2.

A)

aa=(return value from fact(6))
main

aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=4*(return value from fact(3))
n=4
aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=3*(return value from fact(2))
n=3
aa=4*(return value from fact(3))
n=4
aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=2*(return value from fact(1))
n=2
aa=3*(return value from fact(2))
n=3
aa=4*(return value from fact(3))
n=4

aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

result=1
n=1
aa=2*(return value from fact(1))
n=2
aa=3*(return value from fact(2))
n=3
aa=4*(return value from fact(3))
n=4
aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=2*1=2
n=2
aa=3*(return value from fact(2))
n=3
aa=4*(return value from fact(3))
n=4
aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=3*2=6
n=3
aa=4*(return value from fact(3))
n=4
aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6

aa=(return value from fact(6))
main

aa=4*6=24
n=4
aa=5*(return value from fact(4))
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=5*24=120
n=5
aa=6*(return value from fact(5))
n=6
aa=(return value from fact(6))
main

aa=6*120=720
n=6
aa=(return value from fact(6))
main

aa=720
main

Time complexity: O(N).

B)

I would use the tree structure. Moving N disks from rod A to rod B equals moving (N-1) disks from rod A to rod C and move the N disk from A to B and move (N-1) disks from rod C to rod B. Thus, the time taken T(N) equals 2T(N-1)+1. It is easy to know that moving one disk from a rod to another takes 1 step which means T(1)=1. So a recursive method could be used in Towers of Hanoi problem. The steps are shown in the Java code.

Move disk-1 from rod A to rod C
Move disk-2 from rod A to rod B
Move disk-1 from rod C to rod B
Move disk-3 from rod A to rod C
Move disk-1 from rod B to rod A
Move disk-2 from rod B to rod C
Move disk-1 from rod A to rod C
Move disk-4 from rod A to rod B

Move disk-1 from rod C to rod B
Move disk-2 from rod C to rod A
Move disk-1 from rod B to rod A
Move disk-3 from rod C to rod B
Move disk-1 from rod A to rod C
Move disk-2 from rod A to rod B
Move disk-1 from rod C to rod B
Move disk-5 from rod A to rod C
Move disk-1 from rod B to rod A
Move disk-2 from rod B to rod C
Move disk-1 from rod A to rod C
Move disk-3 from rod B to rod A
Move disk-1 from rod C to rod B
Move disk-2 from rod C to rod A
Move disk-1 from rod B to rod A
Move disk-4 from rod B to rod C
Move disk-1 from rod A to rod C
Move disk-2 from rod A to rod B
Move disk-1 from rod C to rod B
Move disk-3 from rod A to rod C
Move disk-1 from rod B to rod A
Move disk-2 from rod B to rod C
Move disk-1 from rod A to rod C
Move disk-6 from rod A to rod B
Move disk-1 from rod C to rod B
Move disk-2 from rod C to rod A
Move disk-1 from rod B to rod A
Move disk-3 from rod C to rod B
Move disk-1 from rod A to rod C
Move disk-2 from rod A to rod B
Move disk-1 from rod C to rod B
Move disk-4 from rod C to rod A
Move disk-1 from rod B to rod A
Move disk-2 from rod B to rod C
Move disk-1 from rod A to rod C
Move disk-3 from rod B to rod A
Move disk-1 from rod C to rod B
Move disk-2 from rod C to rod A
Move disk-1 from rod B to rod A
Move disk-5 from rod C to rod B
Move disk-1 from rod A to rod C
Move disk-2 from rod A to rod B
Move disk-1 from rod C to rod B
Move disk-3 from rod A to rod C

Move disk-1 from rod B to rod A
Move disk-2 from rod B to rod C
Move disk-1 from rod A to rod C
Move disk-4 from rod A to rod B
Move disk-1 from rod C to rod B
Move disk-2 from rod C to rod A
Move disk-1 from rod B to rod A
Move disk-3 from rod C to rod B
Move disk-1 from rod A to rod C
Move disk-2 from rod A to rod B
Move disk-1 from rod C to rod B

C)

Iterative:

For i from 0 to n=6, result=result*x.
result is initialized as 1.

