

Exact Lower Bounds for the Number of Comparisons in Selection

Josua Dörrer

University Stuttgart

Konrad Gendle

University Stuttgart

Johanna Hofmann

University Stuttgart

Julius von Smercek


University Stuttgart

Andreas Steding

University Stuttgart

Florian Stober

University Stuttgart

Joan R. Public¹  

Department of Informatics, Dummy College, [optional: Address], Country

Abstract

Selection is the problem of finding the i -th smallest element among n elements. We apply computer search to find optimal algorithms for small instances of the selection problem. Using new algorithmic ideas we are able to go further than what has previously been possible. Our results comprise optimal algorithms for n up to 15 and arbitrary i , and for $n = 16$ when $i \leq 6$. We determined the precise values $V_7(14) = 25$, $V_6(15) = V_7(15) = 26$, and $V_8(15) = 27$, where previously, only a range was known.

2012 ACM Subject Classification Replace ccsdesc macro with valid one

Keywords and phrases Dummy keyword

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding Joan R. Public: [funding]

Acknowledgements I want to thank ...

1 Motivation

The problem of selecting the i -th smallest element in a list of n elements is a well-known problem in computer science called *selection*. Explicitly, we concern ourselves with the optimal worst-case selection of a single element from a set of initially unordered unique elements, measuring the cost by the number of comparisons made. We denote this cost as $V_i(n)$.

For selecting the smallest element, optimal algorithms are known with $V_1(n) = n - 1$. For the second smallest element it is known that $V_2(n) = n - 2 + \lceil \log n \rceil$ [7] (all logarithms are to base 2). In general, the selection problem is solvable in linear time using the median of medians algorithm [2]. Looking at the special case of selecting the median $i = n/2$, the best known algorithm requires $2.95n$ comparisons [4]. For other values of i , the algorithm in [4] requires fewer comparisons, thus providing a general upper bound. This presents a significant gap compared to the best known lower bound, which is $(1 + H(i/n)) \cdot n + \Omega(\sqrt{n})$, where $H(x) = x \cdot \log \frac{1}{x} + (1 - x) \log \frac{1}{1-x}$ [1]. For the median, this lower bound is $2 \cdot n - o(n)$. Paterson conjectured that the lower bound for selecting the median is $n \log_{4/3} 2 \approx 2.41n$ [11].

¹Optional footnote, e.g. to mark corresponding author



To improve these bounds toward tightness, it is essential to have known optimal reference points that general approaches can be compared against.

Gasarch, Kelly, and Pugh [5] were the first to use computer search to find optimal selection algorithms for fixed n and i . Oksanen continued this line of work, improving upon the previously known lower bounds [10]. His results and the computer program he used to obtain them are available on his website [9]. However, these results are not published in a scientific journal.

We will also tackle the selection problem using computer search. We will reimplement some of the existing ideas and add our own improvements, exploring the benefits of different search strategies, adding α - β -pruning, and exploiting compatible solutions. A quote from Miguel de Cervantes from Don Quijote will hold true for this article: “the journey is better than the inn” [3]. So buckle up.

1.1 Contribution.

In this work, we present a novel approach to finding optimal algorithms for selection. Using our approach, we obtained the following results:

1. We confirm most of the values $V_i(n)$ computed by Oksanen and correct an error in his work which states that $V_5(15)$ would be 25 [9]. We show that the optimal algorithm requires one comparison fewer, that is $V_5(15) = 24$.
2. We determined the precise values $V_7(14) = 25$, $V_6(15) = V_7(15) = 26$, and $V_8(15) = 27$. Previously, only a range of values was known for these instances.
3. We computed $V_i(16)$ for $i \leq 6$ and determined a better lower bound for $V_7(16)$ and $V_8(16)$.

Our algorithmic approach is twofold. The first approach, which we will call forward search in the remainder of this article, is an improvement to the minimax algorithm also used by Gasarch et. al. [5] and Oksanen [9, 10]. We introduce a novel pruning criterion based on the notion of compatible solutions.

The second approach, the backward search, is based on an entirely different idea. Here, the start and endpoint of the search switch places. This type of search has not been applied to the selection problem before, and we will see that its efficient application poses several challenges.

2 Fundamentals

2.1 Posets.

A partial order is a reflexive, transitive, and antisymmetric relation. A partially ordered set, short *poset*, is a set Ω with a partial order $P \subseteq \Omega \times \Omega$. By a slight abuse of notation, we denote the poset by P as well. When necessary, we write Ω_P to refer to the underlying set. Throughout this paper, Ω is finite. By E_n we denote the unordered poset on n elements, where each element is related only to itself. Two posets P and Q are *isomorphic* if there is a bijective mapping $\varphi : \Omega_P \rightarrow \Omega_Q$ such that $(u, v) \in P \iff (\varphi(u), \varphi(v)) \in Q$ for all $u, v \in \Omega_P$. The *dual* of a poset P is obtained by reversing the direction of all edges, i.e., $P^\delta = \{(v, u) \mid (u, v) \in P\}$. Given a poset P , its *Hasse diagram* H is given by the smallest subset $H \subseteq \Omega \times \Omega$ such that P is the reflexive, transitive closure of H . We denote by $P+ab$ the transitive closure of $P \cup \{(a, b)\}$. By $P|_{\Omega'}$ we denote the restriction of P to Ω' . The downset of an element a is $D_P(a) = \{b \in \Omega_P \mid b \leq a\}$, and the upset is $U_P(a) = \{b \in \Omega_P \mid b \geq a\}$.

2.2 The Selection Problem.

The selection problem is, given a poset P and an integer i , to determine the i -th smallest of the n elements in Ω_P where we already know the relation P . We denote an instance of the *selection problem*, or problem for short, by (P, i) . The notion of isomorphism naturally extends to selection problems. For the dual, we have $(P, i)^\delta = (P^\delta, n - i + 1)$. The problem (P, i) is *reduced* if each element has at most $i - 1$ smaller elements and at most $n - i$ larger elements. We denote the reduced problem corresponding to (P, i) by $\text{red}(P, i)$. Hence, each element can still be the i -th smallest.

2.3 Selection Algorithms.

A selection algorithm is a binary decision tree. Each node is labeled with a selection problem. The root node is labeled with (E_n, i) . The leaf nodes are labeled with *solved* problems (P, i) that have a unique element $a \in \Omega_P$, such that $|D_P(a)| = i$ and $|U_P(a)| = n - i + 1$. Thus, a is the i -th smallest element.

The selection algorithm associates each inner node (P, i) with a comparison $\{a, b\}$, meaning that the algorithm compares a with b as its next step. The two children, $(P+ab, i)$ and $(P+ba, i)$, correspond to the two possible outcomes of the comparison $a < b$ and $a > b$. The number of comparisons required by the algorithm (in the worst case) is the maximum length of a path from the root to any leaf.

2.4 Minimum Number of Comparisons.

Let $V_i(n)$ denote the minimum number of comparisons required to select the i -th smallest out of n elements in the worst case. We prove the following transfer lemma for lower bounds, showing that if k is a lower bound for selecting the i -th smallest from n elements, then selecting the i -th smallest from $n + 1$ elements requires at least $k + 1$ comparisons.

