CHAPTER THREE

From Adaptive Analysis to Instance Optimality

Jérémy Barbay

Abstract: This chapter introduces the related concepts of *adaptive analysis* and *instance optimality*. The goal is to define an extremely fine-grained parameterization of a problem instance space in order to argue that a particular algorithm for the problem is "optimal" in a very strong sense. This chapter presents two detailed case studies, for the MAXIMA SET problem and a database aggregation problem, as well as a representative list of additional techniques, results, and open problems.

3.1 Case Study 1: Maxima Sets

Suppose you have a new job and need to find a house to buy or rent. You would like to find a place close to work but not too expensive. You gather a list of possible houses, but there are too many to visit them all. Can you reduce the list of places to visit without compromising any of your criteria?

This is a two-dimensional version of a well-known problem, reinvented multiple times in various areas; we will call it the MAXIMA SET problem. In a computational geometry context, it was first considered by Kung et al. (1975). The input is a set S of n points in the plane. A point of S is called *maximal* if none of the other points in S dominates it in every coordinate. The goal in the MAXIMA SET problem is to identify all of the maximal points (i.e., the maxima set).[1] See also Figure 3.1.[2]

Several algorithms have been proposed for the MAXIMA SET problem in two dimensions.[3] These algorithms highlight the importance of analyzing an algorithm's running time as a function of *two* parameters, the usual input size n (i.e., the number of input points) and also the output size k (i.e., the number of maximal points). We next briefly review several of these algorithms, which are important precursors to the more general notions of adaptive analysis and instance optimality. This sequence of increasingly instance-adaptive running time bounds will illustrate a process of iterative refinement, culminating in a form of instance optimality.

[1] For the house-finding problem, the x- and y-axes correspond to the negative price and negative distance of a house (as in the house-finding problem, smaller prices and distances are better).

[2] The same problem is explored in Chapter 12 in the context of self-improving algorithms.

[3] These algorithms can also be used to compute the convex hull of a set of points, and indeed were originally proposed primarily for this purpose.

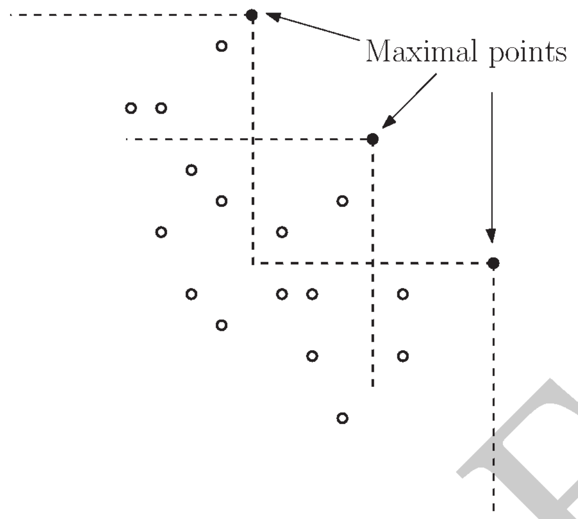

Figure 3.1 A point set and its maxima. Solid circles are the maximal points, hollow circles are the dominated points. The dashed lines indicate the “region of domination” for each of the maximal points.

3.1.1 Jarvis March (a.k.a. Gift Wrapping)

Among the n houses to consider, the cheapest and the closest are especially good candidates and can be identified with $O(n)$ comparisons. If there is only one such house (at once cheaper and closer than any other), the problem is solved. Otherwise, the cheapest and the closest must both be part of the output (they correspond to maximal points, which we will call *candidate* houses), and one can then iterate on the remaining $n - 2$ houses.

On account of a similar algorithm for the Convex Hull problem (Jarvis, 1973), we will call this algorithm Jarvis march. The number of house comparisons performed by the algorithm is $\Theta(nh)$ in the worst case for instances with n houses and h candidate houses selected in the end. This running time ranges from $\Theta(n)$ to $\Theta(n^2)$, depending on the output size h.

3.1.2 Graham’s Scan

Another approach, which improves over the quadratic worst-case running time of Jarvis march, is to first sort the n houses by increasing price using $O(n \log n)$ comparisons and then scan the list of houses in order to eliminate all the houses that are not maxima. Scanning the sorted list requires at most $2n = O(n)$ further home comparisons: the first house of this list is the cheapest and necessarily a candidate, and any house considered after that is either a candidate (if it is closer to work than the previous most expensive candidate considered) or can be pruned (if it is at once more expensive and farther to work than the previously most expensive house considered).

As before, by analogy with a similar algorithm for the Convex Hull problem (Graham, 1972), we’ll call this algorithm Graham’s scan.[4] This algorithm performs

[4] Called Sweeping Line in Chapter 12.

$O(n \log n)$ house comparisons. A reduction from SORTING can be used to show that, in the comparison model of computation, no algorithm uses asymptotically fewer comparisons in the worst case over instances with n houses.

Jarvis march is superior to Graham's scan when $h = o(\log n)$, equivalent to it when $h = \Theta(\log n)$, and inferior to it otherwise. How could we know which algorithm to use, given that the number h of candidate houses is what we wanted to compute in the first place? One idea is to execute both algorithms in parallel and stop as soon as one of them finishes.[5] This would yield a solution that runs in $O(n \cdot \min\{h, \log n\})$ time, but potentially with many comparisons performed twice. Is there a more parsimonious solution?

3.1.3 Marriage Before Conquest

Rather than choosing between `Jarvis march` and `Graham's scan`, Kirkpatrick and Seidel (1985) described a clever solution that performs $O(n \log h)$ house comparisons in the worst case over instances with n input points and h maximal points. They called the analogous algorithm for the CONVEX HULL problem Marriage Before Conquest (Kirkpatrick and Seidel, 1986), and we adopt that name here.[6]

```
Algorithm Marriage Before Conquest($S$):

1. If $|S| = 1$ then return $S$.
2. Divide $S$ into the left and right halves $S_\ell$ and $S_r$ by the median $x$-coordinate.
3. Discover the point $q$ with the maximum $y$-coordinate in $S_r$.
4. Prune all points in $S_\ell$ and $S_r$ that are dominated by $q$.
5. Return the concatenation of Marriage Before Conquest($S_\ell$) and Marriage Before
   Conquest($S_r$).
```

This divide-and-conquer algorithm uses as a subroutine the linear-time median finding algorithm of Blum et al. (1973). After identifying the median house price, one can partition the set of n houses into the $\lfloor n/2 \rfloor$ cheapest houses (corresponding to S_r, as cheaper is better) and the $\lceil n/2 \rceil$ most expensive houses (corresponding to S_ℓ). Given such a partition, one can find a first candidate house by selecting the house closest to work among the $\lfloor n/2 \rfloor$ least expensive houses, prune the houses dominated by this candidate, and recurse on both the set of remaining cheaper houses and the set of remaining more expensive houses.

