Asymptotic time complexity:
n_document = m; n_results = k; num_terms = n; average length of doclists = p

Task 1:

Reset array to store every score of the document	O(m)
Add score of each id to the array	O(np)
Create min-heap of size of results	O(1)
Loop through the array	m *
If heap size is smaller than results, add the item	O(log(k))
If score is larger than the min score, replace the item	O(log(k))
Print number of results items with largest scores	O(klog(k))
Free the heap	O(1)

Total: O(m+np+mlog(k)+klog(k))
Cost: O(np+mlog(k))

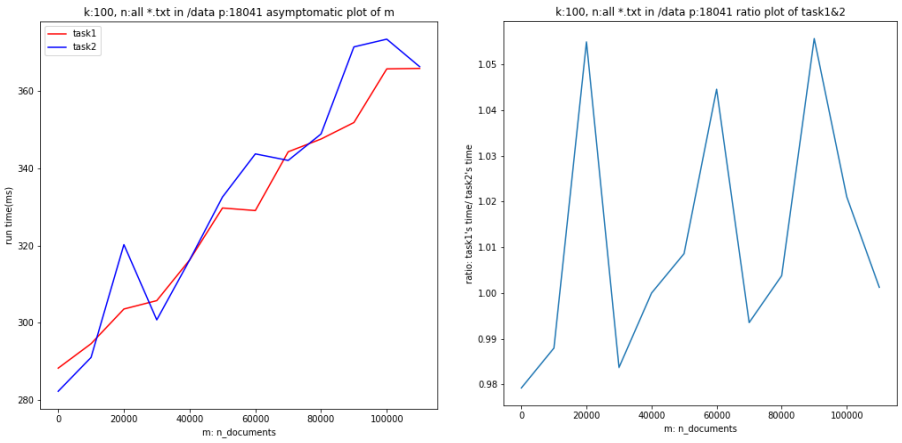
Task 2:

Create 2 heaps data and resu store ids and scores as keys	O(1)
Create an array to store the nodes currently processed	O(1)
Put head nodes of terms to heap data	O(nlog(n)
Get the min id in the heap data, remove it and replace it with the next node in the same term. Loop until the min id in the heap data changes. Add their scores.	O(nplog(n))
Loop to put the score to heap resu	m *
If heap size is smaller than results, add the item	O(log(k))
If score is larger than the min score, replace the item	O(log(k))
Print number of results items with largest scores	O(klog(k))
Free the heap	O(1)

Total: O(nplog(n)+mlog(k)+nlog(n)+klog(k))
Cost: O(nplog(n)+mlog(k))

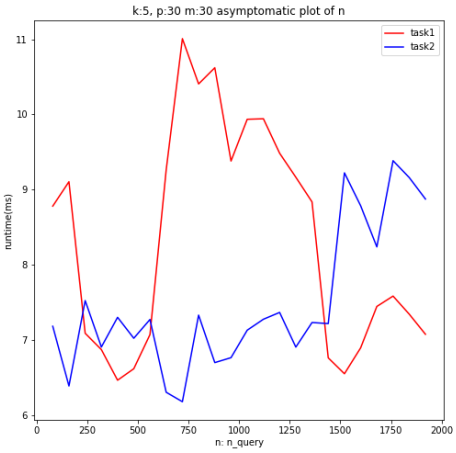
(The analysis line by line will be attached in the zip.)

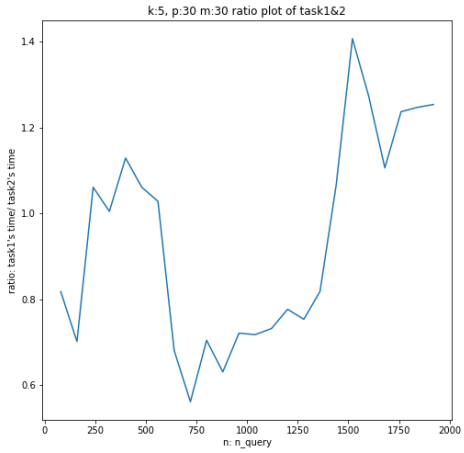
For input parameters:



For n_documents:

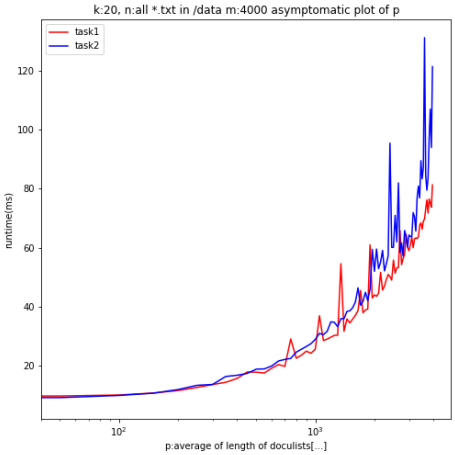
As n_documents increases every time, both costs will increase log(k), where k is number of results, time since both equations only have mlog(k) contains m. As the graph shown, the costs increase in similar rate. Besides, since O(nplog(n)) is always larger than o(np), the cost of task 2 will always larger than task 1. In the left diagram the line of task 1 is mainly below the task 2, and for the right diagram, except certain points most time the ration is about or below 1.0.

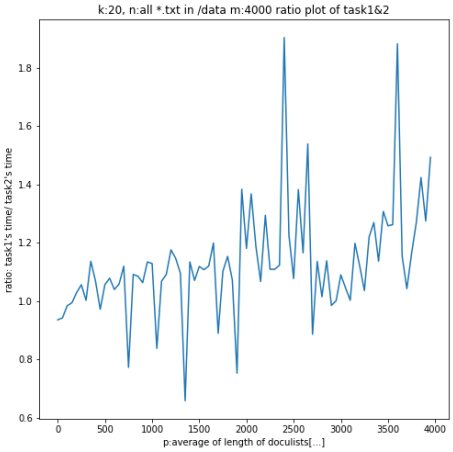




For num_terms (query):

As num_terms increase every time, the cost of task 1 will increase p which is the length of doclists, time since the complexity contains t is np. However, for task 2 since the big O contains t is nplog(n), as n increases, both n and log(n) increase. Therefore task 2 increases more than task 1. At first the cost of task 2 is larger than task 1, so as t increase the gap between two costs will increase. As the left graph shown, as query increases, the line of task 1 is oscillating but the cost of task 2 is increasing. For the right graph, the ration is mainly smaller than 1.0.



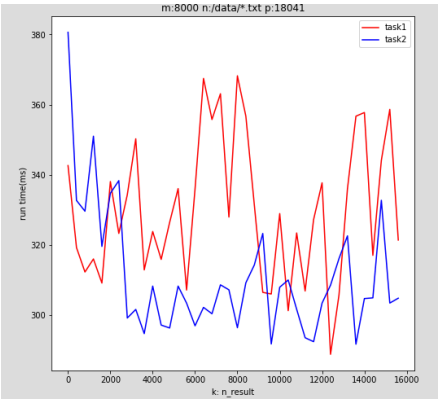


For the length of doclists:

As length of doclists increases every time, the cost of task 1 will increase n times, which n is the number of query time. Moreover the cost of task 2 will increase nlog(n) time, so task 2 will increase more than task 1. As what the left diagram shown as length increases the line of task 2 increase more than task 1, and in the second graph the ration decreases as length increases.

For n_results:

As number of results increases every time, both costs will increase in the association of mlog(k). Because the increase is in the log, the increasing rate will be relatively low. As the right graph shown, the lines of two tasks are almost similar and increase little.



Conclusion：

As a conclusion, compared to task 1, task 2 takes more costs to run the function and increases more as number of query and length of doclists increases. They have similar increasing rate when number of documents and results increases. However, by using heap instead of array, task 2 save more space than task 1, which create an array to store all items. Therefore both task have their own advantages and we had better choose the right method in specific situation to get our purpose.