► **Lemma 1.** $V_i(n + 1) \geq V_i(n) + 1$.

Proof. Let $k = V_i(n + 1)$. There exists an algorithm that selects the i -th smallest from $n + 1$ elements using k comparisons. We now construct an algorithm that selects the i -th smallest from n elements using at most $k - 1$ comparisons. Let a and b be the two elements compared first by the algorithm for $n + 1$ elements. Replace a with a new element ω that is larger than any other element in the input of the algorithm. The algorithm still returns the i -th smallest of the remaining n elements. Any comparison involving $a = \omega$ can be skipped, as ω is always larger. In particular, the first comparison is skipped, reducing the number of comparisons by at least 1. Thus, we obtain an algorithm for selecting the i -th smallest from n elements using $k - 1$ comparisons. ◀

► **Remark 2.** It appears that Oksanen was unaware of Lemma 1, as the ranges provided in his table could be improved using this Lemma [9].

Note that the bound in Lemma 1 is tight for some instances, as can be seen from $V_1(n) = n - 1$. An easy corollary to the lemma is $V_{i+1}(n + 1) \geq V_i(n) + 1$, which follows immediately from the next lemma.

Let $V_i(P)$ denote the minimum number of comparisons required to select the i -th smallest element of the poset P in the worst-case. $V_i(n)$ is the special case $V_i(E_n)$. We prove the following lemma showing that the cost of selection remains unchanged when considering the dual problem.

127 ► **Lemma 3.** $V_i(P) = V_{n-i+1}(P^\delta)$

128 **Proof.** Given $V_i(P)$, we know there exists an algorithm that determines the i -th smallest
 129 element in P using exactly that many comparisons. By viewing this algorithm as a binary
 130 decision tree and swapping all the children, we obtain an algorithm for selecting the i -th
 131 largest element in P^δ , which is also the $(n - i + 1)$ -th smallest element of P^δ . ◀

132 2.5 Compatible Solutions.

133 Suppose we have a solved poset P with a unique i -th smallest element e . We then know
 134 precisely the set of elements that are smaller than e as well as the set of elements that are
 135 larger than e . This observation leads us to the notion of compatible solutions, which is such
 136 a partition compatible with the current relation.

137 ► **Definition 4.** *The solved problem (S, i) is a **compatible solution** of the problem (R, i) if*
 138 *$(a, b) \in S \implies (b, a) \notin R$ and S has no relations other than the $n - 1$ relations involving the*
 139 *i -th smallest element and those resulting from the application of transitivity to the former.*

140 Clearly, a solved problem has exactly one compatible solution. Let

$$141 \quad \mathcal{C}(P, i) = \{(S, i) \mid (S, i) \text{ is compatible with } (P, i)\}$$

142 be the set of all solutions compatible with (P, i) . Observe that, given two elements unrelated
 143 in P , every solution compatible with (P, i) is compatible with at least one of $(P+ab, i)$ and
 144 $(P+ba, i)$ and thus

$$145 \quad \mathcal{C}(P, i) = \mathcal{C}(P+ab, i) \cup \mathcal{C}(P+ba, i). \quad (1)$$

146 We use the concept of compatible solutions to derive a lower bound on the number of
 147 comparisons required to select the i -th smallest element of a poset P .

148 ► **Theorem 5.** *Selecting the i -th smallest element of a poset P requires at least $\lceil \log(|\mathcal{C}(P, i)|) \rceil$*
 149 *comparisons in the worst case.*

150 **Proof.** Assume we have an optimal algorithm for selecting the i -th smallest element of a
 151 poset P . From Equation (1), it follows that for every (S, i) , there is at least one leaf in the
 152 decision tree labeled with $\{(S, i)\}$. Hence, there are at least $|\mathcal{C}(P, i)|$ leaves, implying that
 153 the height of the tree is at least $\lceil \log(|\mathcal{C}(P, i)|) \rceil$. ◀

154 3 Methods and Tools

155 In this section, we describe our two main approaches to determining $V_i(n)$: the forward search
 156 and the backward search. The forward search follows the approach used by Oksanen [10].
 157 We enhance it by using a pruning technique based on compatible solutions. The backward
 158 search is a novel approach that has not been previously applied to the problem of selection.
 159 It allows us to further improve the computation of optimal selection algorithms.

160 3.1 Data Structures and Isomorphism Testing.

161 The key to reducing the search space is to consider only reduced problems and detect
 162 isomorphic problems. Isomorphism testing is performed by computing a *canonical* represent-
 163 ative. For the backward search, we use `nauty` [8] to compute a canonical representative. For

the forward search, we use a best-effort approximation to reduce the cost of computing the representative, at the expense of a slightly larger search space.

By Lemma 3, we do not need to distinguish a problem from its dual. We take advantage of this by switching to the dual if $i \leq \frac{n+1}{2}$. If $i = \frac{n+1}{2}$, one of the two is chosen deterministically. This is described in more detail in Section 3.3.3. A *normal* representative is a uniquely reduced representative of the isomorphism class of the problem and its dual.

We store posets as adjacency matrix. We choose a canonical representative in each isomorphism class so that we have a lower triangular matrix that can be stored using $\frac{n^2-n}{2}$ bits.

3.2 Forward Search.

The forward search algorithm is based on the work of Oksanen [10] and Gasarch et. al. [5]. We first describe the basic algorithm and then discuss the optimizations and pruning techniques we applied, including a novel pruning criterion based on compatible solutions.

The forward search starts with the problem (E_n, i) and recursively determines the cost of selecting the i -th smallest element of a poset P . Between the two possible outcomes of a comparison, we assume the worse. However, since the algorithm is free to choose which elements to compare, we seek the comparison with the lowest cost in the worst case outcome. Thus, the cost $V_i(P)$ can be expressed as follows:

$$V_i(P) = \min_{a,b \in \Omega_P} \max \{ V_i(P+ab), V_i(P+ba) \} . \quad (2)$$

The algorithms generated by the search program are built by saving, for each problem, the comparison that lead to the cheapest result.

To save memory and allow further pruning, we traverse the search tree using a depth-first search approach. This reduces the maximum number of comparisons assigned to child problems to one less than the best result currently found. This principle is implemented using a minimax algorithm, as shown in Figure 1.

3.2.1 Optimizations.

3.2.1.1 Caching.

We can significantly speed up the exploration by caching previous results, even with a simple usage-based ejection policy. Since the search always imposes an upper bound on the number of comparisons, this also includes unsolved posets, for which we record the currently known minimum.

3.2.1.2 Isomorphism Testing.

In the cache, we store an approximated canonical representative of a problem and its dual, allowing us to detect isomorphic problems. Preliminary tests showed that the performance gained by computing an approximated normal form, where some isomorphic problems may have different representatives, outweighs the cost induced by the larger search space.