Theorem 3.1 (Kirkpatrick and Seidel, 1985) *Given a set S of n points in the plane, the algorithm Marriage Before Conquest computes the maximal points of S in $O(n \log h)$ time, where h is the the number of maximal points.*

Proof Sketch The number of comparisons performed by the algorithm is $O(n \log h)$, as in the worst case the median divides the $h - 1$ maxima left to identify into two sets of roughly equal sizes. (In the best case, the median divides the instance into one instance with half the input points and one maximal point,

[5] Kirkpatrick (2009) describes this as a "Dovetailing" solution.

[6] The algorithm described here is a slight variant of that in Kirkpatrick and Seidel (1985); the original pruned only points from S_ℓ in line 4.

which is then solved recursively in linear time, and one instance with $h-2$ maxima but only $n/2$ input points.) □

A reduction from the SORTING MULTISETS problem on an alphabet of size h shows that no algorithm in the comparison model has a running time asymptotically better than $O(n \log h)$, in the worst case over inputs with input size n and output size h.

Marriage Before Conquest is so good that its inventors titled their paper on an extension to the CONVEX HULL problem "The Ultimate Planar Convex Hull Algorithm?" (Kirkpatrick and Seidel, 1986). To answer this question (for MAXIMA SET or CONVEX HULL), one would need to prove that no algorithm can outperform Marriage Before Conquest by more than a constant factor. Haven't we already proved this? What type of optimality could one hope for beyond optimality over instances of a given input size and output size?

3.1.4 Vertical Entropy

It turns out there are natural finer-grained parameterizations of the input space for the MAXIMA SET (and CONVEX HULL) problem, which enable stronger notions of algorithm optimality. For example, consider an instance with output size $h = o(n)$, where one of the h houses in the output is cheaper and closer than the $n-h$ dominated houses. Such an instance is much easier for the Marriage Before Conquest algorithm than, say, one where each of the h candidate houses dominates exactly $(n-h)/h$ noncandidate houses. In the latter case, the algorithm might well run in $\Theta(n \log h)$ time. But in the former case, it performs only $O(n + h \log h) = o(n \log h)$ comparisons: as $h = o(n)$, the median-priced house will be among the $n-h$ dominated houses, leading to the selection of the particular one cheaper and closer than the $n-h$ noncandidate houses and the latter's immediate elimination, leaving only $h-1$ candidate houses to process in $O(h \log h)$ time.

To better measure the difference in difficulty between such instances, Sen and Gupta (1999) defined n_i as the number of noncandidate houses dominated by the ith cheapest candidate house (and no candidate house cheaper than this one), and the *vertical entropy* of an instance as the entropy of the distribution of $\{n_i\}_{i \in [2,\ldots,h]}$. Formally:

$$\mathcal{H}_v(n_2, \ldots, n_h) = \sum_{i=2}^{h} \frac{n_i}{n} \log\left(\frac{n}{n_i}\right). \tag{3.1}$$

Note that, by basic properties of entropy, $\mathcal{H}_v \leq \log_2 h$.

Vertical entropy yields a more fine-grained parameterization and the following result:

Theorem 3.2 (Sen and Gupta, 1999) *Given a set S of n points in the plane, the Marriage Before Conquest algorithm computes the maxima set of S in $O(n\mathcal{H}_v)$ time, where $\mathcal{H}_v$ is the vertical entropy* (3.1) *of S.*

Proof Sketch We claim that the number of comparisons used by the Marriage Before Conquest algorithm is $O(n\log n - \sum_{i=2}^{h} n_i \log n_i) = O(\sum_{i=2}^{h} n_i \log(n/n_i)) = O(n\mathcal{H}_v(n_2, \ldots, n_i))$. The essential idea is that, if a maximal point dominates at least n_i input points, then it will be identified by Marriage Before Conquest after at most $\log_2(n/n_i)$ rounds. (For example, if $n_i \geq n/2$, it will be identified immediately; if $n_i \geq n/4$ it will be identified after at most two levels of recursion; and so on.) Thus, the n_i input points that it dominates contribute at most n_i comparisons to at most $\log_2(n/n_i)$ partition phases. □

A reduction from the SORTING MULTISETS problem on an alphabet of size $h-1$ and frequency distributions $\{n_i\}_{i\in[2,\ldots,h]}$ shows that no algorithm in the comparison model has a running time asymptotically better than $O(n\mathcal{H}_v)$, in the worst case over inputs with input size n and vertical entropy $\mathcal{H}_v$.

Because $\mathcal{H}_v \leq \log_2 h$, the analysis of Sen and Gupta (1999) is more fine-grained that that of Kirkpatrick and Seidel (1985). This shows that Marriage Before Conquest is even more “adaptive” than its authors gave the algorithm credit for. But does Theorem 3.2 prove that it truly is the “ultimate” algorithm for the problem?

3.1.5 (Order-Oblivious) Instance Optimality

Theorem 3.2 is insufficient to claim the “ultimateness” of the Marriage Before Conquest algorithm: one could define in a very similar way the “horizontal entropy” of an instance. There are instances with high vertical entropy and low horizontal entropy, and vice versa. One could also define a “horizontal” version of the Marriage Before Conquest algorithm, which would iteratively partition houses around the one of median distance rather than cost, which would then be optimal with respect to the horizontal entropy parameter. This section outlines a result of Afshani et al. (2017), who showed that the Marriage Before Conquest algorithm is indeed “ultimate,” among algorithms in the comparison model that do not take advantage of the order of the input.

Central to the notion of *instance optimality* is the idea of a *certificate* of an instance.[7] Any correct algorithm for a problem implicitly certifies the correctness of its output, and the description length of this certificate is a lower bound on the algorithm’s running time. In instance optimality, the goal is to define a form of certificate such that, for every instance (1) every correct algorithm implicitly defines such a certificate and (2) the protagonist algorithm (to be proved instance-optimal) runs in time at most a constant factor times the length of the shortest certificate.

In the specific case of the MAXIMA SET problem, any correct algorithm must be able to justify: (1) for each of the $n-h$ noncandidate houses, why it was discarded; and (2) for each of the h candidate houses, why it cannot be discarded. The algorithms presented in this section (Jarvis march, `Graham's scan`, and Marriage Before Conquest) justify their choices in the same way: (1) each noncandidate house is discarded only after the algorithm has found another house that dominates it; and (2) each candidate house is added to the output only after it has been determined that there is no cheaper house which is closer, and no closer house that is cheaper.

[7] A similar notion is used in Chapter 12 in the context of self-improving algorithms.

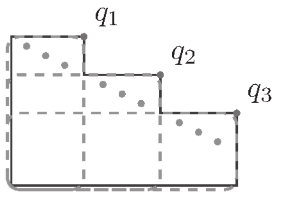

Figure 3.2 A "harder" point set for the computation of the maxima set in two dimensions.

