```
n_documents: m
n_results: k
num_terms: n
average size of doclists: p


void print_array_results(Index *index, int n_results, int n_documents) {

   float total[n_documents];                                                    O(1)
   for (int i = 0; i < n_documents; i++) {                                      Loop m times
     total[i] = 0.0;                                                            O(1)
   }
   for (int j = 0; j < index->num_terms; j++) {                                 Loop n times
      for(Node* node = index->doclists[j]->head; node != NULL; node =
node->next) {                                                                   Loop p times
         total[((Document*)(node->data))->id] += ((Document*)(node->data))-
>score;                                                                         O(1)
      }
   }

   Heap *heap = new_heap(n_results);                                            O(1)
   for (int k = 0; k < n_documents; k++) {                                      loop m times
      if (k < n_results) {
         heap_insert(heap, total[k], k);                                        O(log(k))
      } else if (total[k] > heap_peek_key(heap)){
         heap_remove_min(heap);                                                 O(1)
         heap_insert(heap, total[k], k);                                        O(log(k))
      }
   }
   print_task(heap, n_results);                                                 O(klog(k))
   free_heap(heap);                                                             O(1)
                                                                                Total:
                                                                                O(n)+O(m)+O(np)+
                                                                                O(mlog(k))+O(klog
                                                                                (k))
}
```

```
void print_merge_results(Index *index, int n_results) {
   Heap *data = new_heap(index->num_terms);                                     O(1)
   Heap *resu = new_heap(n_results);                                            O(1)
   Node* node[index->num_terms];                                                O(1)
   float id, score = 0.0;
   int term;

   for (int i = 0; i < index->num_terms; i++) {                                 loop n times
      node[i] = index->doclists[i]->head;                                       O(1)
      heap_insert(data,(float)(((Document*)(node[i]->data))->id),i);            O(log(n))
   }
   while (data->current_position != 0) {                                        loop p times
      id = heap_peek_key(data);                                                 O(log(n))
      while(heap_peek_key(data) == id) {                                        loop n times
         term = heap_peek_min(data);                                            O(log(n))
         heap_remove_min(data);                                                 O(l)
         score += ((Document*)(node[term]->data))->score;                       O(1)
         if(node[term]->next != NULL) {
            node[term] = node[term]->next;                                      O(1)
            heap_insert(data, (float)(((Document*)(node[term]->data))->id),
term);                                                                          O(log(n))
         }
         if (data->current_position == 0){
            break;                                                              O(1)
         }
      }
      if (resu->current_position < n_results){
         heap_insert(resu, score, (int)id);                                     O(log(k))
      } else if(score > heap_peek_key(resu)) {
         heap_remove_min(resu);                                                 O(1)
         heap_insert(resu, score, (int)id);                                     O(log(k))
      }
      score = 0.0;                                                              O(1)
   }
   print_task(resu, n_results);                                                 O(klog(k))
   free_heap(data);                                                             O(1)
   free_heap(resu);                                                             O(1)
                                                                                Total:
                                                                                O(nlog(n))+O(nplog
                                                                                (n))+O(mlog(k))+O
                                                                                (klog(k))
}
```

```
void print_task(Heap *h, int a){
   while(h->current_position != 0) {                                            loop k times
      heap_remove_min(h);                                                       O(log(k))
   }
   for (int i = 0; i < a; i++) {                                                loop k times
      if(h->items[i].key != 0) {
         printf("%6d %.6f\n", h->items[i].value, h->items[i].key);              O(1)
      }
   }

                                                                                Total:
}                                                                               O(klog(k))+O(k)
```