3.2.1.3 Maximum Depth.

We use the minimax search algorithm to cut off unpromising branches. While searching the possible comparisons of a poset, we keep track of the current best result. The remaining comparisons are searched with a limited depth, ensuring that only solutions improving the

23:6 Exact Lower Bounds for the Number of Comparisons in Selection

■ **Algorithm 1** Algorithm for computing the number of compatible solutions for a given poset.

```

function NUMCOMPATIBLESOLUTIONS( $(P, i)$ )
   $c \leftarrow 0$ 
  for  $j \in \Omega_P$  do                                      $\triangleright$  solutions with  $j$   $i$ -th smallest
     $\mathcal{D} \leftarrow \{D_P(j) \setminus \{j\}\}$ 
    for  $k \in \Omega_P \setminus (D_P(j) \cup U_P(j))$  do
      for  $S \in \mathcal{D}$  do
        if  $D_P(k) \subseteq S \cup \{k\}$  then
           $\mathcal{D} \leftarrow \mathcal{D} \cup \{S \cup \{k\}\}$ 
        end if
      end for
    end for
     $c \leftarrow c + |\{S \in \mathcal{D} \mid |S| = i\}|$ 
  end for
  return  $c$ 
end function

```

204 current best result are found. At the start of a search, possible comparisons are sorted using
 205 a heuristic so that the most promising comparisons are searched for first.

206 3.2.2 Pruning.

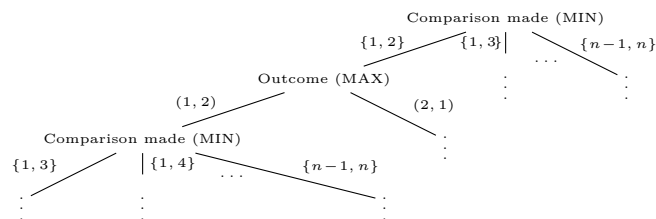
207 We use the following two pruning criteria to reject posets that are not solvable within the
 208 given number of comparisons.

209 3.2.2.1 Compatible Solutions.

210 The first pruning criterion uses the number of compatible solutions. As proven in Theorem 5,
 211 the log of the number of compatible solutions provides a lower bound for the cost of a given
 212 problem.

213 Algorithm 1 shows how we compute the number of compatible solutions for a given
 214 problem. To calculate this number, the algorithm first picks a solution element j – since
 215 problems are always reduced in the forward search, any element is valid – and then counts
 216 the number of partitions into greater and lesser elements, summing these counts over all
 217 solution elements. The algorithm assumes that the elements in the poset are sorted such
 218 that an element smaller than another has a smaller index.

219 As an example, the unordered poset (E_n, i) has $n \cdot \binom{n-1}{i-1}$ compatible solutions because,
 220 for each of the n elements, all separations of the remaining $n - 1$ elements are valid.



■ **Figure 1** Minimax search algorithm

3.2.2.2 Free Comparison.

The second pruning criterion aims to reduce the size of the searched subtree by adding a ‘useful’ comparison to eliminate elements faster. Explicitly, it searches for unordered elements u and v such that u has as many elements less than it and v has as many elements greater than it, and adds $u < v$ to the poset. The new problem is then searched for a solution using the forward search described above without reducing the number of allowed comparisons. If the new problem is not solvable, then the original problem cannot be solvable either. This is valid because adding a comparison ‘for free’ does not make the problem harder to solve.

3.3 Backward Search.

The development of the backward search is primarily based on the backward search method from [13], as it represents a new research area for the selection problem and was not addressed in the previous work by Oksanen [9].

The backward search starts with the set of solved selection problems, and iteratively removes comparisons until the unordered poset is found. Even if we restrict ourselves to a fixed cardinality n and rank i , the set of solved problems remains large. We solve this, by only enumerating reduced problems. With this restriction, the starting point of the backward search is $(E_1, 1)$.

3.3.1 Algorithm.

The input parameters for the backward search are denoted by n and i , similar to the forward search. The backward search starts with the solved problem $(E_1, 1)$ and iteratively computes all posets solvable using $k = 1, 2, 3, \dots$ comparisons until the unordered poset (E_n, i) is encountered.

Let A_k denote the set of all reduced selection problems solvable using k comparisons. For all n and i , we have $A_0 = \{(E_1, 1)\}$.

The backward search begins with A_0 and iteratively computes, for each problem in A_k , the corresponding predecessors, which form the set A_{k+1} . If $(E_n, i) \in A_\ell$, then $V_i(n) = \ell$.

3.3.2 Predecessor calculation.

We begin with a formal definition of a predecessor.

► **Definition 6** (Predecessor). *The problem (Q, j) is a predecessor of (P, i) if there is a comparison (a, b) such that:*

1. $(P, i) = \text{red}(Q+ab, j)$, and
2. $V_j(Q+ba) \leq V_i(P)$.

Any problem (Q, j) satisfying the first condition of the above definition is called a *potential predecessor*. In fact, the first step in enumerating the predecessors is to enumerate the potential predecessors. The second step is to check the second condition.

► **Lemma 7.** *Let (P, i) and (Q, j) be reduced problems, where (Q, j) is a predecessor of (P, i) . Then, $V_j(Q) \leq V_i(P) + 1$.*

Proof. Since (Q, j) is a predecessor of (P, i) , (a, b) exists with $\text{red}(Q+ab, j) = (P, i)$ and $V_j(Q+ba) \leq V_i(P)$. Therefore, $V_j(Q) \leq \max\{V_j(Q+ab), V_j(Q+ba)\} + 1 = V_i(P) + 1$. ◀

23:8 Exact Lower Bounds for the Number of Comparisons in Selection

Storing only reduced problems presents a significant challenge to predecessor enumeration. To illustrate this, consider a problem (P, i) and its predecessor (Q, j) . We know there exists a comparison (a, b) such that $(P, i) = \text{red}(Q+ab, j)$. However, it is possible that the edge (a, b) is not present in P because either a or b could have been removed during the reduction process. The question arises: How can we undo a comparison that is not visible? Furthermore, even if a and b are not removed during the reduction, there may be other elements that are removed. The challenge is to determine how many elements are removed and what their relationships are.

To address these challenges, we will prove two lemmas. The first lemma shows that after adding a comparison (a, b) , at most one of $\{a, b\}$ will be removed by the reduction.

► **Lemma 8.** *Let (P, i) be a reduced problem and let $(Q, j) = \text{red}(P+ab, i)$. Then, $Q \cap \{a, b\} \neq \emptyset$.*

Proof. Since (P, i) is reduced, we have $|D_P(c)| \leq i$ and $|U_P(c)| \leq n - i + 1$, where $n = |\Omega_P|$, for every $c \in P$. In particular, this holds for both a and b . Let $P' = P+ab$ and assume $Q \cap \{a, b\} = \emptyset$. Observe that $D_{P'}(a) = D_P(a) \leq i$ and $U_{P'}(b) = U_P(b) \leq n - i + 1$. Thus, for a and b to be removed, we must have $U_{P'}(a) \geq n - i + 2$ and $D_{P'}(b) \geq i + 1$. Note that there are no elements between a and b , as they are incomparable in P and there is a Hasse arc between them in P' . Hence, $U_{P'}(a) \cap D_{P'}(b) = \{a, b\}$, leading to the contradiction $n = |\Omega_{P'}| \geq |U_{P'}(a) \cup D_{P'}(b)| = |U_{P'}(a)| + |D_{P'}(b)| - |U_{P'}(a) \cap D_{P'}(b)| \geq n + 1$. ◀

The next lemma shows that the elements removed by the reduction, which are not a or b , can be added one after the other.