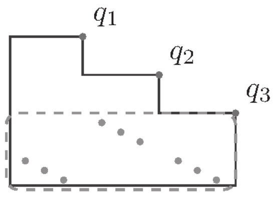

Figure 3.3 An "easier" point set.

The next definition formalizes this idea. By the *staircase* of a point set, we mean the boundary of the union of the "regions of domination" of the maximal points (cf., Figure 3.1).

Definition 3.3 Consider a partition Π of the set S of n input points into disjoint subsets $S_1, \ldots, S_t$. The partition Π is *respectful* if each subset S_k is either a singleton or can be enclosed by an axis-aligned box B_k whose interior is completely below the staircase of S. Define the *entropy* $\mathcal{H}(\Pi)$ of the partition Π to be $\sum_{k=1}^{t}(|S_k|/n)\log(n/|S_k|)$. Define the *structural entropy* $\mathcal{H}(S)$ of the input set S to be the minimum of $\mathcal{H}(\Pi)$ over all respectful partitions Π of S.

The intuition is that each nonsingleton group S_i represents a cluster of points that could conceivably be eliminated by an algorithm in one fell swoop.[8] Thus the bigger the S_i's, the easier one might expect the instance to be (Figures 3.2–3.3).

The structural entropy is always at most the vertical entropy (and similarly the horizontal entropy), as shown by taking the S_i's to be "vertical slabs" as in Figure 3.4 (with each maximal point in its own set).

The following result shows that, for every instance, the running time of the Marriage Before Conquest algorithm is bounded by the number of points times the structural entropy of the instance.

Theorem 3.4 (Afshani et al., 2017) *Given a set S of n points in the plane, the algorithm Marriage Before Conquest computes the maxima set of S in $O(n(\mathcal{H}(S)+1))$ time.*

Proof Consider the recursion tree of the algorithm (Figure 3.5) and let X_j denote the sub-list of all maximal points of S discovered during the first j recursion levels, in left-to-right order. Let $S^{(j)}$ be the subset of points of S that *survive* recursion level j, i.e., that have not been pruned during levels $0, \ldots, j$ of the recursion, and let $n_j = |S^{(j)}|$. The algorithm performs $O(n_j)$ operations to refine level j into level $j+1$, and there are at most $\lceil \log n \rceil$ such levels in the computation, so the total running time is $O(\sum_{j=0}^{\lceil \log n \rceil} n_j)$. Next observe that:

[8] Beginning from the northeast corner of the box, travel north until you hit the staircase, and then east until you hit a maximal point q. The point q dominates all of the points in S_i.

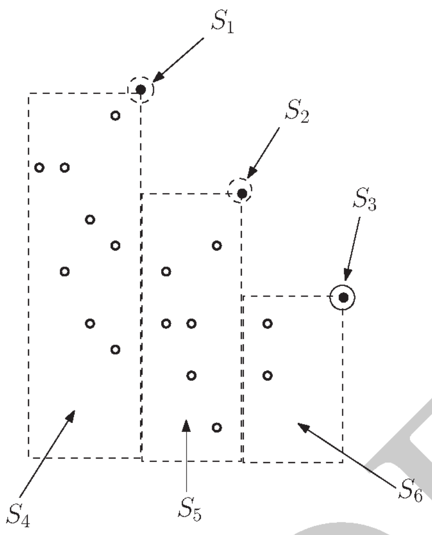

Figure 3.4 A respectful partition using vertical slabs: structural entropy generalizes vertical entropy.

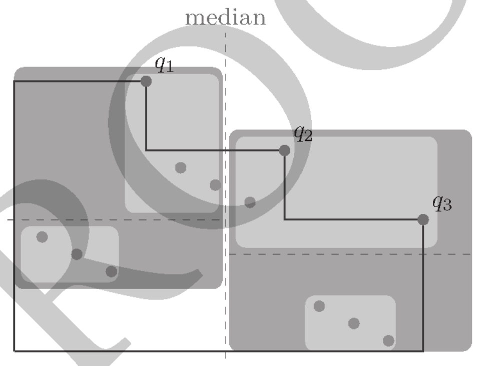

Figure 3.5 The beginning of the recursive partitioning of S. The two bottom boxes are already leaves of the recursion tree, while the two top boxes will be divided further.

(i) there can be at most $\lceil n/2^j \rceil$ points of $S^{(j)}$ with x-coordinates between any two consecutive points in X_j; and
(ii) all points of S that are strictly below the staircase of X_j have been pruned during levels $0, \ldots, j$ of the recursion.

Let Π be a respectful partition of S. Consider a nonsingleton subset S_k in Π. Let B_k be a box enclosing S_k whose interior lies below the staircase of S. Fix a level j. Suppose that the upper-right corner of B_k has x-coordinate between two consecutive points q_i and q_{i+1} in X_j. By (ii), the only points in B_k that can survive level j must have x-coordinates between q_i and q_{i+1}. Thus, by (i), the number of points in S_k that survive level j is at most $\min\{|S_k|, \lceil n/2^j \rceil\}$. (Note that the bound is trivially true if S_k is a singleton.) Because the S_k's cover the entire point set, with a double summation we have

$$\begin{aligned}
\sum_{j=0}^{\lceil \log n \rceil} n_j &\le \sum_{j=0}^{\lceil \log n \rceil} \sum_k \min\left\{|S_k|, \lceil n/2^j \rceil\right\} \\
&= \sum_k \sum_{j=0}^{\lceil \log n \rceil} \min\left\{|S_k|, \lceil n/2^j \rceil\right\} \\
&\le \sum_k (|S_k| \lceil \log(n/|S_k|) \rceil + |S_k| + |S_k|/2 + |S_k|/4 + \cdots + 1) \\
&\le \sum_k |S_k| (\lceil \log(n/|S_k|) \rceil + 2) \\
&\in O(n(\mathcal{H}(\Pi) + 1)).
\end{aligned}$$

This bound applies for every respectful partition of S, so it also applies with $\mathcal{H}(\Pi)$ replaced by the structural entropy $\mathcal{H}(S)$ of S. This completes the proof. □

Moreover, a nontrivial adversary argument can be used to prove a matching lower bound for all *order-oblivious* algorithms (Afshani et al., 2017). Formally, the statement is: For every correct algorithm A for the MAXIMA SET problem and every set S of n points, there exists an ordering of the points in S such that A uses $\Omega(n(\mathcal{H}(S) + 1))$ comparisons to solve the corresponding instance of MAXIMA SET. Thus, any running time bound that does not reference the ordering of the points in the input (like all of the standard running time bounds for algorithms for the MAXIMA SET problem) must be $\Omega(n(\mathcal{H}(S) + 1))$.

