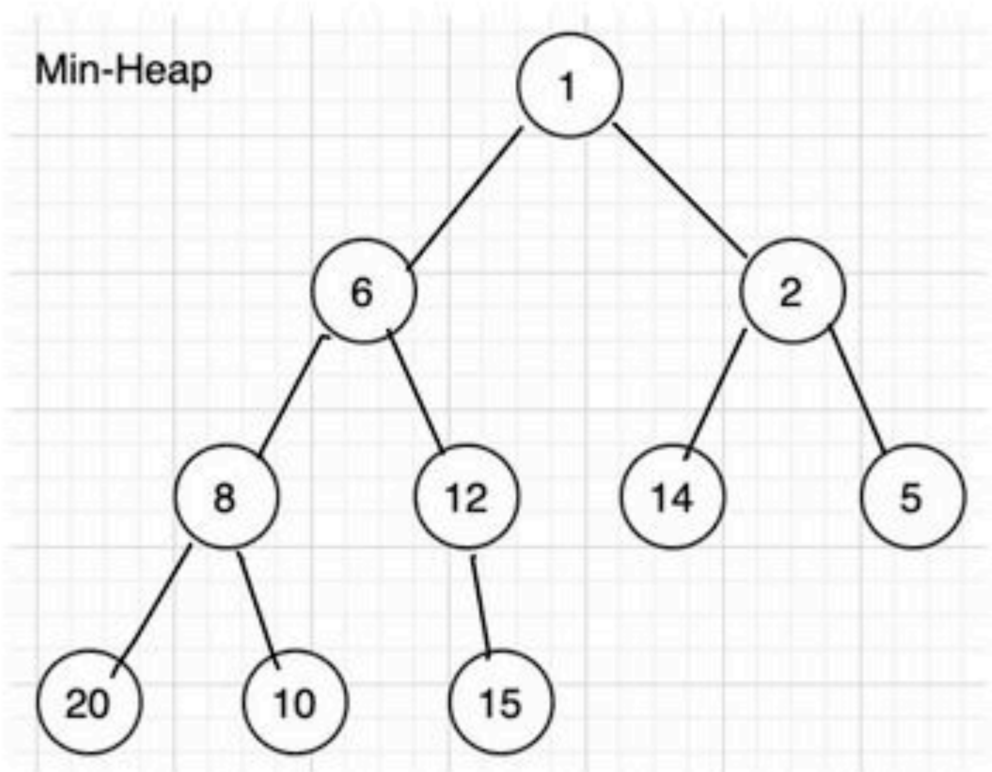
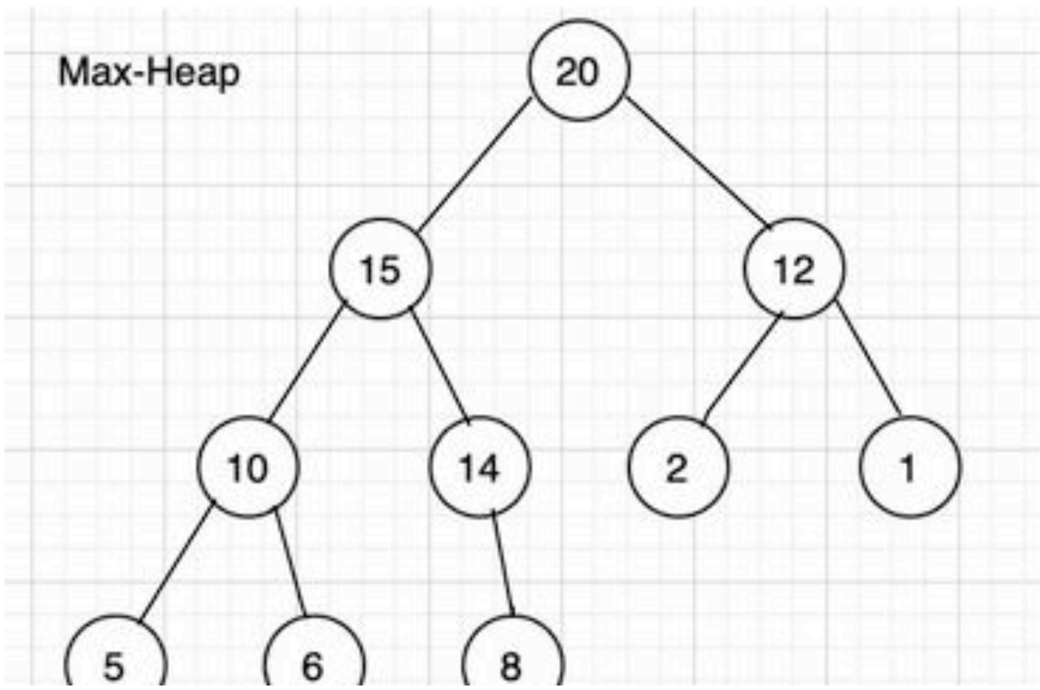
6.

