系級：大氣二　　學號：B08209006　　姓名：謝晉維

解法：

1. 先建立一個list (k_sum) 存放subarray加起來的值。例如說
k_sum[3] = k_sum[2] + array[2]，而k_sum[2]是前面的k_sum[1] + array[1]，則k_sum[3]其實就是array[0] + array[1] + array[2]，也就是index 0~2這個subarray的和。

2. 建立一個min heap (minHeap)，檢查tmp = k_sum[j] - k_sum[i-1]（這個步驟其實是去算任意一個subarray的值，例如說要看index 5~7這個subarray的和，其實就是k_sum[7] - kum[5]）。如果minHeap中的node數目還沒超過k就直接將tmp存放進去，若是超過k則比較tmp跟minHeap最小的值的大小，若tmp比較大就pop minHeap存入tmp。這個步驟實質上就是在算top-k sum的概念。

p.s. i是從1~n, j是從i~n

3. 將minHeap中的值排序（reverse）存到json_sum中輸出即為答案。

資料結構：

1. min heap:
在紀錄前k大的subarray時，利用了minHeap，此結構能夠快速的找到最小值（第一個）並且刪掉。

2. List:
在讀取json的dictionary時是存到list之中，將算好的minHeap排序後也是存到一個list之中。

時間複雜度：

1. 推導

a. k_sum的計算（紀錄subarray的和例如k_sum[5]存array[0] + ⋯ + array[4]）：
總共跑一個從2~n的迴圈，複雜度為 $\Theta(n)$。

b. minHeap list的建立及存放：
需要跑一個巢狀迴圈從i = 1 ~ n, j = i ~ n，並且在迴圈中計算tmp = k_sum[j] - k_sum[i-1]，如果minHeap中的數目小於k則直接存入，tmp若大於等於k則判斷

tmp及minHeap[0]誰比較大，若tmp比較大則pop最小的並且將tmp存入minHeap中。上述若有push值或是pop值到minHeap中，都需要維持minHeap的性質。

因為迴圈中每一步都需要計算tmp（$\Theta(1)$），若對min heap有操作則需要維持minHeap的性質（$O(logk)$），又因為總共跑$O(n^2)$步，所以步驟b的時間複雜度為$O(n^2logk)$。

c. 結果的排序：
最後將minHeap中的結果排序（利用list內建的Sort，預設應為Quick Sort）存入json_sum中，排序（Quick Sort）的複雜度為$O(nlogn)$，此處的n為queue中的數目意即k的大小。

2. 結果
由上述推導可知，在步驟b的複雜度最高，因此總體時間複雜度應為$O(n^2logk)$。