Theorem 3.4 and the matching lower bound prove a strong form of "ultimateness" for the Marriage Before Conquest algorithm. But could we do even better by somehow taking advantage of the input order?

3.1.6 Partially Sorted Inputs

Remember the Graham's scan algorithm (Section 3.1.2), which first sorted the houses by price and then scanned the sorted list in linear time to discard noncandidate houses? This algorithm shows that the MAXIMA SET problem can be solved in *linear* time for instances in which the input is already sorted. An analogous observation holds for inputs that are partially sorted, meaning that the (ordered) input can be partitioned into a small number of sorted fragments. Here, the maxima set of each fragment can be computed in time linear in the fragment length, and the merging of all the maxima sets can in some cases be done quickly enough to obtain a better running time bound than an order-oblivious algorithm.[9]

Theorem 3.5 (Ochoa, 2019) *Consider a sequence S of n points in the plane comprising sorted fragments of lengths $r_0, \ldots, r_\rho$, with structural entropy $\mathcal{H}(S)$.*

[9] For further examples of algorithms that adapt to partially sorted inputs, see Exercise 3.6 and Section 3.3.1.

There is an algorithm which computes the maxima set of S in $O(n+\min\{n\log n-\sum_{i=0}^{\rho} r_i \log r_i, n\mathcal{H})\})$ *time.*

Perhaps techniques along the lines of Theorem 3.5 can lead to a truly instance-optimal algorithm for the MAXIMA SET problem, without any restriction to order-obliviousness?

3.1.7 Impossibility Result

In fact, the very existence of the Graham's scan algorithm implies that *no* algorithm for the MAXIMA SET problem can be truly instance optimal in the comparison model:

Theorem 3.6 (Afshani et al., 2017) *There is no instance-optimal algorithm for the MAXIMA SET problem.*

In fact, for every algorithm A for the MAXIMA SET problem, there is another algorithm B for the problem and an infinite family of inputs z such that A runs in $\Omega(n\log n)$ time on these inputs while B runs in $O(n)$ time.

Proof Sketch The intuition of the proof is very simple: For any given instance I, there is at least one competing algorithm that correctly guesses an order in which to process the input so that the Graham's scan algorithm computes the maxima set of I in linear time. Furthermore, no algorithm in the comparison model can solve *all* instances of size n in $o(n\log n)$ time (by a simple counting argument). Hence, no algorithm can compute the maxima set for every instance I in time bounded by a constant factor times that of the best algorithm for I. □

Theorem 3.6 shows that the "order-oblivious" qualifier (or some other restriction) is necessary for an instance optimality result for the MAXIMA SET problem. Thus, if one had to choose an "ultimate" algorithm for the problem, the Marriage Before Conquest algorithm is the best candidate around: it (or the minor variant described here) is instance optimal among order-oblivious algorithms, which would seem to be the next best thing to a truly instance-optimal algorithm (which does not exist).

3.2 Case Study 2: Instance-Optimal Aggregation Algorithms

This section considers a second case study of an instance-optimal algorithm, for the database aggregation problem for which the concept was originally defined.

3.2.1 Instance Optimality

We begin by zooming out to discuss instance optimality in general. Some measures of difficulty are finer than others. Could there be a "finest-possible" measure, so that an algorithm that is optimal with respect to that measure is automatically optimal also with respect to every other (coarser) measure? This may seem like a pipe dream for any natural computational problem, but Fagin et al. (2003) described such a result for a database aggregation problem.[10] Algorithms that are optimal with respect to such a finest-possible measure – *instance-optimal* algorithms – can be viewed as the ultimate adaptive algorithms, always as good (up to a constant factor) as every other algorithm on every input.

Consider a computational problem and cost measure, with $\text{cost}(A, z)$ denoting the cost incurred (e.g., number of operations) by the algorithm A on the input z.

Definition 3.7 (Instance Optimality) An algorithm A for a problem is *instance optimal with approximation c with respect to the set $\mathcal{C}$ of algorithms* if for every algorithm $B \in \mathcal{C}$ and every input z,

$$\text{cost}(A, z) \leq c \cdot \text{cost}(B, z),$$

where $c \geq 1$ is a constant, independent of B and z.

The constant c in Definition 3.7 is called the *optimality ratio* of A (with respect to $\mathcal{C}$). By an *instance-optimal algorithm*, one generally means an algorithm with optimality ratio bounded by a constant.[11] This is a demanding definition, and for many problems there is no instance-optimal algorithm with respect to any reasonably rich class of algorithms $\mathcal{C}$. In Theorem 3.4, we saw an example of an instance-optimal algorithm for the Maxima Set problem with respect to the class of order-oblivious algorithms, and from Theorem 3.6 we learned that there is no instance-optimal algorithm for the problem with respect to the class of all comparison-based algorithms.

The rest of this section covers the original success story for instance-optimal algorithms.

3.2.2 The Setup

The problem is as follows. There is a very large set X of objects, such as Web pages. There is a small number m of attributes, such as the ranking (e.g., PageRank) of a Web page under m different search engines. To keep things simple, assume that attribute values lie in $[0, 1]$. Thus an object consists of a unique name and an element of $[0, 1]^m$.

We are also given a *scoring function* $\sigma : [0, 1]^m \to [0, 1]$ which aggregates m attribute values into a single score. We interpret higher attribute values and scores as being "better." We assume that the scoring function is *monotone*, meaning that its output is nondecreasing in each of its inputs. An obvious scoring function is the average, but clearly there are numerous other natural examples.

The algorithmic goal is, given a positive integer k, to identify k objects of X that have the highest scores (ties can be broken arbitrarily).

We assume that the data can be accessed only in a restricted way. It is presented as m sorted lists $L_1, L_2, \ldots, L_m$. Each list L_i is a copy of X, sorted in nonincreasing order

[10] This paper by Fagin et al. (2003) was the winner of the 2014 EATCS-SIGACT Gödel prize, a "test of time" award for papers in theoretical computer science.

[11] One drawback of this coarse definition of an instance-optimal algorithm is its Manichean nature – it does not differentiate between competing instance-optimal algorithms whose optimality ratios differ by large (constant) factors, nor does it differentiate between different problems that, even though they do not admit instance-optimal algorithms, might nevertheless differ in difficulty.

of the ith attribute value. An algorithm can access the data only by requesting the next object in one of the lists. Thus an algorithm could ask for the first (highest) object of L_4, followed by the first object of L_7, followed by the second object of L_4, and so on. Such a request reveals the name of said object along with all m of its attribute values. We charge an algorithm a cost of 1 for each such data access.[12] Thus, in the notation of Definition 3.7, we are defining the cost measure cost(A, z) as the number of data accesses that the algorithm A needs to correctly identify the top k objects in the input z.