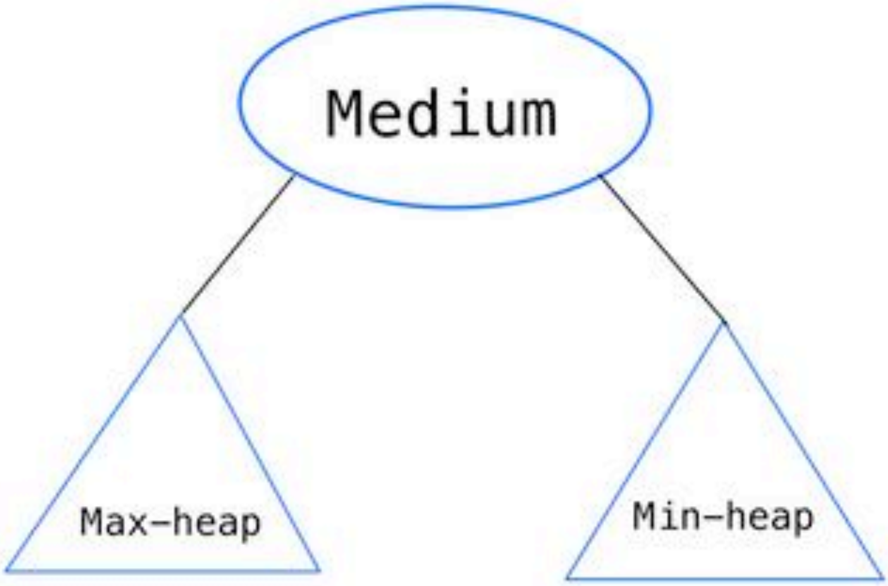
(a)



(b)



(c)



Assume min-heap and max-heap are given. Every element in the min-heap is greater or equal to the median, and every element in the max-heap is less or equal to the median. And the median is linked with max-heap as the left child, min-heap as the right child.

And the functions `Q.pop(), Q.push()` for heap Q in $O(logn)$ time are given,

`find_min(), find_max()` in $O(1)$ time are also given.

```
H: medium-heap
max: max-heap (Every element in the max-
```

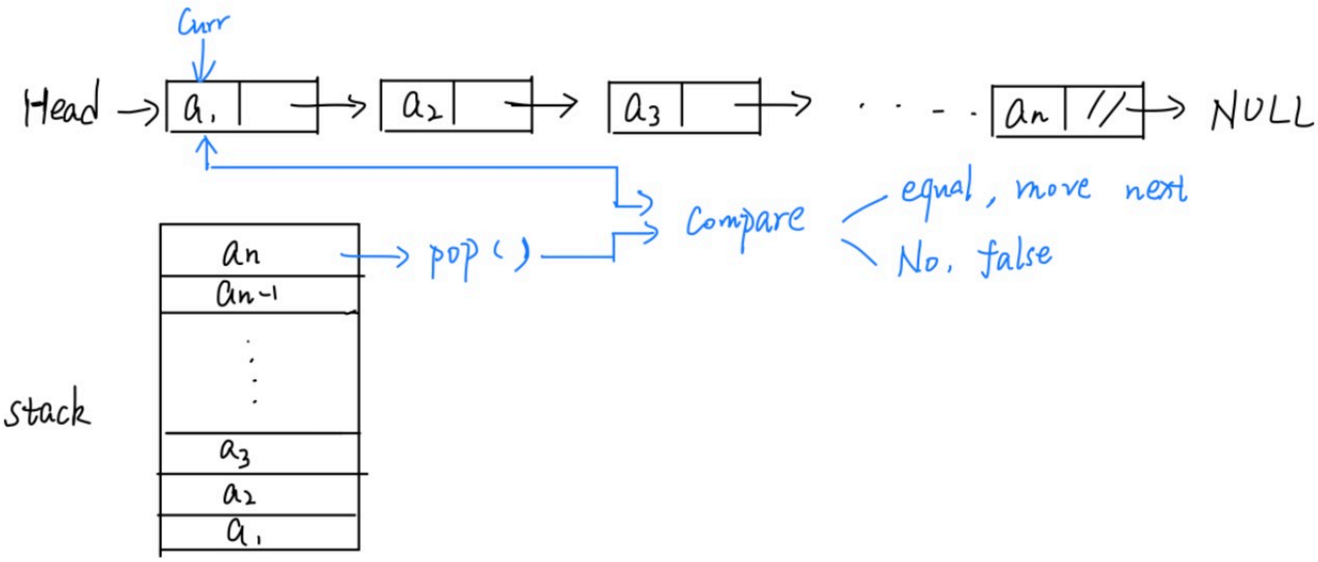
```
heap is less or equal to the median. The
root node link H[1] as the left child)
min: min-heap (Every element in the min-
heap is greater or equal to the median.
The root node link H[1] as the right
child)
Let maxSize = length(max)
Let minSize = length(min)

H.push():
      Let n = length(H)
    Insert ele into H
      n = n+1
      //insert element into the heap
      if (ele > H[1])
        min.push(ele)
      Else
        max.push(ele)

       //Assign median value
      if (minSize > maxSize) then
//Retrun and remove the root node of min-
heap
        H[1] = min.pop()
      Else if (maxSize > minSize) then
  //Retrun and remove the root node of
max-heap
```

```
max heap
          H[1] = max.pop()
        Else if (minSize = maxSize)
          H[1] = (H[2]+H[3])/2


        //Balance the heap
         let temp = 0
         if (minSize > maxSize + 1 ) then
           temp = min.find_min()
           max.push(temp)
         Else if (maxSize > minSize + 1) then
           temp = max.find_max()
           min.push(temp)


H.find_medium():
    let median = H[1]
    return median

```

Time Complexity O(n log n).