► **Lemma 9.** *Let (P, i) be a reduced problem, and let $(Q, j) = \text{red}(P+ab, i)$. If (P, i) is a predecessor of (Q, j) and $\Omega_P \setminus (\Omega_Q \cup \{a, b\}) \neq \emptyset$, there exists an element $c \in \Omega_P \setminus (\Omega_Q \cup \{a, b\})$ such that $(P|_{\Omega_P \setminus \{c\}}, i')$, where $i' = i - 1$ if $|U_{P+ab}(c)| \geq n - i + 2$ and $i' = i$ otherwise, is a reduced predecessor of (Q, j) .*

Proof. Let $R = \Omega_P \setminus (\Omega_Q \cup \{a, b\})$ be the set of elements removed by the reduction. It is easy to see that for every $c \in R$, the problem (P', i') where $P' = P|_{\Omega_P \setminus \{c\}}$ and $i' = i - 1$ if $|U_{P+ab}(c)| \geq n - i + 2$ and $i' = i$ otherwise is a predecessor of (Q, j) :

■ It is obvious that $(Q, j) = \text{red}(P'+ab, i')$.

■ The problem $(P'+ba, i')$ is at least as easy to solve as $(P+ba, i)$, hence $V_{i'}(P'+ba) \leq V_i(P+ba)$.

The challenge is to find an element c such that P' is reduced. We define the following sets:

$$C^+ = \{e \in \Omega_P \mid U_P(e) = n - i + 1\}$$

$$C^- = \{e \in \Omega_P \mid D_P(e) = i\}$$

$$R^+ = \{c \in V \mid U_{P+ab}(c) \geq n - i + 2\}$$

$$R^- = \{c \in V \mid D_{P+ab}(c) \geq i + 1\}$$

Note that $R = R^- \cup R^+$. The elements in C^- and C^+ are critical: If P' is not reduced, it is because one of these elements can no longer be the i -th smallest. To avoid this, we need an element $c \in R^+$ that is smaller (in P) than all elements in C^- . By symmetry, any element $c \in R^-$ larger than all elements in C^+ works as well. We first show that if we have an element in $R^- \cap C^-$ or $R^+ \cap C^+$, then this is the case as

$$\forall c \in C^+, e \in C^- : (c, e) \in P. \quad (3)$$

303 Assume we have $c \in C^+$ and $e \in C^-$, but $(c, e) \notin P$. Then the sets $U_P(c)$ and $D_P(e)$ are
 304 disjoint, leading to the contradiction $|U_P(c)| + |D_P(e)| = n + 1 > |\Omega_P|$.

305 The second step is to show that if $R^- \cap C^- = \emptyset$ and $R^+ \cap C^+ = \emptyset$, we can pick any $c \in R$.
 306 We show:

$$307 \quad \forall c \in R^-, e \in C^+ : (e, c) \in P \text{ or } e \in R^+. \quad (4)$$

308 Assume we have $c \in R^-$ and $e \in C^+$. By another counting argument we observe that the
 309 sets $D_{P+ab}(c)$ and $U_P(e)$ cannot be disjoint: Assuming $D_{P+ab}(c) \cap U_P(e) = \emptyset$ leads to the
 310 contradiction $|D_{P+ab}(c)| + |U_P(e)| \geq n + 2$. Thus, $(e, c) \in P+ab$. Hence, we have either
 311 $(e, c) \in P$ or $(e, a) \in P$. If $(e, a) \in P$ but $(e, c) \notin P$, we have to show $a \neq e$ to conclude
 312 $e \in R^+$. Assume $a = e$. Then $U_P(a) \setminus \{a\}$ and $D_{P+ab}(c)$ must be disjoint, as $(e, c) \notin P$. We
 313 obtain the contradiction $|U_P(a) \setminus \{a\}| + |D_{P+ab}(c)| \geq n + 1$. By symmetry, we also obtain:

$$314 \quad \forall c \in R^+, e \in C^- : (c, e) \in P \text{ or } e \in R^-, \quad (5)$$

315 which concludes the proof. \blacktriangleleft

316 The computation of a predecessor for a given problem (P, i) comprises three steps:

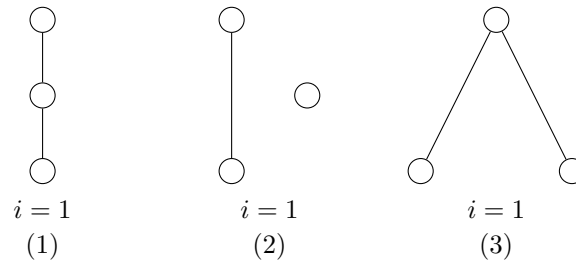
- 317 1. Compute predecessors on the same set of elements.
- 318 2. Compute predecessors with exactly one additional element that is involved in the com-
 319 parison.
- 320 3. Starting with the predecessors obtained in the preceding steps, add additional elements
 321 iteratively.

322 Of the predecessors to the problems in A_k , those that actually require $k + 1$ comparisons
 323 (those not in A_k) make up the set A_{k+1} . We describe the individual steps below.

324 3.3.2.1 Predecessors on the same set of elements.

325 First, we search for all posets with n elements that result in poset P after inserting a
 326 comparison $a < b$. In other words, we remove a comparison. First, we compute the potential
 327 predecessors. Each edge in the Hasse diagram of P potentially represents a comparison by
 328 which (P, i) can be obtained from a predecessor. A challenge arises from transitive relations,
 329 as the insertion of a single comparison can lead to the insertion of multiple transitive relations.
 330 This is illustrated in Figure 2. Removing a comparison from (1) can result in either (2) or
 331 (3). Therefore, both (2) and (3) are potential predecessors, even though the same comparison
 332 is removed each time.

333 The second step is to check whether each potential predecessor is actually a predecessor
 334 of (P, i) . For a potential predecessor (Q, j) where $(P, i) = \text{red}(Q+ab, j)$, we check whether
 335 $(Q+ba, j)$ can be solved using at most $V_i(P)$ comparisons. This is done by checking whether



■ **Figure 2** Case where further comparisons can be removed transitively by removing a comparison.

336 $\text{red}(Q+ba, j)$ is contained in one of the sets A_k for $k \leq V_i(P)$, which have already been
 337 computed.