3.2.3 The Threshold Algorithm

We study the following *threshold algorithm* (TA). The algorithm is natural but perhaps not the first algorithm one would write down for the problem. The reader is encouraged to think about "more obvious" algorithms, which will probably not be instance-optimal.

Algorithm 1 The threshold algorithm (TA)

Input: a parameter k and m sorted lists.
Invariant: of the objects seen so far, S is those with the top k scores.

```
1. Fetch the next item from each of the $m$ lists.
2. Compute the score $\sigma(x)$ of each object $x$ returned, and update $S$ as needed.
3. Let $a_i$ denote the $i$th attribute value of the object just fetched from the list $L_i$, and
   set a threshold $t := \sigma(a_1, \ldots, a_m)$.
4. If all objects of $S$ have score at least $t$, halt; otherwise return to step 1.
```

We first claim that the TA is correct – for every input, it successfully identifies the k objects with the highest scores (even if it halts well before encountering all of the objects of X).

Proof By definition, the final set S returned by the TA is the best of the objects seen by the algorithm. If an object $x \in X$ has not been seen by the TA, then its ith attribute value x_i is at most the lowest attribute value a_i of an object fetched from the list L_i (since the lists are sorted). Since σ is a monotone scoring function, $\sigma(x)$ is at most $\sigma(a_1, \ldots, a_m)$, which by definition is the final threshold t of the TA, which by the stopping rule is at most the score of every object in S. Thus every object in S has score at least as large as every object outside of S, as desired. □

[12] This is not the most realistic cost model, but it serves to illustrate our main points in a simple way. In the terminology of Fagin et al. (2003), this corresponds to a sequential access cost of 1 and a random access cost of 0. More generally, Fagin et al. (2003) charge some constant c_s for each data access of the type we describe and assume that accessing list L_i only reveals the value of the ith attribute; the other attribute values are then determined via $m - 1$ "random accesses" to the other lists, each of which is assumed to cost some other constant c_r. Analogous instance optimality results are possible in this more general model (Fagin et al., 2003).

The main takeaway point of the proof is: the threshold t acts as an upper bound on the best possible score of an unseen object. Once the best objects identified so far are at least this threshold, it is safe to halt without further exploration.

3.2.4 Instance Optimality of the Threshold Algorithm

The threshold algorithm is in fact *instance-optimal with optimality ratio m*.

Theorem 3.8 (Instance optimality of the TA) *For* every *algorithm A and* every *input z,*

$$cost(TA, z) \leq m \cdot cost(A, z). \tag{3.2}$$

In words, suppose you precommit to using the TA, and you have a competitor who is allowed to pick *both* an input z *and* a (correct) algorithm A that is specifically tailored to perform well on the input z. Theorem 3.8 says that even with this extreme advantage, your opponent's performance will be only a factor of m better than yours. Recall that we view m as a small constant, which makes sense in many natural motivating applications for the problem. We will see in the text that follows that no algorithm has an optimality ratio smaller than m.

Proof (of Theorem 3.8) Consider a (correct) algorithm A and an input z. Suppose that A accesses the first $k_1, \ldots, k_m$ elements of the lists $L_1, \ldots, L_m$ en route to computing the (correct) output S on z. For each i, let b_i denote the ith attribute value of the last accessed object of L_i – the lowest such attribute value seen for an object fetched from L_i.

The key claim is that, on accord of A's correctness, every object x in A's output S must have a score $\sigma(x)$ that is at least $\sigma(b_1, \ldots, b_m)$. The reason is: For all A knows, there is an unseen object y with attribute values $b_1, \ldots, b_m$ lurking as the $(k_i + 1)$th object of list L_i for each i (recall that ties within an L_i can be broken arbitrarily). Thus, A cannot halt with $x \in S$ and $\sigma(x) < \sigma(b_1, \ldots, b_m)$ without violating correctness on some input z'. (Here z' agrees with z on the first k_i objects of each L_i, and has an object y as above next in each of the lists.)

Now, after $\max_i k_i$ rounds, the TA has probed at least as far as A into each of lists, and has discovered every object that A did (including all of S). Thus a_i, the ith attribute value of the final item fetched by the TA from the list L_i, is at most b_i. Since σ is monotone, $\sigma(a_1, \ldots, a_m) \leq \sigma(b_1, \ldots, b_m)$. Thus after at most $\max_i k_i$ rounds, the TA discovers at least k objects with a score at least its threshold, which triggers its stopping condition. Thus $\text{cost}(TA, z) \leq m \cdot \max_i k_i$; since $\text{cost}(A, z) = \sum_i k_i \geq \max_i k_i$, the proof is complete. □

3.2.5 A Matching Lower Bound on the Optimality Ratio

The factor of m in Theorem 3.8 cannot be improved, for the TA or any other algorithm. We content ourselves with the case of $k = 1$ and a scoring function σ with the property that $\sigma(x) = 1$ if and only if $x_1 = x_2 = \cdots = x_m = 1$. More general lower bounds are possible (Fagin et al., 2003), using extensions of the simple idea explained here.

The guarantee of instance optimality is so strong that proving lower bounds can be quite easy. Given an arbitrary correct algorithm A, one needs to exhibit an input z and a correct algorithm A' with smaller cost on z than A. Getting to choose A' and z in tandem is what enables simple lower bound proofs.

Suppose $k = 1$. We will use only special inputs z of the following form:

- there is a unique object y with $\sigma(y) = 1$; and
- this object y appears first in exactly 1 of the lists $L_1, \dots, L_m$. (Recall that arbitrary tie-breaking within a list is allowed.)

The lower bound follows from the following two observations. For every such input z, there is an algorithm A' with $\text{cost}(A', z) = 1$: It looks in the list containing y first, and on finding it can safely halt with y as the answer, since no other object can have a higher score. But for every fixed algorithm A, there is such an input z on which $\text{cost}(A, z) \geq m$: A must look at one of the lists last, and an adversary can choose the input z in which y is hidden in this last list.

The fact that lower bounds for instance optimality arise so trivially should give further appreciation for instance-optimal algorithms with small optimality ratios (when they exist).

3.3 Survey of Additional Results and Techniques

Many techniques have been introduced to refine worst-case analysis through parameterizations of input difficulty beyond input size. There are too many such results to list here, so we present only a selection that illustrates some key notions and techniques.

3.3.1 Input Order

We distinguish between algorithms adaptive to the *ordering* of the input versus those to the (unordered) *structure* of the input. An example of the former is the algorithm in Theorem 3.5, which adapts to partially sorted instances of the MAXIMA SET problem. For further results along these lines for adaptive sorting, see the survey of Estivill-Castro and Wood (1992) and the overview of Moffat and Petersson (1992).

