	3	4



Binary Heap Insertion

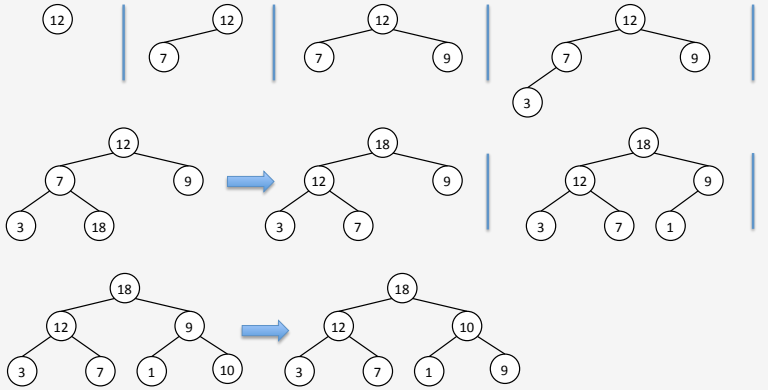
1. Insert the new element in the one and only one location that will maintain the complete shape.
2. Swap values as necessary on a leaf-to-root path to maintain the partial order.

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

Binary Heap Insertion

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



Binary Heap Insertion

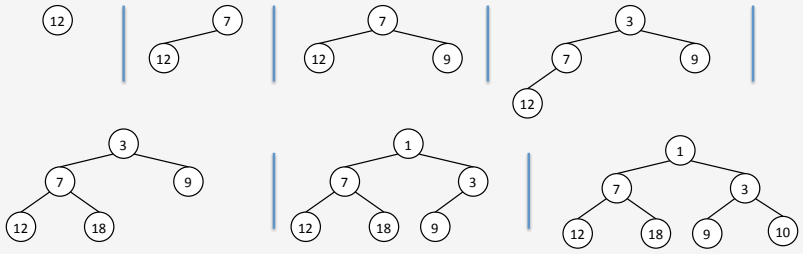
1. Insert the new element in the one and only one location that will maintain the complete shape.
2. Swap values as necessary on a leaf-to-root path to maintain the partial order.

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

Binary Heap Insertion

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.

Min heap example: 12, 7, 9, 3, 18, 1, 10 *(Without showing separate swap step)*

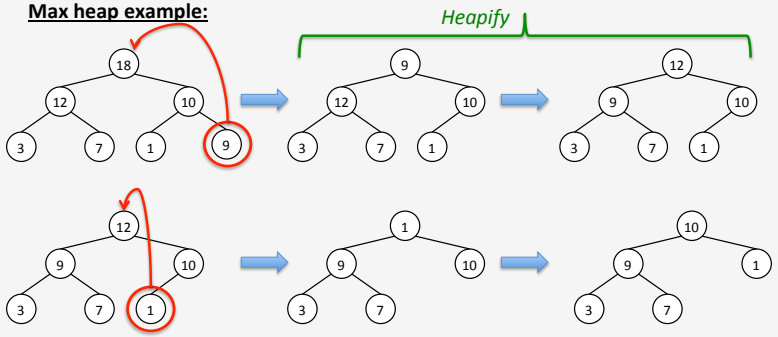


Binary Heap Deletion

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

This is an application of the idea, rather than the data structure/collection per se.

Heapsort is an in-place comparison sort with O(N log N) time complexity.

Why is this important?

Sorting complexity

For **comparison sorts**, $N \log N$ is a lower bound on the number of comparisons necessary.
(fun for 3270)

So, in that sense, both merge sort and quicksort are optimal.

Quicksort is expected to be faster on typical data sets.

Once again, does this really matter??

Let's say your home computer can execute 10^8 compares/second and your neighborhood supercomputer can execute 10^{12} compares/second.

	Insertion sort – O(N²)			Mergesort – O(N log N)			Quicksort – O(N log N)		
computer	thousand	million	billion	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min	instant	0.3 sec	6 min
super	instant	1 second	1 week	instant	instant	instant	instant	instant	instant

Take away: Good algorithms can be better than supercomputers.