338 **3.3.2.2 One additional element involved in the comparison.**

339 In the next step, all predecessors with $n + 1$ elements are computed, where the additional
 340 element is involved in the comparison. We construct the potential predecessor for this case
 341 as follows. We insert a new element into poset P and enumerate all possibilities for the
 342 relation between the new element and the existing elements. We want the predecessor to be
 343 reduced, thus it is crucial to ensure that the new element cannot be immediately reduced
 344 and that no existing elements can be reduced either. Since the new element is either smaller
 345 or larger than the i -th smallest element being searched for, either the $(i + 1)$ -th smallest or
 346 the i -th smallest element is searched for in these predecessors. Furthermore, for each (Q, j)
 347 obtained this way, there must exist elements a and b such that $\text{red}(Q+ab, j) = (P, i)$. This
 348 is required for (Q, j) to be a potential predecessor, and since we additionally want the new
 349 element to be part of the comparison, we mandate that either a or b is the new element.
 350 Checking whether the potential predecessors constructed this way are predecessors is done in
 351 the same way as in the preceding step.

352 For the correctness of our approach, note that if we have an arbitrary reduced predecessor,
 353 then by removing all elements that are not present in P , with the exception of a and b , we
 354 obtain another reduced predecessor. We get this by induction on the number of elements
 355 removed using Lemma 9. If both $a, b \in \Omega_P$ then this predecessor is enumerated in the first
 356 step. Otherwise, as proven in Lemma 8, if one of a or b is in Ω_P , the predecessor is enumerated
 357 in this second step. In the next step, we will iteratively enumerate all predecessors by adding
 358 additional elements to the ones already found. This way, we will discover the arbitrary
 359 predecessor we started with.

360 **3.3.2.3 Adding elements iteratively.**

361 In the third step, new elements are iteratively inserted. We alternate between generating new
 362 potential predecessors and checking which of those are actually predecessors. We stop when
 363 no new predecessors are found or when we reach an upper limit on the number of elements.
 364 New potential predecessors are generated by adding a new element to each predecessor and
 365 enumerating all possible relations with the existing elements. When inserting a new element,
 366 it is important to note that it may no longer be the i -th smallest but rather the $(i + 1)$ -th
 367 smallest element being searched, similar to the second step. We only consider potential
 368 predecessors that are reduced and of cause by the addition of the comparison $a < b$ and
 369 subsequent reduction, the resulting problem (P, i) should be obtained anew.

370 Figure 3 illustrates the backward search for $n = 4$ and $i = 2$.

371 **3.3.3 Normalform.**

372 Note that the backward search requires a unique normal form and cannot use the approxi-
 373 mation method applied in the forward search. The following outlines the computation of the
 374 normal form:

375 First, determine whether $i < n - i + 1$ holds. If not, replace the problem with its dual.
 376 According to Lemma 3, the cost of the problem remains unchanged.

377 Finally, the elements of the poset are arranged in a canonical order. We use **nauty** to
 378 obtain a canonical labeling of the elements. With this canonical labeling, the poset can be
 379 represented in its canonified form.

A potential issue arises if $i = n - i + 1$. In this case, it is impossible to decide whether (P, i) or $(P, i)^\delta$ corresponds to the normal form based solely on the value of i .

In Figure 4, the posets appear different in the Hasse diagram despite being each other's duals. To resolve this ambiguity, the dual poset is computed and canonified for each poset where $i = n - i + 1$ holds true. Subsequently, one of the posets is deterministically selected by comparing their binary representations.

Since canonification is inevitable for the backward search but consumes significant computational time, all simple cases are handled manually, and only the remaining cases are canonified using **nauty**.

First, compute the in- and out-degree for each node. Then, assign a hash value to each node based on these degrees, considering the recursive topological structure of adjacent nodes up to a specified depth limit.

The manual canonification process is as follows: Next, sort the nodes according to their hash values. If all hash values are unique, we have a canonical labeling. As it is often the case in the searched posets that two nodes have the same hash values, this case was intercepted.

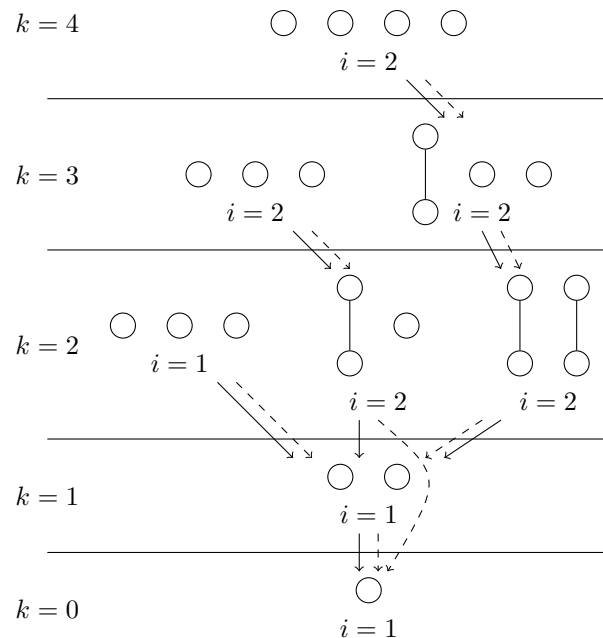


Figure 3 Search tree for $n = 4$ and $i = 2$. Level k contains all posets that can be solved in k comparisons and contribute to the solution for the given parameters n and i . Solid arrows indicate predecessors, while dashed arrows represent the resulting problem when the reversed comparison is inserted.



Figure 4 According to Lemma 3, the two posets are dual to each other. However, **nauty** cannot be used to transform the posets into each other since they are different graphs. This can also be seen in the Hasse diagram, which represents a directed graph, although the arrows are not shown here and always run implicitly from top to bottom.

23:12 Exact Lower Bounds for the Number of Comparisons in Selection

■ **Table 1** Percentage of canonification requiring **nauty** for variable n and i , where lower values are preferable.

n	i							
	1	2	3	4	5	6	7	8
13	0	30.205	6.808	1.526	0.467	0.185	0.114	
14	0	33.667	7.552	1.651	0.425	0.151	0.073	
15	0	36.390	8.184	1.678	0.459	0.132	0.065	0.041
16	0	39.407	8.805	1.796	0.467	0.144	-	-

Let there be l pairs of nodes with identical hash values. Then, there are 2^l possible posets that could correspond to the normal form, since each of the l pairs may or may not be swapped. Given the realistic assumption that l is small, all 2^l permutations can be efficiently iterated. With the aim of obtaining values for $n = 16$, it follows that there are at most $\frac{n}{2}$ pairs, hence $l \leq 8$ always holds. All 2^l posets are then calculated, and one permutation is deterministically selected based on its binary representation, similar to the case of the dual poset. It should be noted that this optimisation only works with pairs of two elements and no longer works if there are three nodes with the same hash value.

Implementing this canonification preprocessing significantly reduces the number of cases requiring **nauty**, as illustrated in Table 1.

It is particularly noteworthy that as i increases, the percentage of **nauty** calls decreases. For small values of i , the high percentage of **nauty** calls is not critical, as computations for small i are generally quick.

3.3.4 Optimizations.

3.3.4.1 Limit search space.

Since there are potentially many predecessors that cannot contribute to the solution, they are not even calculated. Many posets can be excluded based on n and i . As illustrated in Table 2, the search for $n = 7$ and $i = 4$ only considers predecessors that have an ‘x’ in the corresponding row or column. Since adding a comparison can reduce n by a maximum of 1 and therefore i by a maximum of 1, all other predecessors can be ignored as they can never contribute to the solution. For example, no poset with $n = 6$ and $i = 2$ can result in a poset of size $n = 7, i = 4$ by adding a comparison.