3.3.2 Input Structure

An early example of adapting to (unordered) input structure is due to Munro and Spira (1976), who showed how algorithms could adapt to the frequencies of the elements in a multiset M in order to sort them with fewer comparisons than would be required in the worst case. We discuss this and a few additional examples in the text that follows.

Output size: In Section 3.1.3 we saw the concept of an *output-sensitive* algorithm, one of the most basic notions of adaptivity to input structure. Kirkpatrick and Seidel (1985) gave the first output-sensitive algorithm for computing the maximal points of a point set in d dimensions. Their algorithm's running time is $O(n(1+\log h))$ in 2 and 3 dimensions, and $O(n(1+\log^{d-2} h))$ for dimensions $d > 3$, where h is the number of maximal points. The next year Kirkpatrick and Seidel (1986) proved similar results for the CONVEX HULL problem, if only in two and three dimensions.

Such results were later refined for the CONVEX HULL problem by Sen and Gupta (1999) to adapt to the *vertical entropy* [cf. (3.1)] and by Afshani et al. (2017) to adapt to the structural entropy of a point set (cf., Definition 3.3). See Section 3.5.1 for recent progress and open questions for point sets in four or more dimensions.

Repetitions. For another example of (unordered) input structure, consider a multiset M of size n (e.g., $M = \{4,4,3,3,4,5,6,7,1,2\}$ of size $n = 10$). The *multiplicity* of an element x of M is the number m_x of occurrences of x in M (e.g., $m_3 = 2$). The distribution of the multiplicities of the elements in M is the set of pairs (x, m_x) (e.g., $\{(1,1),(2,1),(3,2),(4,3),(5,1),(6,1),(7,1)\}$ in M). Munro and Spira (1976) described a variant of the MergeSort algorithm that uses counters, and which takes advantage of the distribution of the multiplicities of the elements in M when sorting it. This algorithm runs in $O(n(1+\mathcal{H}(m_1,\ldots,m_\sigma)))$ time, where σ is the number of distinct elements in M, $m_1,\ldots,m_\sigma$ are the multiplicities of the σ distinct elements, and $\mathcal{H}$ is the entropy of the corresponding distribution. They proved that this running time is the best possible in the decision tree model (up to constant factors), in the worst case over instances of size n with σ distinct elements of multiplicities $m_1,\ldots,m_\sigma$.

Miscellaneous Input Structure. Barbay et al. (2017a) proposed adaptive algorithms for three related problems where the input is a set $\mathcal{B}$ of axis-aligned boxes in d dimensions: the KLEE'S MEASURE problem (i.e., computing the volume occupied by the union of the boxes of $\mathcal{B}$); the MAXIMAL DEPTH problem (i.e., computing the maximal number of boxes of $\mathcal{B}$ that cover a common point of space); and the DEPTH DISTRIBUTION problem (i.e., for each i compute the volume of the points that are covered by exactly i boxes from $\mathcal{B}$).

3.3.3 Synergy between Order and Structure

Are there algorithms that profitably take advantage of both input order and input structure? This is the question considered by Barbay et al. (2017b). They showed that, for the problem of sorting a multiset, there is an algorithm that adapts simultaneously to partially sorted inputs (as in Section 3.1.6) and also the entropy of the distribution of elements' frequencies; for some instances, this results in a running time that is asymptotically faster than what can be achieved when taking advantage of only of one of the two aspects. This article also considers data structures for answering rank and select queries, while taking advantage of the *query structure* and *query order* (in addition to the input order and input structure). Finally, Barbay and Ochoa (2018) show analogous results for the MAXIMA SET and CONVEX HULL problems (in two dimensions).

3.4 Discussion

This section compares and contrasts adaptive analysis and instance optimality with parameterized algorithms and the competitive analysis of online algorithms.

3.4.1 Comparison with Parameterized Algorithms

Both adaptive analysis and parameterized algorithms (as described in Chapter 2) analyze the running time of algorithms using parameters above and beyond the input size. One major difference between the two areas is that the former focuses on polynomial-time solvable problems (often with near-linear worst-case complexity) while the latter is focused on NP-hard problems.[13] Lower bounds for NP-hard parameterized problems are necessarily conditional (at least on $P \neq NP$, and often on stronger assumptions such as the Strong Exponential Time Hypothesis). Meanwhile, tight unconditional lower bounds are a prerequisite for instance optimality results, and these are typically known only for near-linear-time solvable problems (such as SORTING or CONVEX HULL in two or three dimensions) and restricted models of computation (comparison-based algorithms or decision trees).

Adaptive analysis is relevant more generally to polynomial-time solvable problems for which we don't know good lower bounds. For example, Barbay and Pérez-Lantero (2018) and Barbay and Olivares (2018) analyzed adaptive algorithms for various string problems (in the spirit of EDIT DISTANCE) while Barbay (2018) presented similar results for the DISCRETE FRECHET DISTANCE problem.

3.4.2 Comparison with the Competitive Analysis of Online Algorithms

An *online algorithm* is one that receives its input one piece at a time and is required to make irrevocable decisions along the way. In the competitive analysis of online algorithms (initiated by Sleator and Tarjan (1985) and covered in Chapter 24 of this book), the goal is to identify online algorithms with a good (close to 1) competitive ratio, meaning that the objective function value of the algorithm's output is guaranteed to be almost as good as what could be achieved by an all-powerful and all-knowing offline optimal algorithm.

The competitive ratio provided inspiration for the optimality ratio (Definition 3.7) and instance optimality.[14] Indeed, we can interpret a guarantee of c on the competitive ratio of an online algorithm A as a guarantee on the optimality ratio of A (where the cost(A, z) is the objective function value of the output of A for the input z) with respect to the family $\mathcal{C}$ of *all* algorithms (and in particular, the offline optimal one).

3.5 Selected Open Problems

We close our chapter with two open research directions.

3.5.1 High Dimensions

One may have more than two criteria for choosing a house: Rather than just the price and the distance to work, certainly its size, whether it has a garden, the quality of

[13] One superficial distinction is that in parameterized algorithms the relevant parameter value is generally given as part of the input, while in adaptive analysis it shows up only in the analysis of an algorithm. But a typical fixed-parameter tractable algorithm can be extended to handle the case where the relevant parameter is not part of the input, merely by trying all possible values for that parameter.

[14] Fagin et al. (2003) write, "We refer to c as the optimality ratio. It is similar to the competitive ratio in competitive analysis."

the neighborhood, and so on should be taken into account for such an important decision.

