1 Pointer machine model

This can be found in Chazelle's paper, [Ref. 37] Bernard Chazelle Lower bounds for orthogonal range searching: the reporting case.

2 Multi-partitioning

This lower bound is a short argument using Ben-Or's theorem. It is section 4 of the following paper:

Amr Elmasry: Distribution-sensitive set multi-partitioning.
available at
http://www.kurims.kyoto-u.ac.jp/EMIS/journals/DMTCS/pdfpapers/dmAD0132.pdf

There is also a very interesting upper bound mentioned in this paper, which matches the above lower bound.

3 Multi-partitioning in external memory

Once you've read Section 4 in the above paper, you know that the number of permutations required to check is at least

$$\frac{n/2!}{n_1! \cdots n_k!}$$

In the external memory model, as in the proof of the sorting lower bound, the maximum fan-out at an I/O is $\binom{M}{B}$, since this is the number of ways to place B (incoming) elements into $M - B$ (already in memory) elements. Assume that a linear scan is done at first, which ends up sorting all the blocks. Finding the number of permutations now required, taking logarithm of that, and dividing by $\log \binom{M}{B}$ gives the desired lower bound:

$$\Omega\left(\frac{n}{B}\log_{M/B}\frac{n}{B} - \sum_{i=1}^{k}\frac{n_i}{B}\log_{M/B}\frac{n_i}{B} + \frac{n}{B}\right)$$

where the last term is a scan-everything-lower-bound.

4 Batched predecessor problem

Both X and Y are sorted and distinct. The number of ways to place X in Y is clearly at least $\binom{y}{x} \geq (y/x)^x$. A comparison between an element in Y and an element in X can reduce this number by a factor of at most 2. Thus

we get the desired lower bound. Again, the $+x$ at the end is a lower bound, since we must scan all elements of X.

Collect every (y/x)th element of Y to from another array Y' of size x. Now merge Y' and X. This takes $O(x)$ time, and ends up telling us for every $x \in X$ the "chunk" of size y/x it belongs to in Y. Now do binary search on this chunk, which takes $\log(y/x)$ comparisons, and we have the desired upper bound.

In the external memory model we will argue the lower bound using an adversary argument. The main problem is that as long as the search spaces of $x \in X$ intersect, an element of Y can "help" search for multiple elements in X. Our adversary strategy is as follows.

- The adversary first arbitrarily separates the search spaces of elements of X. That is, to every element in X it assigns a chunk of size y/x in Y, and tells the algorithm that x is in this chunk. Basically it gives the first x bits for free. This extra information can only help the algorithm. After giving this information, the number of bits needed to achive is $x \log(y/x)$. (Initially it was $x \log y$, and the adversary gave $x \log x$ for free.)

- Now we can assume that the algorithm keeps all the elements in X in internal memory for free (it can only do so if $X < M$, but we are arguing a lower bound, so this does not hurt us).

- Now consider an incoming block of B elements from from Y. These B elements can help at most B elements from X, as an element in Y cannot help more than one element in X(because it can help only if it comes from the current search space, and the search spaces for different xs are disjoint). It is easy to see that if these B elements help $j < B$ elements from X, the maximum number of bits they can achieve is $j \log(B/j + 1)$. This is maximized at $j = B$, so the maximum number of bits achieved during an I/O is B (B elements half their search space when this block has the median pivot from their current search spaces).

- Dividing the number of bits needed to achieve by thenumber of bits achievable in one I/O gives us the desired lower bound.