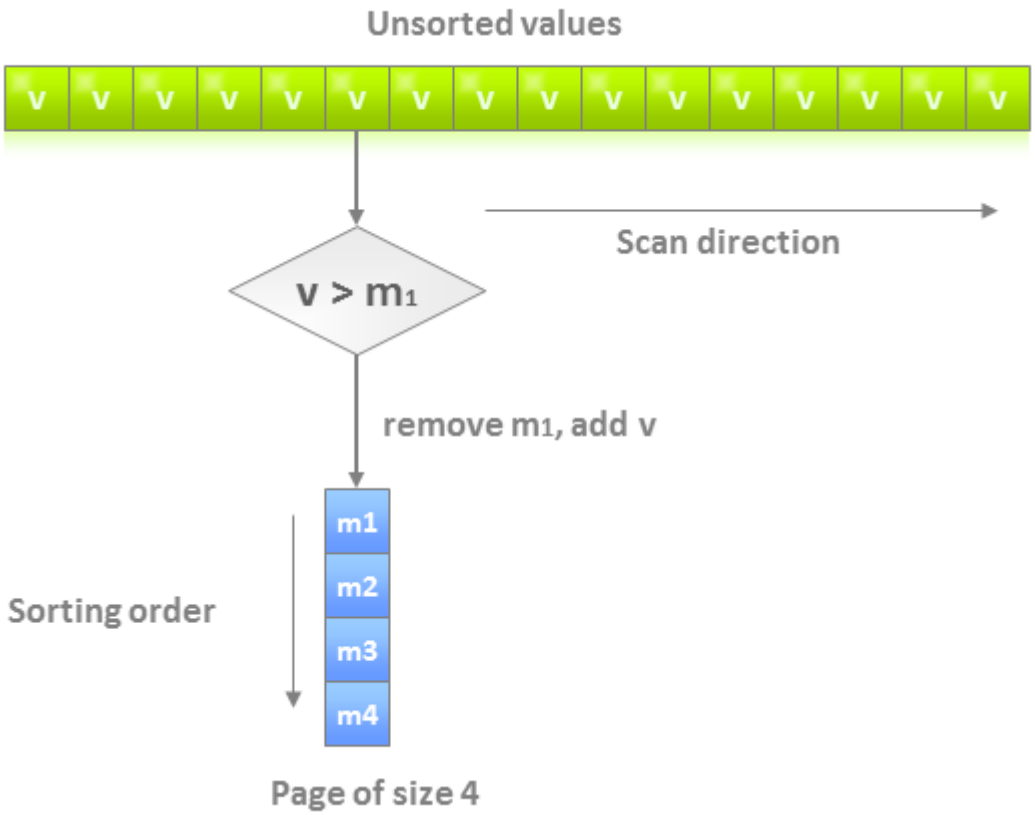
Performance of Priority Queue Sorting with Pagination

In web applications, it is a very common task to sort some set of items according to the user-selected criteria and return only the first or N-th page of the sorted result. The page size can be much less than the total number of items, hence it is typically not reasonable to sort the entire set and crop a one page; it's much more efficient to extract this page on the fly, running through the initial unsorted set. Sorting with priority queue is well know solution for this problem. In this post I present analysis of priority queue sorting for page-oriented use cases.

Let us assume that sorting unit iterates over the unsorted list of items and maintains a sorted page (a queue) of the selected maximums. If the unit meets an item which is greater than the minimal element in the queue then it removes the minimal element from the page and inserts the current item into the queue. This logic is illustrated below:



Let we have N randomly permuted items and we need to extract p maximums. If we have set of k items x_1, ...,x_k and x_k is p-th largest element in this list then we say that rank(x_k)=p. In other words rank(x_k) is a position of x_k in a list after sorting. Let us consider the following case: we scanned k-1 > p items and the page contains p largest items among x_1, ..., x_{k-1}. In this case, next item x_k will be inserted in the page with the following probability (assuming flat distribution of all possible permutations):

$$Pr\{rank(x_k) \geq p\} = \frac{(k-1)! \cdot p}{k!} = \frac{p}{k}, \quad k > p$$

