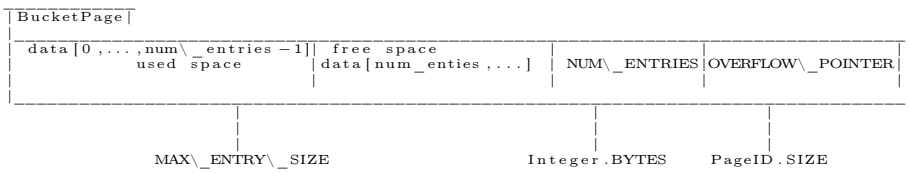
Exercise 1: Static Hash Index

a. (a) BucketPages are prepended to a chain.
From the BucketPage.java class:
The bucket page of hash index. The page stores entries at the beginning of the byte array and meta-data at the end.



(b) On the first page; if the first page does not have sufficient space left, the overflow bucket page is prepended

(c) If the entry is on the first page, it's simply deleted from there by moving the last element of that page to the newly empty spot and invalidating the last spot. Invalidation is done by decrementing the number of entries and due to the invariant that elements are always stored consecutively so that the first free place is always at page address +|entries|.
If the entry is on another page move the last element of the first page to the newly empty spot and decrement |entries| from the first page. The same invariance principle as mentioned above applies.

(d) Elements are always stored consecutively, meaning that they are inserted on the first page at |entries| or on a new page if the former first page was full and removal is done in such a way that gaps are filled by the last entry on the page.

(e) Insert in $O(1)$ as it's a constant amount of work to hash and insert at a given postion (array access, always the first page or a new one).
search is in $O(e \cdot k)$ with e constant (and in average 1.2 IO ops) where the constant part is calculating the hash and look through the e entries of one bucket page. k is the number of overflow pages that one needs to iterate over in order to look at all elements.
Remove is in $O(e \cdot k)$ as one needs to search for the element as described above and then moving/replacing accordingly which is done via arraycopy in constant time

b. an alternative would be to append the pages and store the values with holes but sorted. Advantages are that the search could use binary search and reduce complexity from linear to logarithmic over the number of overflow chains, the removal wouldnt include moving entries, index nested loop joins could be further optimized, using the partial ordering, e.g. extending to non equi joins doing sth. like a block index nested loops join (or block based selection,...). Disadvantages are the obvious costs of the sort, the additional search at insertion and the less compressed storage.

c. e.g. for index only query execution plans or for index nested loop joins (equi-joins only ofc)