The results of Kirkpatrick and Seidel (1985) on the MAXIMA SET problem also covered the case of high dimensions, with the higher-dimensional analog of Marriage Before Conquest computing the maximal set of a d-dimensional point set with $d \geq 3$ in $O(n \log^{d-2} h)$ time, where as usual n and h denote the size of the input and output, respectively. Afshani et al. (2017) refined this analysis not only in dimension two (as described in Section 3.1.5), but also in dimension three, with a rather different algorithm that partitions the input based on a carefully chosen sample of points. Barbay and Rojas-Ledesma (2017) proved analogous results in $d > 3$ dimensions:

Theorem 3.9 (Barbay and Rojas-Ledesma, 2017) *Consider a set S of n points in $\mathbb{R}^d$, and let Π be a respectful partition of S into subsets $S_1, \ldots, S_t$ of sizes $n_1, \ldots, n_t$, respectively. There is an algorithm that computes the maximal points of S in*

$$O\left(n + \sum_{k=1}^{t} n_k \log^{d-2} \frac{n}{n_k}\right) \tag{3.3}$$

time.

Could there be a matching lower bound, as is the case in two and three dimensions? The (open) problem is that there is no reason to believe that the expression in (3.3) is the minimal description length of a certificate of correctness; for example, for all we know there is a bound that depends linearly on d (rather than exponentially).

3.5.2 Layers of Maxima Sets

In the MAXIMA SET problem, every point is given a binary classification (maximal or not). More generally, one could identify the maximal set S_1 of the input S (the "first layer"), followed by the maximal set S_2 of the remaining point $S \setminus S_1$ (the "second layer"), and so on.

Nielsen (1996) described this problem and an output sensitive solution (similar to that described in Section 3.1.3 for the MAXIMA SET problem). Extending this result to obtain order-oblivious instance optimality is not overly difficult, but it remains an open problem to make the algorithm adaptive to various forms of input order.

3.6 Key Takeaways

In the following we summarize what we consider to be two of the main lessons from this incomplete survey of results on adaptive analysis and instance optimality.

1. Most of the techniques used in the adaptive analysis of algorithms and data structures resemble those used in classical worst-case analysis over instances of a given input size, the difference being that the ideas are applied in a more fine-grained context.

2. The concept of instance optimality (for various computational models) can be further refined to the concept of the optimality ratio among a class of algorithms over a class of instances. Such a refinement differentiates between more pairs of algorithms than the coarser criterion of instance optimality.

3.7 Notes

We conclude with some final bibliographic remarks, to supplement those given throughout the preceding sections.

1. McQueen and Toussaint (1985) originally introduced the minor variant of Marriage Before Conquest which was proved (order-oblivious) instance optimal in Afshani et al. (2017).
2. Petersson and Moffat (1995) introduced a notion of formal reductions between measures of difficulty, which induces a partial order on difficulty measures (Estivill-Castro and Wood, 1992; Moffat and Petersson, 1992). Such a theory of reductions is similar to the reductions between pairs of problems and parameters discussed in Chapter 2 on parameterized algorithms (as reductions between parameterized problems induce a partial order on them according to difficulty), but in a context where one can prove unconditional lower bounds.
3. Barbay and Navarro (2013) formalized the notion of *compressibility* measures for the analysis of the space used by compressed data structures, inspired by difficulty measures used in the running time analysis of algorithms.

Acknowledgments

Some of the examples, definitions, and results in this chapter were inspired by similar ones in the PhD theses of Carlos Ochoa and Javiel Rojas.

References

Afshani, Peyman, Barbay, Jérémy, and Chan, Timothy M. 2017. Instance-optimal geometric algorithms. *Journal of the ACM*, **64**(1), 3:1–3:38.

Barbay, Jérémy. 2018. Adaptive computation of the discrete Fréchet distance. In *Proceedings of the 11th Symposium on String Processing and Information Retrieval (SPIRE)*, pp. 50–60.

Barbay, Jérémy, and Navarro, Gonzalo. 2013. On compressing permutations and adaptive sorting. *Theoretical Computer Science*, **513**, 109–123.

Barbay, Jérémy, and Ochoa, Carlos. 2018. Synergistic computation of planar maxima and convex hull. In *Proceedings of the 23rd Annual International Computing and Combinatorics Conference (COCOON)*, pp. 156–167.

Barbay, Jérémy, and Olivares, Andrés. 2018. Indexed dynamic programming to boost edit distance and LCSS computation. In *Proceedings of the 11th Symposium on String Processing and Information Retrieval (SPIRE)*, pp. 61–73.

Barbay, Jérémy, and Pérez-Lantero, Pablo. 2018. Adaptive computation of the swap-insert correction distance. *ACM Transactions on Algorithms*, **14**(4), 49:1–49:16.

Barbay, Jérémy, and Rojas-Ledesma, Javiel. 2017. Multivariate analysis for computing maxima in high dimensions. *CoRR*, abs/1701.03693.

Barbay, Jérémy, Pérez-Lantero, Pablo, and Rojas-Ledesma, Javiel. 2017a. Depth distribution in high dimensions. In *Proceedings of the 23rd Annual International Computing and Combinatorics Conference (COCOON)*, pp. 38–40.

Barbay, Jérémy, Ochoa, Carlos, and Satti, Srinivasa Rao. 2017b. Synergistic solutions on multiSets. In *Proceedings of the 28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pp. 31:1–31:14.

Blum, Manuel, Floyd, Robert W., Pratt, Vaughan, Rivest, Ronald L., and Tarjan, Robert E. 1973. Time bounds for selection. *Journal of Computer and System Sciences*, **7**(4), 448–461.

Estivill-Castro, Vladimir, and Wood, Derick. 1992. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, **24**(4), 441–476.

Fagin, Ronald, Lotem, Amnon, and Naor, Moni. 2003. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, **66**(4), 614–656.

Graham, Ron L. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, **1**, 132–133.

Jarvis, Ray A. 1973. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, **2**(1), 18–21.

Kirkpatrick, David. 2009. Hyperbolic dovetailing. In *Proceedings of the 17th Annual European Symposium on Algorithms*, pp. 516–527. Springer Science+Business Media.

Kirkpatrick, David G., and Seidel, Raimund. 1985. Output-size sensitive algorithms for finding maximal vectors. In *Proceedings of the First International Symposium on Computational Geometry (SOCG)*, pp. 89–96. ACM.

Kirkpatrick, David G, and Seidel, Raimund. 1986. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, **15**(1), 287–299.

Kung, H T, Luccio, F, and Preparata, F P. 1975. On finding the maxima of a set of vectors. *Journal of the ACM*, **22**, 469–476.

Lucas, Édouard. 1883. *La Tour d'Hanoï, Véritable Casse-Tête Annamite*. In a puzzle game, Amiens. Jeu rapporté du Tonkin par le professeur N.Claus (De Siam).

McQueen, Mary M., and Toussaint, Godfried T. 1985. On the ultimate convex hull algorithm in practice. *Pattern Recognition Letters*, **3**(1), 29–34.

Moffat, Alistair, and Petersson, Ola. 1992. An overview of adaptive sorting. *Australian Computer Journal*, **24**(2), 70–77.