i=0	result = result * x = 1 * x = x
i=1	result = result * x = x * x = x^2
i=2	result = result * x = x^2 * x = x^3
i=3	result = result * x = x^3 * x = x^4
i=4	result = result * x = x^4 * x = x^5
i=5	result = result * x = x^5 * x = x^6

Recursive:

If n equals 1, pow(x,n)=x
Otherwise, pow(x,n)=pow(x,(n-1))*x

aa=(return value from pow(x, 6))
main

aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
main

aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
main

aa=x*(return value from pow(x,3))

n=4
aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
Main

aa=x*(return value from pow(x, 2))
n=3
aa=x*(return value from pow(x, 3))
n=4
aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
Main

aa=x*(return value from pow(x, 1))
n=2
aa=x*(return value from pow(x, 2))
n=3
aa=x*(return value from pow(x, 3))
n=4
aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
Main

result=x
n=1
aa=x*(return value from pow(x, 1))
n=2
aa=x*(return value from pow(x, 2))
n=3
aa=x*(return value from pow(x, 3))
n=4
aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))

n=6
aa=(return value from pow(x, 6))
Main

aa=x*x=x^2
n=2
aa=x*(return value from pow(x, 2))
n=3
aa=x*(return value from pow(x, 3))
n=4
aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
Main

aa=x* x^2= x^3
n=3
aa=x*(return value from pow(x, 3))
n=4
aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
Main

aa=x* x^3= x^4
n=4
aa=x*(return value from pow(x, 4))
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
Main

aa=x* x^4= x^5
n=5
aa=x*(return value from pow(x, 5))
n=6
aa=(return value from pow(x, 6))
Main

aa=x* x^5= x^6
n=6
aa=(return value from pow(x, 6))
main

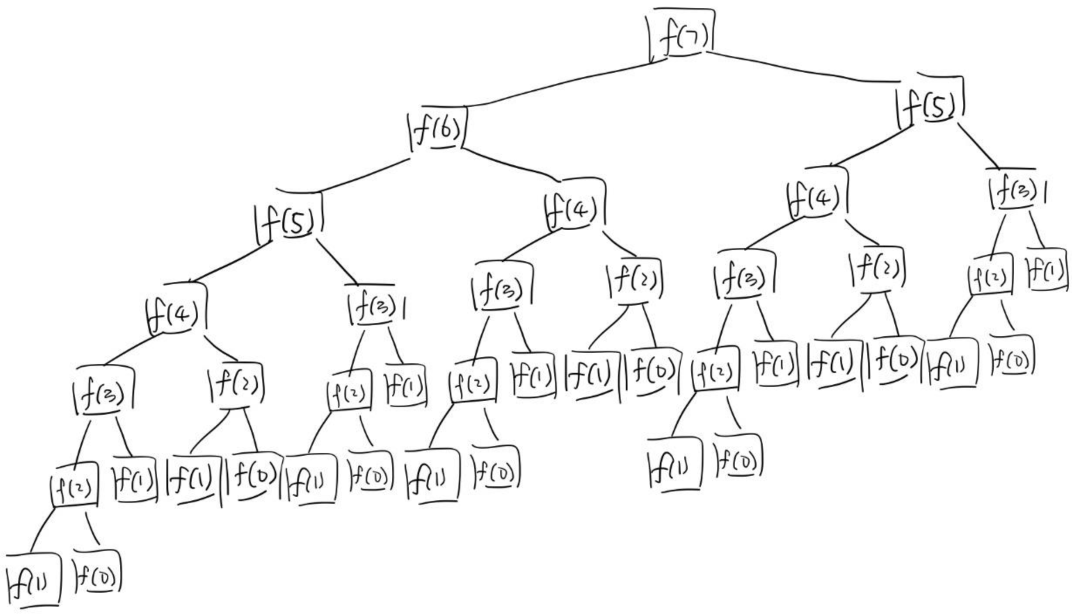
aa= x^6
main

D)

a) It's recursive

b) Tree structure

c)



n=7 is a combination of the diagrams of n=6 and n=5.

d)

Recursive algorithms are easier for coding because they are easy to be understood.

Iterative algorithms are more space saving and time saving. As we can see in the diagram before, the tree structure for the recursive algorithm shows that f(2), f(3), f(4), f(5) and f(6) are calculated repeatedly for many times. When using iterative algorithm, there won't be such a trouble.

3.