5.

My idea is to push all the elements of the list into a stack. Then use the stack.pop() to get the latest element in the stack and compare it with the list since the start of the linked list. If it is equal, a pointer would move next and compare the second element in the list with stack.pop(), and keep going. The below figure shows my idea.



Once there are any numbers that do not equal, it means this list is not palindrome. Otherwise, when the stack becomes empty, the list is palindrome.

```
Checkpalindrome :
stack S     //Declare a stack S

//Push all element of the linked list into the stack
Initial curr = head          //curr is a pointer
while (curr ≠ NULL)
    S.push(curr)            //Push in the stack
    curr = curr.next        //next one of the current

curr = head                 //reset the pointer
while (curr ≠ NULL)
    let tmp = s.pop()       //Pop the latest element in the stack
    if (curr ≠ tmp) then
    return -1             //It is not palindrome
    Endif
    curr = curr.next      //Move next

return true               //It is palindrome


```

Time complexity O(n)

4.

(a)

Transpose symmetry.

Proof:

By the definition of Asymptotic upper bounds,

$f(n) = O(g(n)) => f(n) \leq c \cdot g(n)$

for some constants $c > 0$ and $n_0 \geq 0$ for all $n \geq n_0$. It can be transformed as

$g(n) \geq (\frac{1}{c}) \cdot f(n) \quad$ where $\frac{1}{c} > 0$ for all $n \geq n_0$.

Then by the definition of asymptotic lower bounds, we have $g(n) = \Omega(f(n))$. Proved.

(b)

Proof:

By the definition of Asymptotic upper bounds,

$f(n) = O(g(n)) => f(n) \leq c \cdot g(n)$

for some constants $c > 0$ and $n_0 \geq 0$ for all $n \geq n_0$. It can be transformed as

$f(n) \cdot g(n) \leq c \cdot g(n)^2$

So the $f(n) \cdot g(n) = O(g(n)) \cdot O(g(n)) = O(g(n)^2)$ by the definiton. Proved.

(c)

Disprove by counterexample:

Suppose $f(n) = 2n,\ g(n) = n$ such that $f(n) = O(g(n))$ since $2n \leq c \cdot n$ for any constant $c \geq 2$.

$2^{2n} \notin O(2^n)$ since we cannot find a constant $c > 0$ for $2^{2n} \leq c \cdot 2^n$. $\quad (c = \frac{2^{2n}}{2^n} = 2^n$ which is not constant.$)$

3.

Since $f_4(n)$ and $f_5(n)$ are exponentional function and others are polynomial function, $f_4(n)$ and $f_5(n)$ will grow the fastest, so they should be placed at end of the function list. And $f_4(n)$ < $f_5(n)$ because 10 < 100.

For the polynomial functions,

$f_1(n) = n^{2.5}$ has degree of 2.5

$f_2(n) = \sqrt{2n} = \sqrt{2}n^{0.5} = O(f_3)$

Since $\lim_{x\to\infty} \frac{n^2 \log n}{n+10} = \infty,$

so $f_3(n) = O(f_6)$.

$f_6(n) = n^2 log(n) = O(f_1)$

The result of the arranged list is:

$f_2(n) < f_3(n) < f_6(n) < f_1(n) < f_4(n) < f_5(n)$

2.

(a)

Assume S = ∅ and all elements are free, start from m_1 as the first who proposes to w. Follow m_1's preference ranking, m_1 proposes to w_1. Since w_1 is unmatched, add (m_1 - w_1) to S.

Then m_2 proposes to w_2 according to m_2's preference ranking list. And since w_2 is free, add (m_2 - w_2) to S.

At this point, S = {(m_1 - w_1), (m_2 - w_2)}.

(b)

Suppose a stable matching S that m_1, m_2 are not matched to w_1, m_2.

In other words, these elements are assigned with partners which do not rank 1st or 2nd in their own preference list. In this situation, m_1, m_2 both prefer w_1 or w_2 to their assigned partners; similar, w_1, w_2 would also both prefer m_1 or m_2 to their assigned partners. i.e, m_1, m_2 and w_1, w_2 cannot stay with their current assigned partners.

S would becomes unstable which is contradictory. Therefore, m_1, m_2 has to be matched to w_1, w_2 in every stable matching S.

1.

True.

Proof: Assume there is a stable matching S which does not have `(m, w)`, but since `m` and `w` both are ranked first on their own preference list of each other, which mean they would prefer each other over their assigned partner, and it is **unstable** (contradictory). Therefore, in every stable matching S, the pair `(m, w)` must belongs to S.