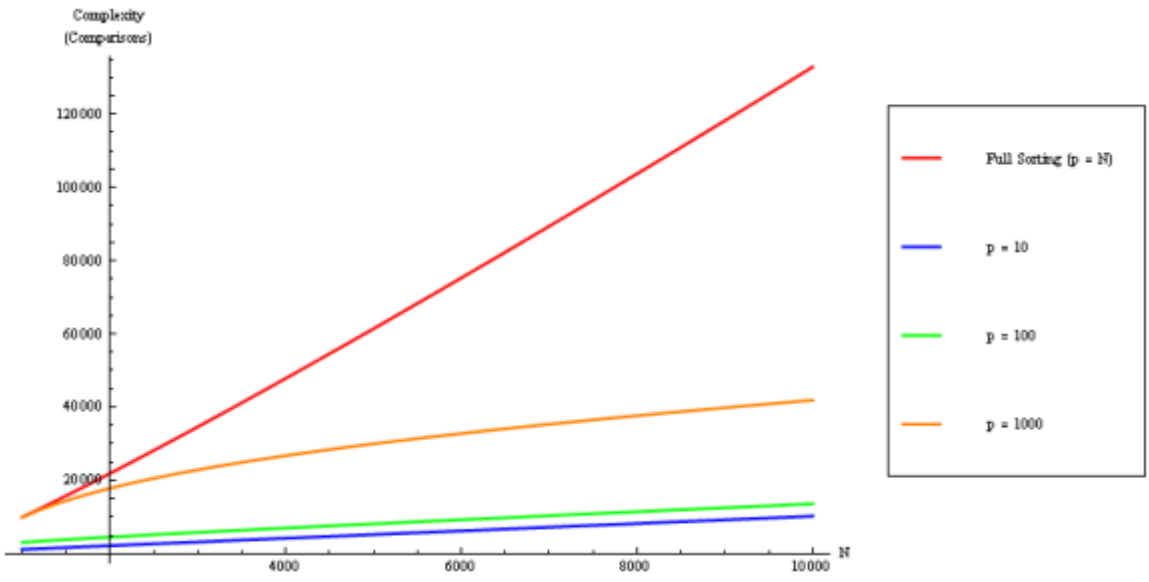
To see this note that we have k! possible permutations of k items and k-th item will be inserted if it is a largest element ((k-1)! permutations) or second largest element ((k-1)! permutations), … or p-th largest element ((k-1)! permutations) . Now we can estimate an average number of item comparisons for page extraction. This number consist of two components. The first one is an initial effort of scanning of first p of N elements – the list that represents the extracted page is initially empty, so we simply add first p items into it and sort them – this requires about p*log p comparisons. The second component is an effort of scanning of the remaining items. It requires N-p-1 comparisons with the page minimal element ("1 + Pr{.." term in the formula below) and insertion into the page with complexity log p, but only in the cases when item has a higher rank than a minimal item in the currently extracted page. Combining all this into one formula and reducing it we obtain that total complexity is about p log p ln N + N:

$$E\{comparisons\} = \underbrace{p \log p}_{sort\ first\ p\ items} + \sum_{k=p+1}^{N} \left(\underbrace{1}_{comparison\ with\ the\ tail} + Pr\{rank(x_k) \geq p\} \cdot \underbrace{\log p}_{insertion} \right) =$$

$$= p \log p + \sum_{k=p+1}^{N} \left(1 + \frac{p}{k} \log p\right) = p \log p \cdot (1 + \ln N - \ln(p+1) + \varepsilon) + N - p - 1$$

At the same time direct sorting of all items is estimated in N*logN comparisons. For example, if we select first 48 items (second page of size 24) from 5000 items then simple sorting will take ~61500 comparisons and page-oriented sorting will take ~6470 comparisons.

The plots below depict complexity (number of comparisons) of page-aware sorting and sorting of entire item set for different values of p and N.



Source: http://highlyscalable.wordpress.com/2012/01/02/sorting-with-pagination/