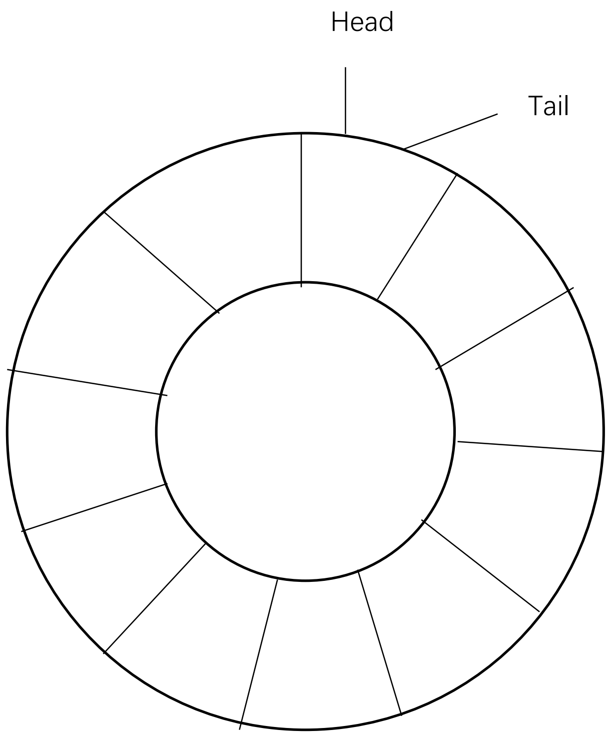
When n equals 1, returns array {0, 1}.

Otherwise, create an array with double size of that of n-1. For the first half of the array, add "0" before each element of that of n-1. On the other half of the array, add "1" before each element of that of n-1 but in the reverse order. Return the newly created array.

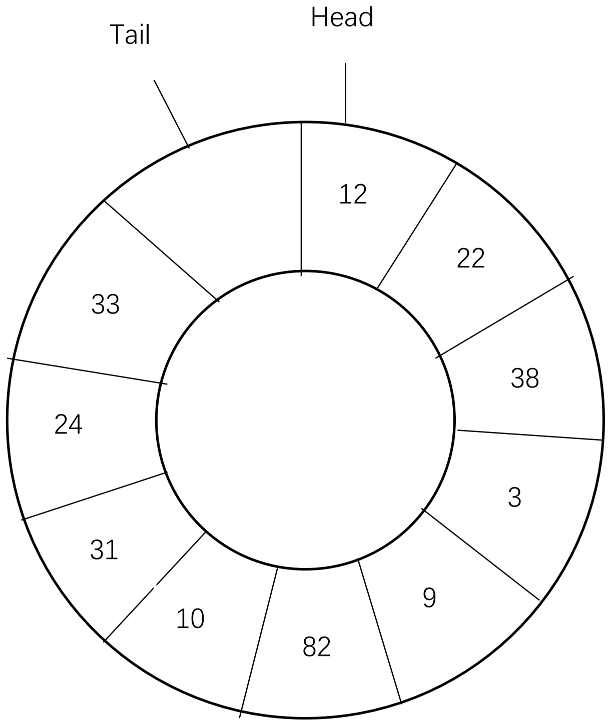
In the case of n=6, it first needs the array of n=5, which needs that of n=4, so on and so forth

until n=1 which returns the array {0, 1}.
Then we got the array of n=2 with the algorithm: {00, 01, 11, 10}.
Then we got the array of n=3 with the algorithm: {000, 001, 011, 010, 110, 111, 101, 100}.
So on and so forth until we finally get the array of n=6, which is: {000000 000001 000011 000010 000110 000111 000101 000100 001100 001101 001111 001110 001010 001011 001001 001000 011000 011001 011011 011010 011110 011111 011101 011100 010100 010101 010111 010110 010010 010011 010001 010000 110000 110001 110011 110010 110110 110111 110101 110100 111100 111101 111111 111110 111010 111011 111001 111000 101000 101001 101011 101010 101110 101111 101101 101100 100100 100101 100111 100110 100010 100011 100001 100000}.