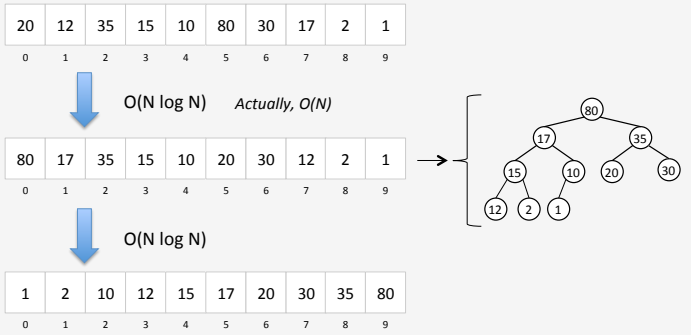
Great algorithms are better than good ones.

Heapsort

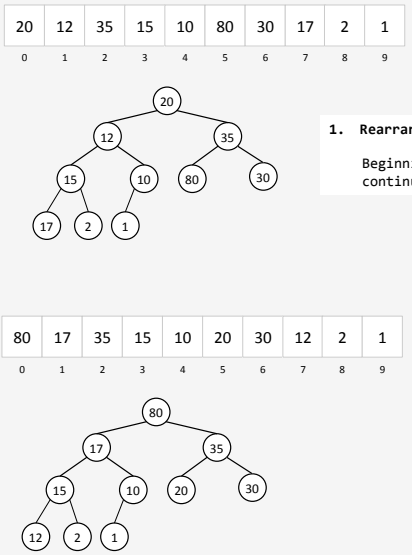
Heapsort works in two phases: (1) The initial arbitrary order of the array is transformed into a partial order, and then (2) the partial order is transformed into a total order.

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:



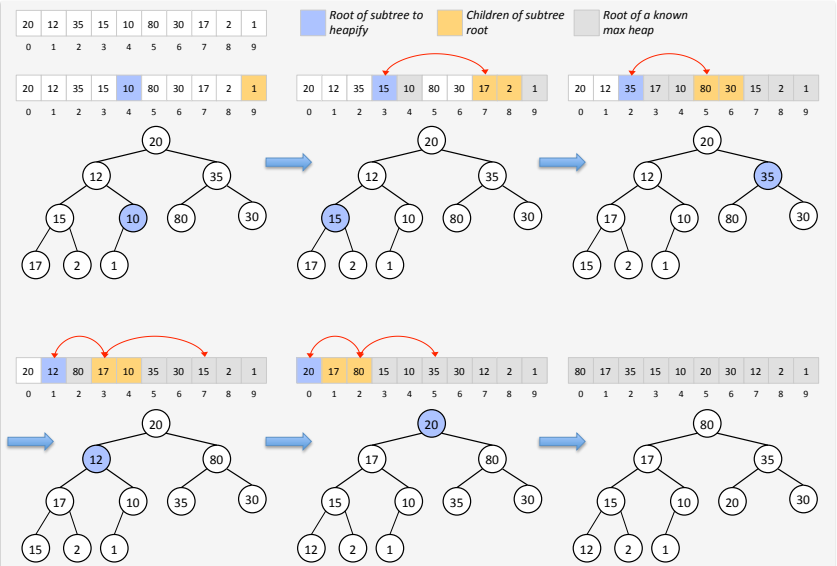
Heapsort – rearrange into max heap order



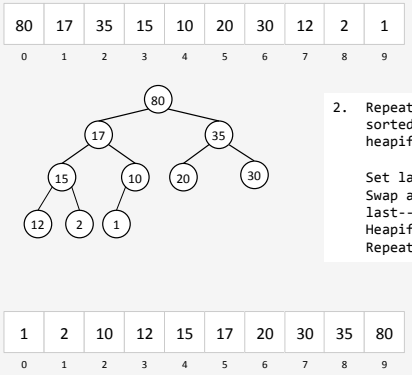
1. **Rearrange the array elements into max heap order.**