Munro, J. Ian, and Spira, Philip M. 1976. Sorting and searching in multisets. *SIAM Journal on Computing*, **5**(1), 1–8.

Nielsen, Frank. 1996. Output-sensitive peeling of convex and maximal layers. *Information Processing Letters*, **59**(5), 255–259.

Ochoa, Carlos. 2019. *Synergistic (Analysis of) Algorithms and Data Structures*. PhD thesis, University of Chile.

Petersson, Ola, and Moffat, Alistair. 1995. A framework for adaptive sorting. *Discrete Applied Mathematics*, **59**, 153–179.

Sen, Sandeep, and Gupta, Neelima. 1999. Distribution-sensitive algorithms. *Nordic Journal on Computing*, **6**, 194–211.

Sleator, D. D., and Tarjan, R. E. 1985. Amortized efficiency of list update and paging rules. *Communications of the ACM*, **28**(2), 202–208.

Exercises

Exercise 3.1 The HANOÏ TOWER problem is a classic example of recursion, originally proposed by Lucas (1883).[15] A recursive algorithm proposed in 1892 completes

[15] Recall the setup: The game consists of three identical rods and *n* disks of different sizes, which can slide onto any rod. The puzzle starts with all disks on the same rod, ordered from the biggest (at the bottom) to the

the task using $2^n - 1$ moves, and an easy argument shows that $2^n - 1$ moves are necessary. For this exercise, consider the variant in which we allow disks of equal size; everything else about the setup is the same as before. (A disk is allowed to be placed on another disk with the same size.) We call this the DISK PILE problem. In the extreme case, when all disks have the same size, the tower can be moved in a linear number of moves.

(a) Prove that there is an algorithm for DISK PILE that performs $\sum_{i\in\{1,\dots,s\}} n_i 2^{s-i}$ moves, where s denotes the number of distinct sizes and n_i the number of disks with size i.
(b) Prove that no algorithm can move such a tower with less than $\sum_{i\in\{1,\dots,s\}} n_i 2^{s-i}$ moves.
(c) What is the worst-case performance of your algorithm over all instances with a fixed value of s and a fixed total number of disks n?
(d) Which analysis is more fine-grained: the one for s and n fixed, or the one for $n_1, \dots, n_s$ fixed?

Exercise 3.2 Given an unsorted array A and an element x, the UNSORTED SEARCH problem is to decide whether A contains at least one element with the same value as x. The cost measure is the number of times that an algorithm probes an entry of A.

(a) What is the best possible optimality ratio for the UNSORTED SEARCH problem with respect to *deterministic* algorithms over instances of size k and r elements with the same value as x?
(b) What is the best possible optimality ratio for the UNSORTED SEARCH problem with respect to *randomized* algorithms over instances of size k and r elements with the same value as x?
(c) What is the best possible optimality ratio for the UNSORTED SEARCH problems with respect to randomized algorithms over instances of size k and σ distinct elements?

Exercise 3.3 Given a sorted array A and an element x, the SORTED SEARCH problem is to decide whether A contains at least one element with the same value as x. What are the best-possible optimality ratios for the SORTED SEARCH problem with respect to deterministic and randomized algorithms over instances of size k and r elements with the same value as x?

Exercise 3.4 In the ELEMENTARY INTERSECTION problem, the input is an element x and k sorted arrays $A_1, \dots, A_k$, and the goal is to decide whether x belongs to all k of the arrays. In the ELEMENTARY UNION problem, the input is the same but the goal is to decide whether x belongs to at least one of the k arrays.

smallest. A legal move consists of removing the topmost disk of one of the rods and placing it on top of the stack on one of the other two rods. A key constraint is that no move is allowed to place a larger disk on top of a smaller disk. The goal is to move the disks so that all are on a common rod (necessarily in sorted order) different from the one they started on.

(a) What is the best possible optimality ratio of an algorithm for the ELEMENTARY INTERSECTION problem over instances formed by k arrays of size n/k each, when ρ of those arrays contain an element of value equal to x?
(b) What is the best possible optimality ratio of an algorithm for the ELEMENTARY UNION problem over instances formed by k arrays of size n/k each, when ρ of those arrays contain an element of value equal to x?

Exercise 3.5 The algorithm Bubble Sort is well known to run in time $\Theta(n^2)$ in the worst case and $\Theta(n)$ for alreadysorted inputs. Consider the related procedures Bubble Up and `Bubble Down`, defined as follows: Bubble Up compares each pair of consecutive elements from the smallest index to the largest one, swapping them if inverted; while Bubble Up compares each pair of consecutive elements from the largest index to the smallest one, swapping them if inverted. In order to simplify the notation, suppose that the first and last elements of the array are $-\infty$ (at index 0) and $+\infty$ (at index $n+1$).

(a) Prove that a position p whose corresponding element is left unmoved by both Bubble Up and Bubble Down is a natural *pivot*: in the input array, the element is larger than all elements with smaller indices and smaller than all elements with larger indices.
(b) Prove that there is an algorithm sorting an array of n elements with η natural pivots in $O(n(1+\log\frac{n}{\eta}))$ time.
(c) Refine the previous proof to show that there is an algorithm sorting an array of n elements with η natural pivots separated by $\eta+1$ gaps of sizes $(r_0,\ldots,r_\eta)$ in $O(n+\sum_{i=0}^{\eta} r_i\log r_i)$ time.
(d) Prove that, in the worst case over instances formed by n elements with η natural pivots separated by $\eta+1$ gaps of sizes $(r_0,\ldots,r_\eta)$, every sorting algorithm in the comparison model runs in time $\Omega(n+\sum_{i=0}^{\eta} r_i\log r_i)$.

Exercise 3.6 The algorithm QuickSort is well known to have worst-case running time $\Theta(n^2)$ when sorting length-n arrays and using arbitrary pivots, and worst-case running time $\Theta(n\log n)$ when using median elements as pivots (using a linear-time median subroutine). Consider the implementation QuickSortWithRepetitions in which the partition induced by the median m yields *three* areas in the array: all elements of value strictly smaller than m on the left, all the elements of value strictly larger than m on the right, and all the elements of value *equal* to m in the remaining central positions.

(a) Prove that such an implementation performs $O(n(1+\log\sigma))$ comparisons in the worst case over instances formed by n elements from an alphabet with σ distinct values.
(b) Refine the previous proof and show that such an implementation performs $O(n+\sum_{i=1}^{\sigma} n_i\log\frac{n}{n_i})$ comparisons in the worst case over instances formed by n elements taken from the alphabet $[1,\ldots,\sigma]$, where n_i is the number of occurrences of the ith value.
(c) Prove a matching lower bound (up to constant factors) that applies to all order-oblivious algorithms.
(d) Can you combine the analysis in (b) with that of natural pivots (Exercise 3.5)?