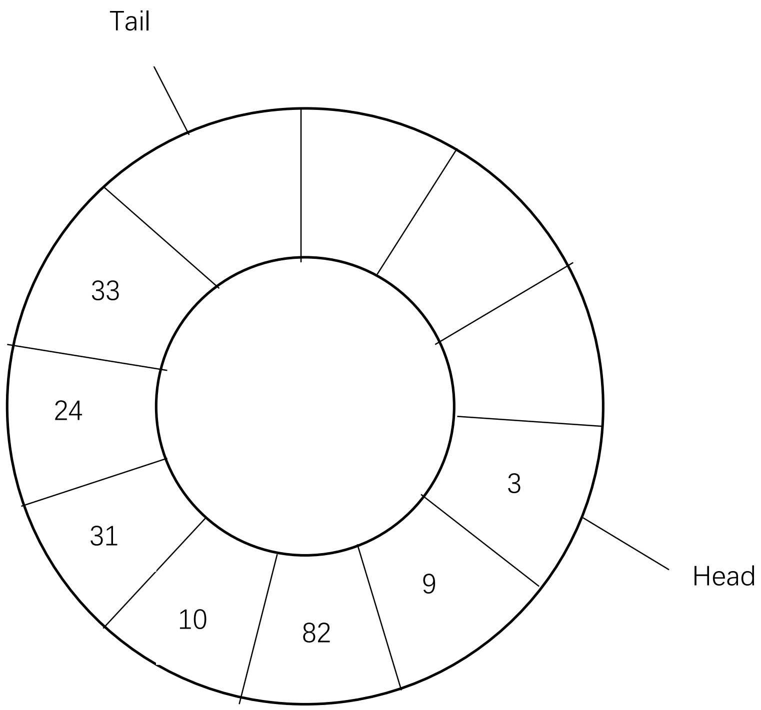
4.



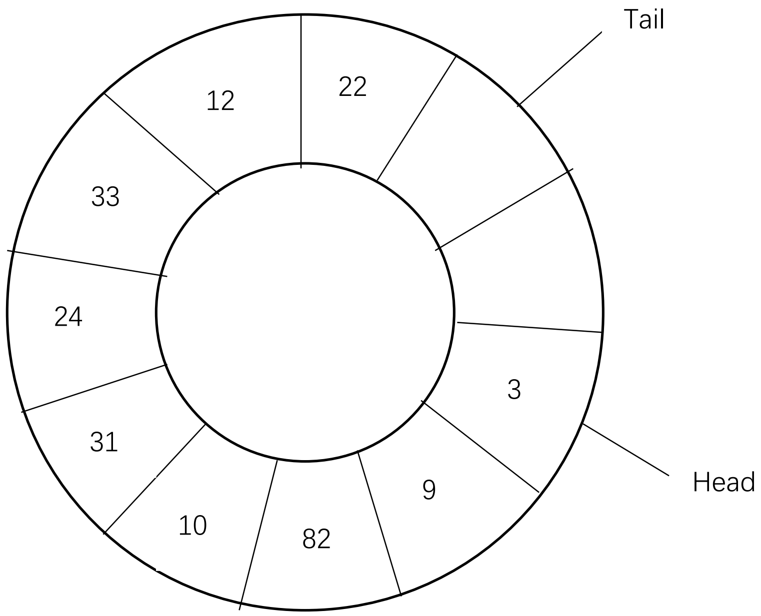
enqueuer all input data:



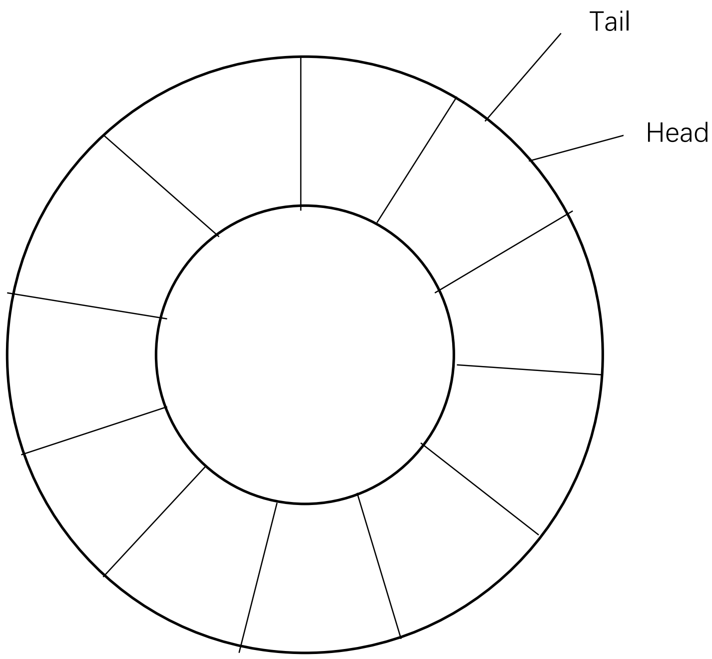
dequeue three elements:



enqueue two elements:

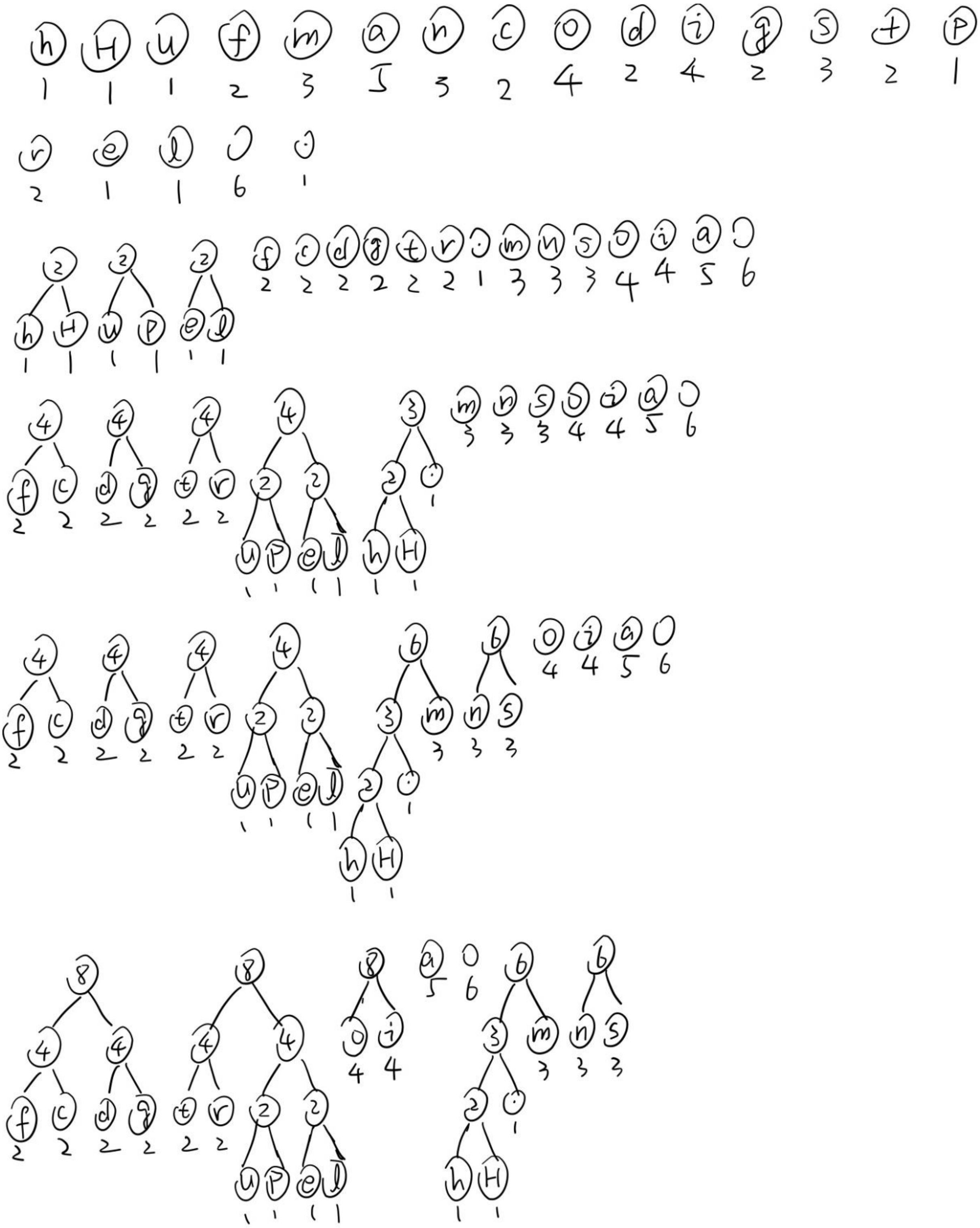


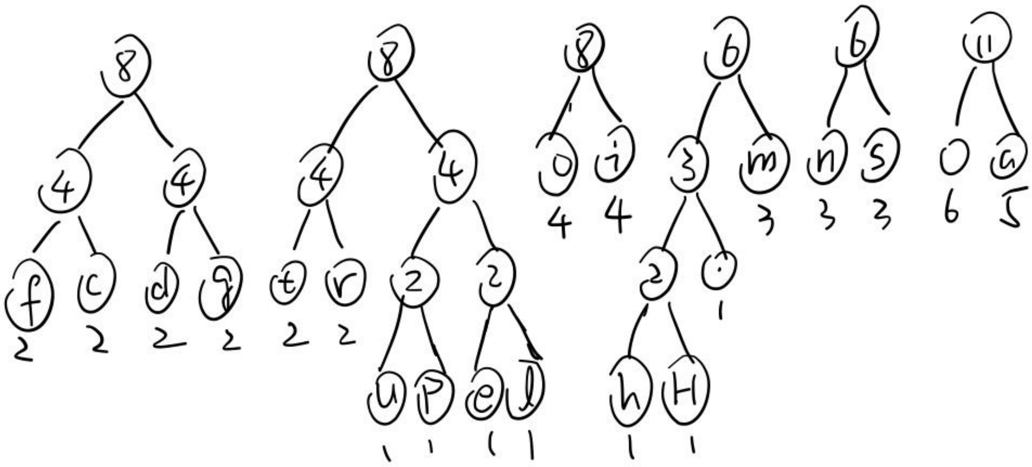
dequeue all elements:

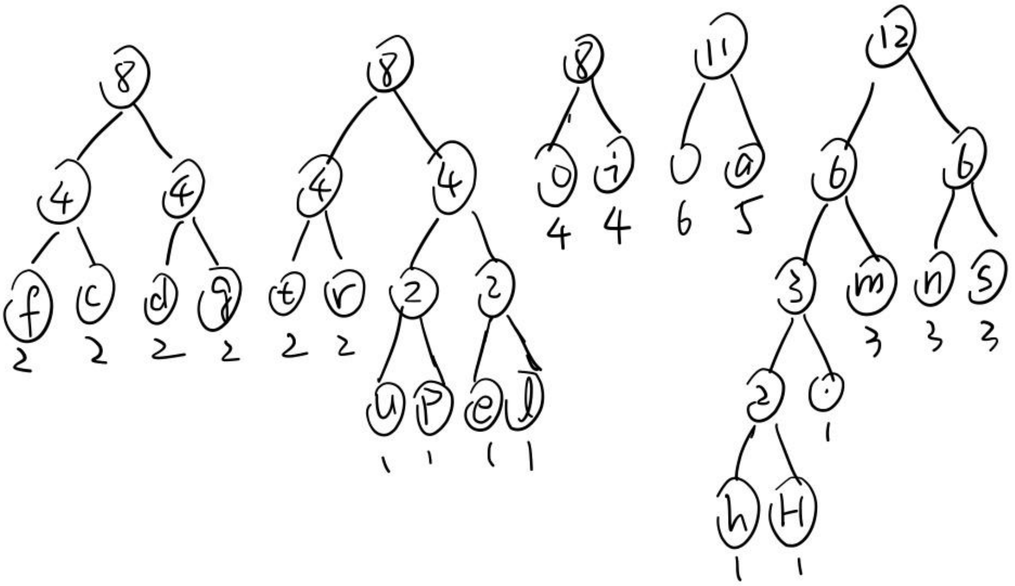


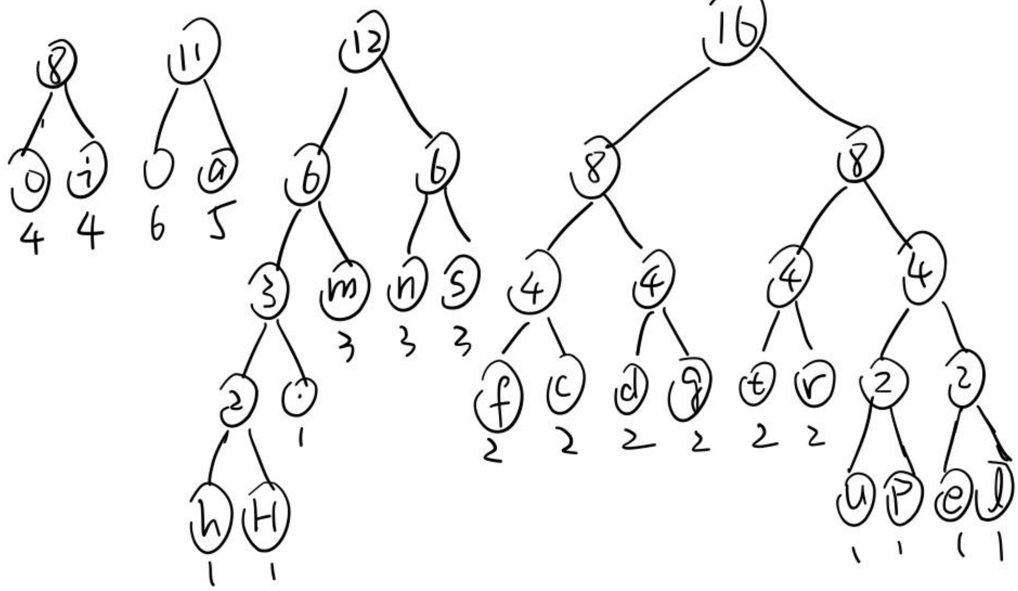
5.

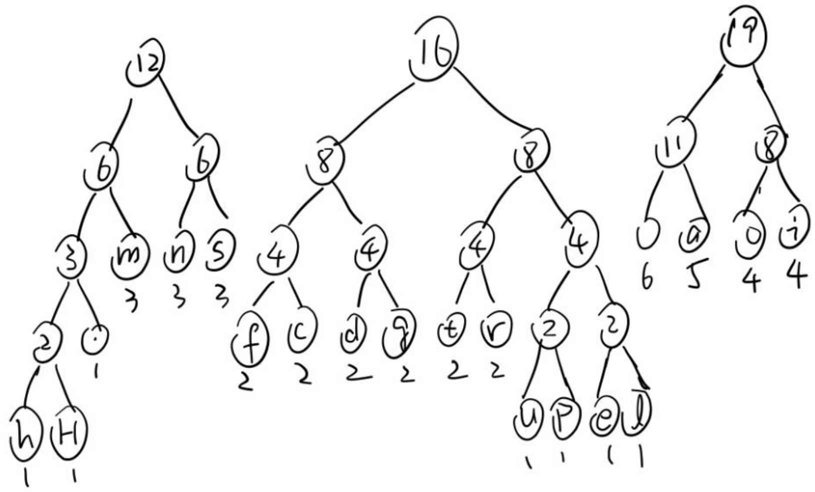
a)

