(a)

$$T(N) = \overset{a}{2}T\left(\frac{N}{\underset{b}{2}}\right) + NlogN, \qquad T(1) = 1$$

①基准函数 $\Theta(n^{\log_2 2}) = \Theta(n)$ ②$f(n) = n\log n$
$= n\cdot\log^{1} n$
$\therefore T(n) = \Theta(n^{\log_b a}\log^{k+1} n)$
$= \Theta(n\log^{2} n)$

(b)

$$T(N) = 4T\left(\frac{N}{2}\right) + N, \qquad T(1) = 0$$

①基准函数 $\Theta(n^{\log_2 4}) = \Theta(n^2)$ ②$f(n) = n$ $\therefore T(n) = \Theta(n^2)$

(c)

$$T(N) = T\left(\frac{N}{2}\right) + 2^N, \qquad T(1) = 1$$

①基准函数 $\Theta(n^{\log_2 1}) = \Theta(1)$ ②$f(n) = 2^n$
$c<1$时足够大的n
有 $2^{\frac{n}{2}} < c2^n$ 成立
$\therefore T(n) = \Theta(2^n)$

```
Merge(arr, left, middle, right)
    p1<-left, p2<-middle+1
    A<-arr[left:middle]
    A[middle-left+1] <- ∞
    B<-arr[middle+1:right]
    B[right-middle] <- ∞

    for p=left to right
        if(tmp[p1]<=tmp[p2])
            arr[p]<-tmp[p1]
            p1++,
        else
            arr[p]<-tmp[p2]
            p2++

Merge_Sort(arr, left, right)
    if left<right
        middle<-floor(left+right)/2
        Merge_Sort(arr, left, middle)
        Merge_Sort(arr, middle+1, right)
        Merge(arr, left, middle, right)
```

假设有六个项的权重为（6、2、4、3、9、12），值为（9、4、6、5、14、20）。背包容量为16。

```
class NODE:
    v_now=0
    w_now=0
    is_in=0
    i=0
    bound=0
    def __init__(self, v_now, w_now, is_in, i):
        self.v_now=v_now
        self.w_now=w_now
        self.is_in=is_in
        self.i=i
        self.bound=Bound(self)
    def Bound(self):
        i=self.i+1  # i就是二叉树的哪一层
        v_now=self.v_now
        w_now=self.w_now
        cleft = c-w_now  # 剩余背包容量
        bound = v_now  # 现有价值
        while(index[i]<n and w[index[i]]<=cleft):
            cleft-=w[index[i]]
            v_now+=v[index[i]]
            i++
        if index[i]<=n:
            v_now+=v[index[i]]/w[index[i]]*cleft
        return bound

C=16
v = [9,4,6,5,14,20]
w = [6,2,4,3,9,12]
v_w = [9/6, 4/2, 6/5, 14/9, 20/12]
index = np.argsort(-v_w)  # 降序获得v/w排序后的索引

best_v = 0

que = queue.PriorityQueue()
max_bound = v[index[0]]/w[index[0]]*C
# 优先级：max_bound - bound，谁越小，谁越先


def Maxknapsack(now):
    while now.i!=n:
        if(now.w+w[index[i]]<=c):  # 放入（左孩子）为活结点
            if now.v+v[index[i]]>best_v:  # 可能会更新当前最优解
                best_v = now.v+v[index[i]]
            NODE node(now.v+v[index[i]], now.w+w[index[i]], true, i)
            que.put((max_bound-node.bound), node)  # 将这个二叉树结点加到优先级队列里

        #若不放入这个物品
        NODE node(v[index[i]], w[index[i]], false, i)
        if node.bound>best_v:  # 如果不放入这个物品也可能有更优解
            que.put((max_bound-node.bound), node)  # 将这个二叉树结点加到优先级队列里

        # 左孩子和右孩子都已经检查了，取下一个扩展结点
        now=que.get()
    return best_v

NODE now
best_v=MAXknapsack(now)
```