The table can be calculated by marking the initial n, i with an ‘x’ and then recursively marking the entries for $n - 1, i$ and $n - 1, i - 1$ with an ‘x’, as in each step the new element could be smaller or larger than the i -smallest element.

For Table 2, this means that $n = 6, i = 3$ and $n = 6, i = 4$ should be marked. It must be noted that $n = 6, i = 4$ does not exist, as the dual poset would be formed at this point. In this case, only $n = 6, i = 3$ is marked with an ‘x’.

3.3.4.2 Remaining comparisons.

In the next step, the minimum number of comparisons that must be removed until the unordered poset is reached is calculated for each predecessor. This number corresponds to the edges in the corresponding Hasse diagram. Since no more than one comparison can be

■ **Table 2** Possible predecessors that must be calculated for $n = 7$ and $i = 4$. All predecessors for which the corresponding field is marked with 'x' must be calculated.

n	i			
	1	2	3	4
7	-	-	-	x
6	-	-	x	-
5	-	x	x	-
4	x	x	-	-
3	x	x	-	-
2	x	-	-	-
1	x	-	-	-

■ **Table 3** Efficiency of parallelism for $n = 13, i = 7$

cores	1	2	3	6	12	24
time (m)	165	84	64	32	17	9
efficiency	1.00	0.99	0.90	0.87	0.81	0.75

removed in each step, all posets containing too many comparisons can be discarded, as they cannot lead to an unordered poset with the remaining comparisons.

3.3.4.3 Iterative deepening.

As the theoretical upper bounds are too high in practice, the program uses an iterative deepening approach. It starts with an upper bound that corresponds to the theoretical lower bound, derived by Lemma 1 from the smaller values for n and increments this bound until a solution is finally found. As it is not possible to save which posets are lost due to the guessed upper bound without considerable effort, the backward search is restarted several times. Although results from previous rounds are not used, the search space can be considerably reduced, making the program more efficient.

3.3.4.4 Parallelization.

The backward search can be ideally parallelized by performing the calculation of the predecessors in parallel. The only two bottlenecks here are read access to the cache and the efficient merging of all partial results.

As shown in Table 3 for $n = 13$ and $i = 7$, it can be seen that the backward search scales well with the number of cores. To set the different times in relation to the number of cores, the efficiency was determined, which represents a direct correlation between the two variables. This can be calculated as follows

$$\text{efficiency} = \frac{\text{single-core time}}{\text{number of cores} \cdot \text{multi-core time}}$$

The higher the efficiency, the better the time scales with the number of cores.

Figure 5 shows the size of the search space of the backward search for different values of i . It is noticeable that the maximum number of problems is searched for all i when there are 8 to 9 comparisons remaining, with a slight tendency towards more comparisons for larger i for $n = 14$.

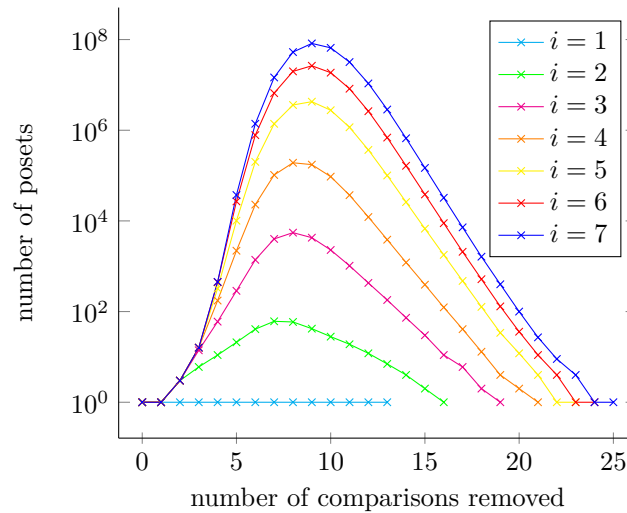
23:14 Exact Lower Bounds for the Number of Comparisons in Selection

■ **Table 4** Minimum number of comparisons needed to select the i -th smallest of n elements. Values resulting from our work are printed in bold.

n	i							
	1	2	3	4	5	6	7	8
1	0							
2	1							
3	2	3						
4	3	4						
5	4	6	6					
6	5	7	8					
7	6	8	10	10				
8	7	9	11	12				
9	8	11	12	14	14			
10	9	12	14	15	16			
11	10	13	15	17	18	18		
12	11	14	17	18	19	20		
13	12	15	18	20	21	22	23	
14	13	16	19	21	23	24	25	
15	14	17	20	23	24	26	26	27
16	15	18	21	24	26	27	28 - 33	28 - 36

4 Results

Running our computer search, we obtained the values $V_i(n)$ shown in Table 4. Our findings confirm most of the values computed by Oksanen [9]. Notably, $V_5(12) = 19$ contradicts a conjecture by Gasarch [5] that the optimum can be achieved using a “pair-forming algorithm”, where the first comparison of any singleton is with another singleton (in this case, the best pair-forming algorithm requires 20 comparisons). The values printed in bold were unknown previously. For $V_7(14)$, $V_6(15)$, $V_7(15)$, and $V_8(15)$, only a range was known prior. Oksanen



■ **Figure 5** Number of posets generated by the backward search for $n = 14$ depending on the number of comparisons for various i . Be aware of the logarithmic scale of the y-axis and that the reverse search does not add comparisons, but rather removes them.

incorrectly lists $V_5(15)$ as 25 on his website [9], although his search algorithm does produce the correct value of 24. The values for $n = 16$ have not been computed before; we provide all values for $i \leq 6$ and ranges for $V_7(16)$ and $V_8(16)$. The upper bound for the ranges is $V_i(n) \leq n - i + (i - 1)\lceil \log(n + 2 - i) \rceil$ [6].

To validate the upper bounds of the values we calculated, we checked the algorithm certifying each number on each of the $n!$ permutations. Certifying the correctness of the lower bound is nearly impossible. We computed each number twice using two different algorithms, the forward and the backward search. Thus, it is unlikely that the results are incorrect due to a coding error.

Table 5 compares the execution times of the different algorithms to find optimal selection algorithms. All experiments were conducted on a machine with two Intel Xeon CPUs, each equipped with 12 cores (24 threads), and a total of 768 GB of RAM. The forward search was started with 500 GB of RAM and restarted for each combination of n and i , ensuring no use of cached data from previous runs to provide comparability. The ‘Oksanen’ column presents execution times of Oksanen’s program [9] on our hardware, started with a cache size of 25 GB RAM. It was originally designed to use 400 MB RAM and 25 GB is close to a natural limit due to the use of 32-bit indices. Additionally, we measured the number of posets stored in the cache after the calculation, which can be found in Table 6.