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

Heapsort – rearrange into max heap order



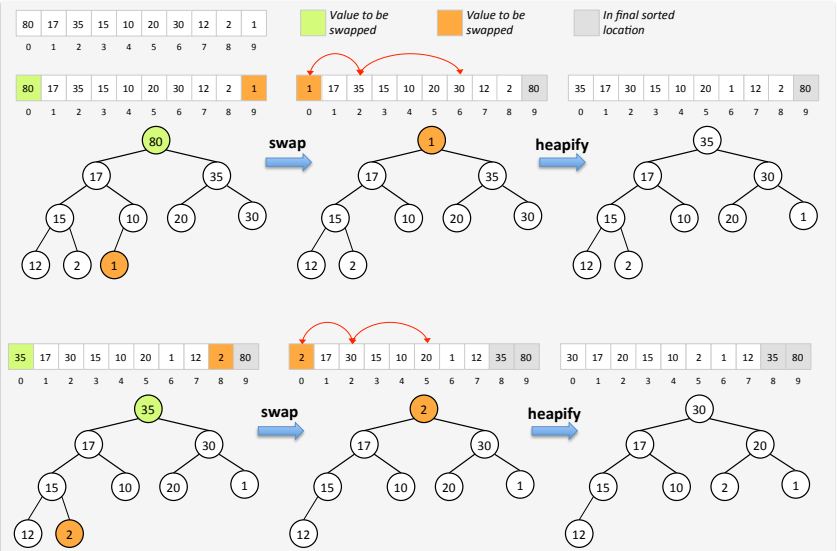
Heapsort – transform into total order



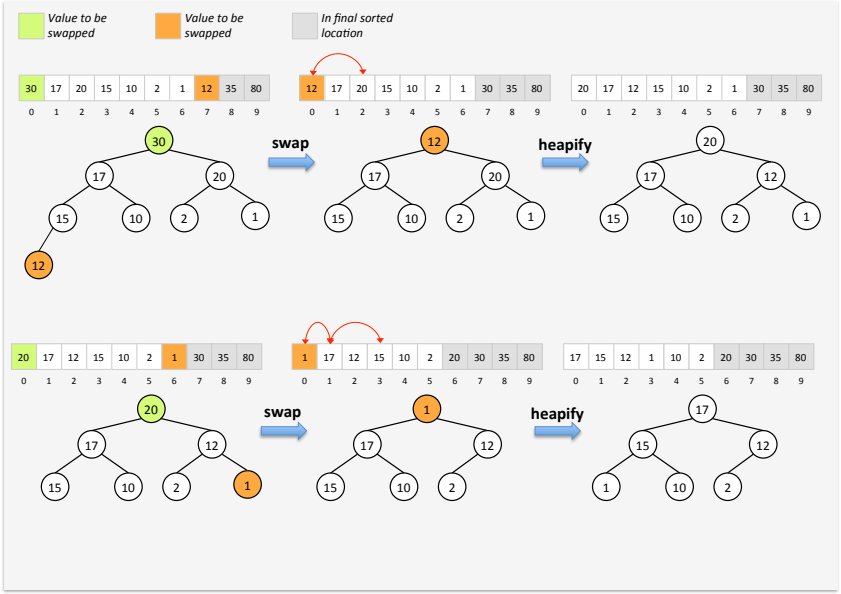
2. Repeatedly move the maximum element to its final sorted place toward the end of the array, and heapify the remaining elements.