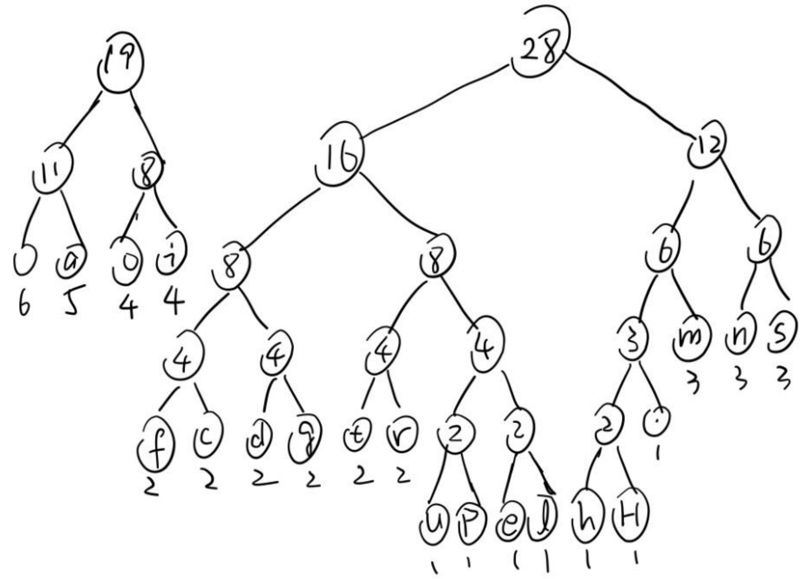


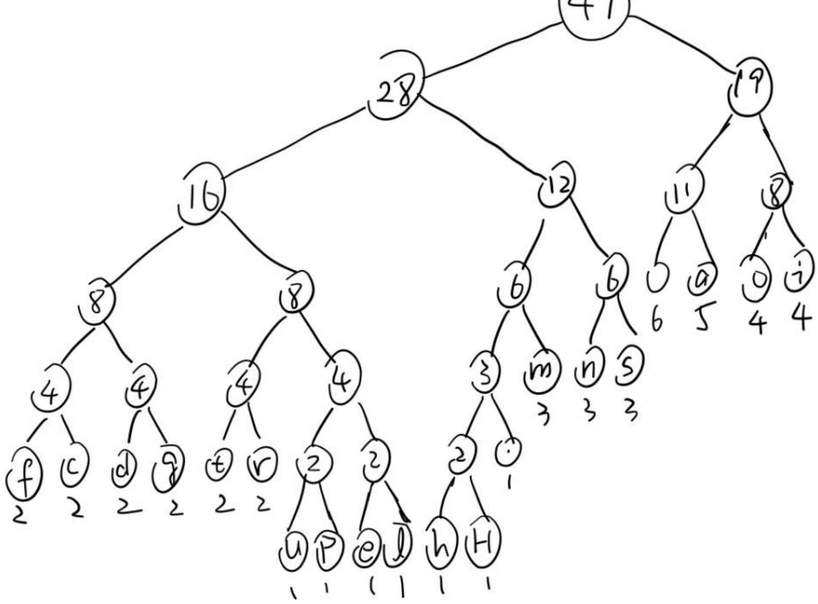












Suppose ‘0’ is used for left edges and ‘1’ for right edges
Then the Huffman code is:
‘f’:00000 ‘c’:00001 ‘d’:00010 ‘g’:00011 ‘t’:00100 ‘r’:00101 ‘u’:001100 ‘p’:001101 ‘e’:001110 ‘l’:001111 ‘h’:010000 ‘H’:010001 ‘.’:01001 ‘m’:0101 ‘n’:0110 ‘s’:0111 ‘ ‘:100 ‘a’:101 ‘o’:110 ‘i’:111

b)

char	ASCII	Before	After
f	102	1100110	00000
c	99	1100011	00001
d	100	1100100	00010
g	103	1100111	00011
t	116	1110100	00100
r	114	1110010	00101
u	117	1110101	001100
p	112	1110000	001101
e	101	1100101	001110
l	108	1101100	001111
h	104	1101000	010000
H	72	1001000	010001
.	46	0101110	01001
m	109	1101101	0101
n	110	1101110	0110
s	115	1110011	0111
(space)	32	0100000	100
a	97	1100001	101
o	111	1101111	110
i	105	1101001	111

Before compression, 7 bits are needed for each char, but only 6 bits are needed for each char at most after compression.
Before compression, 47*7=329 bits are needed for the whole sentence
After compression, 194 bits are needed for the whole sentences, 135 bits, 41.0% of the space is saved.

d) The Huffman coding needs to compare the frequency of the appearance of each letter. A PriorityQueue can help identify which one has the highest frequency. The algorithm implements PriorityQueue through comparing the frequency integer in each node, and returns the node that has the lowest frequency while polling.

f)
The result of the program is:
11000110101100000000011001001111000011111011001111101110001100111110111000101001100101011011010100001111100110110101001011000010111011111111100111100010101010000111100010111111011110100011010100

194 bits in total which is exactly the same as that I did step-by-step. However, the Huffman code is not exactly the same:

{ =100, a=010, c=0011, d=11001, e=110000, f=0000, g=0001, H=110001, h=110100, i=1111, l=101010, m=0110, n=0111, .=10100, o=1110, p=110101, r=0010, s=1011, t=11011, u=101011}

The reason is that when I choose the nodes with the same frequency, I didn't choose them exactly the same order as that the computer did. So it results in the difference of Huffman code. However, for the reason that they have the same frequency, it won't have any influence on the final result. The space the Huffman code cost should be exactly the same.