To evaluate the potential of compatible solutions as pruning criterion, we determined the maximum number of compatible solutions encountered for a given cost for $n \leq 14$ and used that as a boundary in a subsequent run. For $n = 14, i = 7$, this resulted in a time of 1h 21m with $147 \cdot 10^6$ posets in the cache, representing improvements by factors of 11 and 6.3, respectively.

5 Conclusion and Open Questions

As stated in the motivation before: the road is better than the inn. Along the way we improved the forward search using a pruning criterion based on compatible solutions and evaluated a new algorithmic approach – the backward search coming to a final conclusion that both are valid. Between the structural constraints of our algorithms and the available hardware, $n = 16$ is likely the limit of feasible calculation for the current state. We believe that higher n are realistic with new algorithmic ideas and conclude this work by talking about promising directions for further research. The latest version of our software is available at GitHub (https://github.com/JGDoerrrrr/selection_generator).

5.0.0.1 Bidirectional Search.

The classical meet-in-the-middle approach for a bidirectional search will not work for the selection problem. This is illustrated in Figure 6: Assume the two searches meet after 11 comparisons have been performed. At this meeting point, both searches have already covered over 99% of their search space.

An alternative approach to the bidirectional search would be to first run the backward search, but restrict it to problems with specific properties such as having a large number of compatible solutions, e.g. problems (P, i) in A_k with $\mathcal{C}(P, i) \geq \alpha \cdot 2^k$ for some $\alpha \in [0, 1]$. These are presumably hard to solve, so the subsequent forward search only has to explore problems with a small number of compatible solutions, which we estimate to be easier to solve.

There might be better metrics than the compatible solutions, as there is a significant gap between the lower bound and the cost of a problem. We observed that the number of comparisons required to solve a selection problem is typically about twice the lower bound

23:16 Exact Lower Bounds for the Number of Comparisons in Selection

■ **Table 5** Execution times of different search methods.

n	i	Forward	Backward	Oksanen
12	1	0.0s	0.0s	0.0s
12	2	0.0s	0.2s	0.0s
12	3	0.4s	0.6s	0.0s
12	4	3.5s	0.9s	21.4s ^a
12	5	36.1s	3.8s	4m 59s ^a
12	6	1m 30s	18.0s	1.9s ^a
13	1	0.0s	0.0s	0.0s
13	2	0.0s	0.5s	0.0s
13	3	0.8s	1.2s	0.2s
13	4	13.8s	10.3s	55.4s
13	5	3m 42s	44.5s	26m 36s
13	6	17m 10s	3m 22s	3h 25m
13	7	59m 20s	7m 16s	16h 10m
14	1	0.0s	0.0s	0.0s
14	2	0.0s	1.3s	0.0s
14	3	1.4s	5.1s	0.6s
14	4	35.9s	33.0s	1m 47s
14	5	17m 27s	7m 1s	6h 29m
14	6	2h 40m	37m 54s	4d 10h
14	7	14h 40m	2h 17m	>5d ^b
15	1	0.0s	0.0s	0.0s
15	2	0.1s	3.9s	0.0s
15	3	2.8s	24.5s	1.4s
15	4	2m 24s	11m 2s	27m 17s
15	5	1h 12m	22m 11s	1d 5h 40m
15	6	1d 8h 37m	7h 17m	>5d ^b
15	7	4d 23h 37m	9h 45m	>5d ^b
15	8	14d 1h 51m ^c	1d 3h 7m	>5d ^b
16	1	0.0s	0.0s	-
16	2	0.2s	12.3s	-
16	3	6.4s	1m 55.1s	-
16	4	7m 22s	52m 9.4s	-
16	5	7h 33m	6h 48m 14.8s	-
16	6	6d 11h 21m	1d 1h 26m	-

^aThe publicly available version 1.6 of Oksanen's program did not find an optimal algorithm for $V_4(12)$, $V_5(12)$ and $V_6(12)$. On his website he gives an optimal algorithm computed with the unavailable version 1.1.

^bWe aborted Oksanen's program after 5 days.

^cThe 14 days for $n = 15$, $i = 8$ were measured with an older version of our program. The latest version would likely be a bit faster.

504 obtained from the number of compatible solutions, minus a constant. This observation is
505 reasonable, as the lower bound for the median is $n + o(n)$, which is far from the best known
506 asymptotic lower bound $2n + o(n)$.

■ **Table 6** Number of posets stored in the cache after the corresponding search

n	i	Forward Search	Backward Search
13	1	12	13
13	2	329	245
13	3	$9.7 \cdot 10^3$	$10.9 \cdot 10^3$
13	4	$199.7 \cdot 10^3$	$276.9 \cdot 10^3$
13	5	$3.7 \cdot 10^6$	$2.2 \cdot 10^6$
13	6	$18.1 \cdot 10^6$	$9.7 \cdot 10^6$
13	7	$67.6 \cdot 10^6$	$14.5 \cdot 10^6$
14	1	13	14
14	2	442	319
14	3	$15.2 \cdot 10^3$	$19.6 \cdot 10^3$
14	4	$438.0 \cdot 10^3$	$644.2 \cdot 10^3$
14	5	$14.1 \cdot 10^6$	$13.9 \cdot 10^6$
14	6	$149.5 \cdot 10^6$	$84.1 \cdot 10^6$
14	7	$925.3 \cdot 10^6$	$263.3 \cdot 10^6$
15	1	14	15
15	2	741	407
15	3	$23.6 \cdot 10^3$	$34.9 \cdot 10^3$
15	4	$1.3 \cdot 10^6$	$3.1 \cdot 10^6$
15	5	$53.0 \cdot 10^6$	$40.0 \cdot 10^6$
15	6	$1.6 \cdot 10^9$	$0.73 \cdot 10^9$
15	7	$5.3 \cdot 10^9$	$1.3 \cdot 10^9$
15	8	$15.7 \cdot 10^9$	$2.2 \cdot 10^9$
16	1	15	16
16	2	990	520
16	3	$35.8 \cdot 10^3$	$62.3 \cdot 10^3$
16	4	$2.4 \cdot 10^6$	$7.4 \cdot 10^6$
16	5	$211.1 \cdot 10^6$	$275.3 \cdot 10^6$
16	6	$3.6 \cdot 10^9$	$2.6 \cdot 10^9$

5.0.0.2 Improved Weight Function.

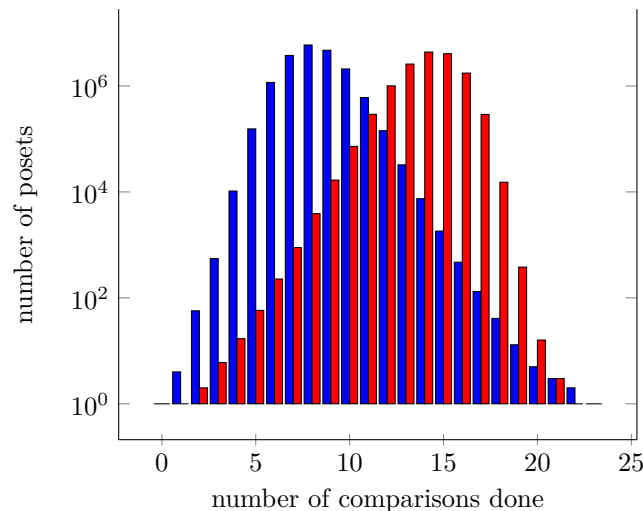
A better lower bound can be achieved by assigning a weight to each solution rather than merely counting compatible solutions. It is relatively straightforward to create a weight function that leads to a lower bound of $1.5n + o(n)$ for the median. We also tried to develop a weight function mimicking the techniques used to obtain the $2n + o(n)$ bound in [1]. However, this lead to a negative $4\sqrt{n}$ term in the resulting lower bound for $n \leq 16$, cancelling out any improvement over counting compatible solutions.