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

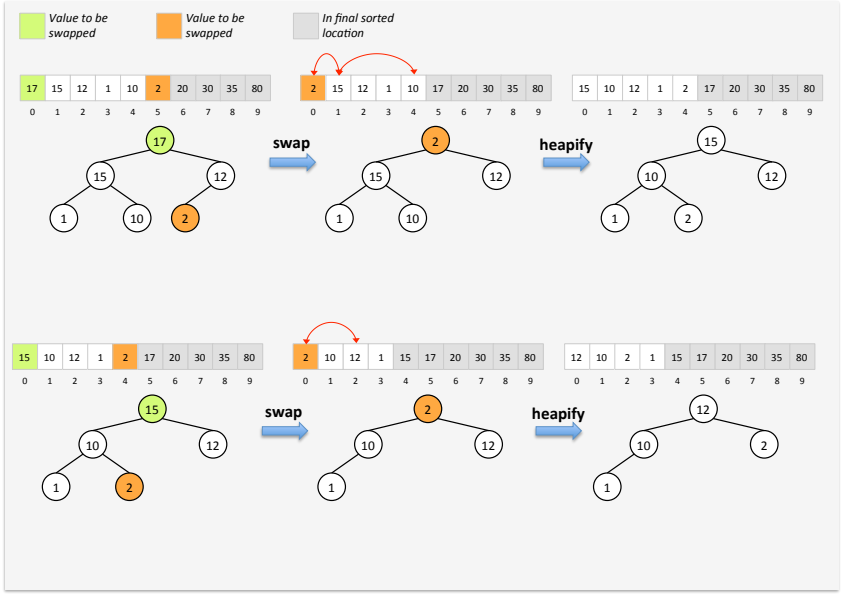
Heapsort – transform into total order



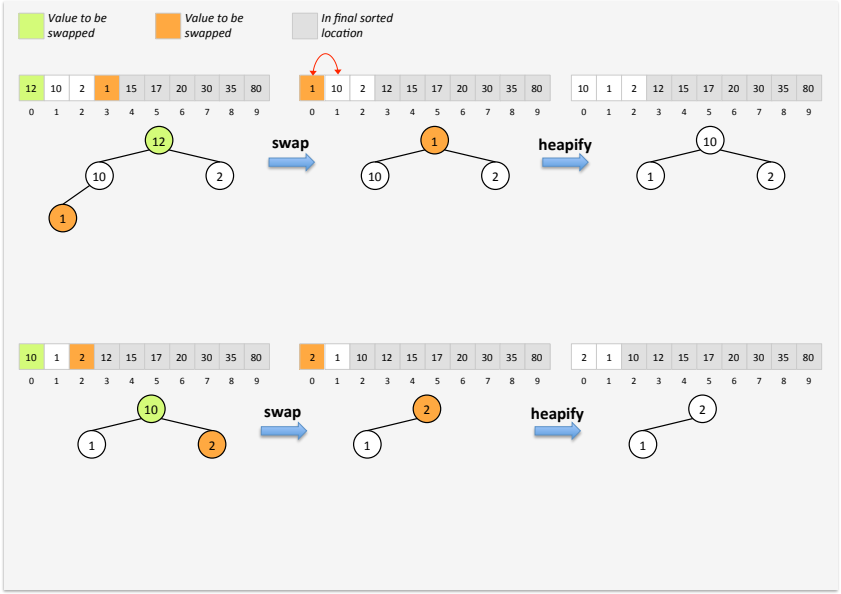
Heapsort – transform into total order



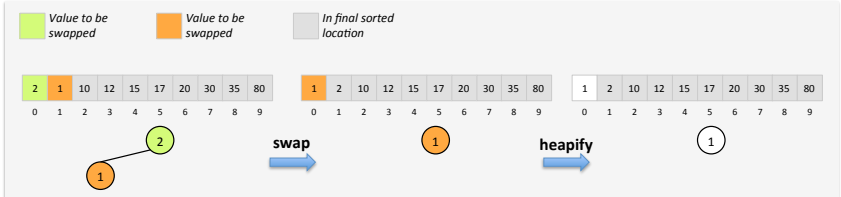
Heapsort – transform into total order



Heapsort – transform into total order



Heapsort – transform into total order



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

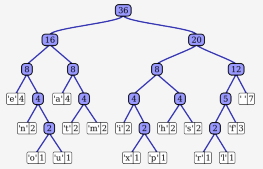
Huffman's algorithm generates a variable-length encoding for a given alphabet for the purposes of data compression.

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).

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.*



Is that all there is to it???
Robert Fano

Character encoding: :-)

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

ASCII example

Text file:

BABACEDABCDABABACD
ADABCCABCA

28 characters

Text compression stores the same information in fewer bytes.

Binary ASCII form:

01000010010000010100
00100100000101000011
01000101010001000100
00010100001001000011
01000100010000010100
0010…

28 bytes

28 * 7 = 196 bits

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

Variable-length codes

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:

10111011000100111110000111111010111011101100011111
0111110000011100011

Only 61 bits

Uncompressed file required 196 bits

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

Generating a vlc

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

↓ *zip*

"Compressed" file:

1010011110010110…

↓ *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.

Code trees

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

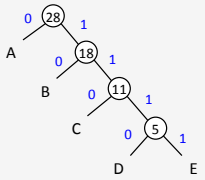
Interior nodes are just place-holders.

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

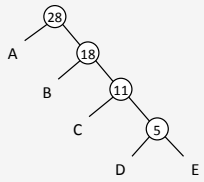


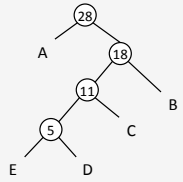
A = 0
B = 10
C = 110
D = 1110
E = 1111

No code is the prefix to another.

Many possible code trees

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





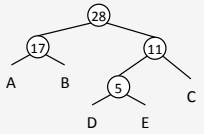
• • •

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.

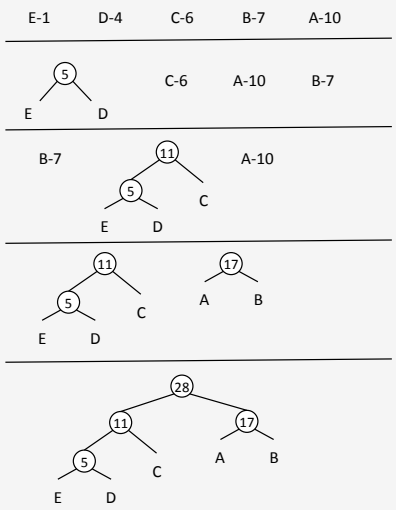
Huffman's algorithm

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

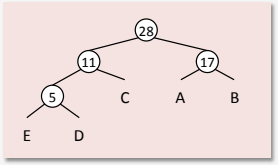
```
Create a single node code tree for each
character and insert each of these
trees into a priority queue (min heap).

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*

Generated by the algorithm:

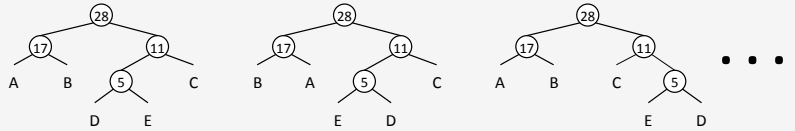


Char	Encoding
A	10
B	11
C	01
D	001
E	000

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.



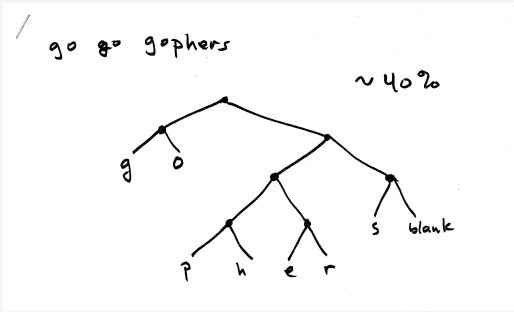
Char	Encoding
A	00
B	01
C	11
D	100
E	101

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

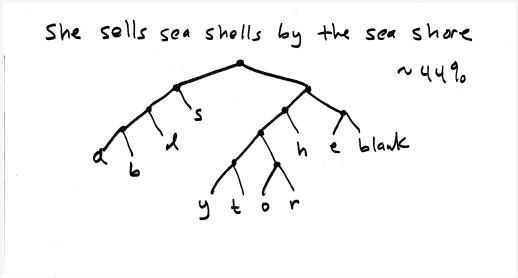
Char	Encoding
A	00
B	01
C	10
D	110
E	111

Huffman's algorithm *A-6, B-2, C-3, D-3, E-4, F-9*

Huffman's algorithm



Huffman's algorithm



Huffman's algorithm