5.0.0.3 Yao's conjecture.

Yao conjectured that finding the i -th smallest of n elements is at least as hard as finding an n -element subset S of m elements, where $m > n$, and an element $s \in S$ such that s is the i -th smallest element in S [14]. If true, it would imply a $2.5n + o(n)$ algorithm for computing the median [12]. By adapting our search algorithm, one could search for counter examples to the conjecture.

References

- 1 Samuel W. Bent and John W. John. Finding the Median Requires $2n$ Comparisons. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 213–216, 1985. doi:10.1145/22145.22169.
- 2 Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time Bounds for Selection. *Journal of Computer and System Sciences*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.
- 3 Miguel de Cervantes Saavedra. *Don Quijote*. Madrid, 1605. Originally published as “El ingenioso hidalgo don Quixote de la Mancha”.
- 4 Dorit Dor and Uri Zwick. Selecting the Median. *SIAM Journal on Computing*, 28(5):1722–1758, 1999. doi:10.1137/S0097539795288611.
- 5 William Gasarch, Wayne Kelly, and William Pugh. Finding the i th largest of n for small i, n . *SIGACT News*, 27(2):88–96, Jul 1996. doi:10.1145/235767.235772.
- 6 Abdollah Hadian and Milton Sobel. Selecting the t -th Largest Using Binary Errorless Comparisons. Technical report, University of Minnesota, 1969. doi:11299/199105.
- 7 Donald Knuth. *The Art of Computer Programming, Vol. 3: Sorting and Searching*. Addison-Wesley, 1973.
- 8 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60:94–112, 2014. doi:10.1016/j.jsc.2013.09.003.
- 9 Kenneth Oksanen. Selecting the ‘ i ’th largest of ‘ n ’ elements, 2002. URL: <https://www.cs.hut.fi/~cessu/selection/>.
- 10 Kenneth Oksanen. Searching for Selection Algorithms. *Electronic Notes in Discrete Mathematics*, 27:77, 2006. ODSA 2006 - Conference on Optimal Discrete Structures and Algorithms. doi:10.1016/j.endm.2006.08.064.
- 11 Mike Paterson. Progress in Selection. In *Scandinavian Workshop on Algorithm Theory*, pages 368–379. Springer, 1996. doi:10.1007/3-540-61422-2_146.
- 12 A Schönhage, M Paterson, and N Pippenger. Finding the Median. *Journal of Computer and System Sciences*, 13(2):184–199, 1976. doi:10.1016/S0022-0000(76)80029-3.
- 13 Florian Stober and Armin Weiß. Lower Bounds for Sorting 16, 17, and 18 Elements. In *2023 Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 201–213. SIAM, 2023. doi:10.1137/1.9781611977561.ch17.



■ **Figure 6** Number of posets depending on the number of comparisons for $n = 13$ and $i = 7$ (red: backward search, blue: forward search).

551 **14** Foong Frances Yao. On Lower Bounds for Selection Problems. 1974. doi:1721.1/149426.

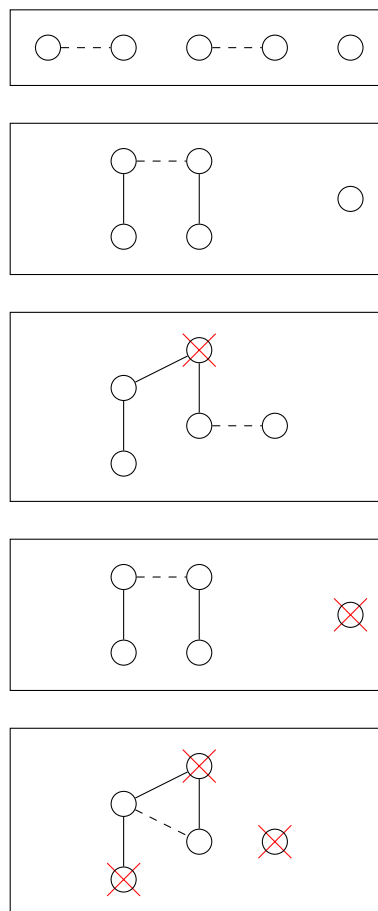
23:20 Exact Lower Bounds for the Number of Comparisons in Selection

552 **A** Example.

553 A good visual example is finding the median $i = 3$ in a list of $n = 5$ elements. Figure 7
 554 illustrates the search process of finding the median using Hasse diagrams. Each step shows
 555 the comparisons to be performed next, indicated by dashed lines. A Hasse diagram of the
 556 order relation found so far (with smaller elements positioned lower and larger elements
 557 higher) is shown with solid lines. The red crosses indicate elements that have been found to
 558 be greater or smaller than three other elements, thereby disqualifying them from being the
 559 median. The larger element of the final comparison is the median. This is also the optimal
 560 algorithm for $i = 3$ and $n = 5$.

561 **B** Hard- and Software Used.

562 Our results were facilitated by advancements in both hardware and software. All versions of
 563 the software used are listed in Table 7. For hardware, we employed two Intel Xeon E5-2650v4
 564 CPUs (2.20 GHz, 12 Cores/24 Threads, 30 MB L3-Cache per CPU), and a total of 768 GB
 565 of RAM.



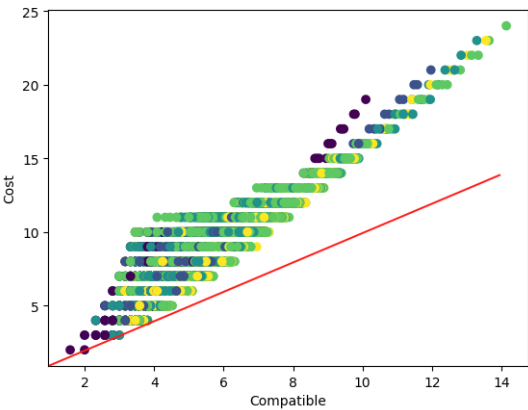
■ **Figure 7** Finding the median $i = 3$ of $n = 5$ values using six comparisons.

■ **Table 7** Specific versions of the software used.

Command	Output
<code>rustc -V</code>	rustc 1.77.2
<code>clang -v</code>	Ubuntu clang version 14.0.0-1
<code>uname -a</code>	Linux plankton 5.15.0-105-generic

566

C Distribution of Compatible Solutions



Cost distribution for all instances solved in the forward search for $n \leq 14$, excluding $i \leq 2$. Purple indicates low i , yellow indicates high i . \log_2 is shown in red.