A Las Vegas Algorithm for the Ordered Majority Problem

by

Ben Baral

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

In this thesis, we study the majority problem using ordered comparisons under the Las Vegas randomized algorithm model. The majority problem asks whether a given set of n elements, each with some colour, has a colour which appears on more than half of the elements. We focus on algorithms for this problem whose fundamental operation is to compare two elements, and in particular the comparison returns one of $\{<, =, >\}$. Additionally, we are interested specifically in Las Vegas randomized algorithms for this problem, which solve the problem correctly in all cases but whose running time is a random variable. Interestingly, most previous work studying this problem considers a different model where comparisons return just whether two elements are equal or not, instead of providing ordered information.

Our contribution is a novel Las Vegas algorithm that uses only $n + o(n)$ comparisons in the expectation, compared to $7n/6 + o(n)$ comparisons required in the expectation by the previous best algorithm for this problem.

Acknowledgements

First and foremost, I would like to thank my supervisor, Ian Munro. Ian gave me plenty of guidance throughout my degree and also suggested the problem and proposed the approach used in my thesis. He was wonderful to work with, and always a pleasure to talk to, whether it be about algorithms or curling. I would also like to thank my lab mate Anurag Naredla for always being there for me and listening while I talked through parts of my thesis. Thanks as well to Siwei Yang, a former student of Ian's whom I never met but whose work contributed to my thesis. I would also like to thank my parents, for always encouraging me to find what I was most interested in doing and giving me guidance and support along the way.

Finally, thanks to my committee members Naomi Nishimura and Lap Chi Lau for reading my thesis and giving constructive feedback.

Dedication

This is dedicated to my partner Julia.

Table of Contents

Chapter 1

Introduction

In this thesis, we study the majority problem, and in particular we focus on the ordered comparison model in a Las Vegas randomized algorithm context.

In this section, we give an overview of previous work on the majority problem, both in general as well as in specific models, as well as a formal introduction to the problem and relevant definitions. We begin with the classic algorithm for the problem due to Boyer and Moore [1], then we cover the optimal deterministic algorithm due to Salzberg [5]. Next, we discuss the problem in the Las Vegas randomized algorithm model, and specifically the previous best algorithm for the problem due to Gawrychowski et al. [7], which uses equality-test comparisons. Then, we survey the state of the art on solving the majority problem using 3-way comparisons.

1.1 Background

1.1.1 Comparison-based Algorithms and Problems

The class of comparison-based algorithms is a classic and well-studied aspect of computer science, and especially computational complexity.

Definition 1.1.1 (comparison-based algorithm)**.** An algorithm where the algorithm can only gain information about the input through comparing two elements.

Common problems often studied from the lens of minimizing the number of comparisons required include finding the minimum element, finding the k-th smallest element, finding

the majority element, and sorting a list of elements. There are still many fundamental problems unsolved in the field, however. For example, there is still a discrepancy between the best known lower bound and the best known upper bound for sorting.

Comparison-based problems are studied not only as a proxy for performance of algorithms but also as an interesting fundamental question to ask about the information needed to solve a problem. While many algorithms aiming to minimize comparisons are not useful in practice, they are an important concept to study for theoretical computer scientists.

Comparison Models

Within comparison-based algorithms, depending on the underlying data there are multiple different types of comparisons that can be performed.

If there is no ordering defined on the input elements, then a comparison can only return that two elements are equal or that they are not equal. We refer to this as the equality-test comparison model.

Definition 1.1.2 (equality-test comparison model)**.** A comparison model where each comparison between two elements returns only whether or not they are equal.

If there is an ordering defined, then a comparison between two elements α and β could return any of the three possibilities: $\alpha < \beta$, $\alpha = \beta$, or $\alpha > \beta$. We refer to this as the ordered comparison model.

Definition 1.1.3 (ordered comparison model)**.** A comparison model where each comparison between two elements returns whether or not they are equal, and if they are not, it returns their relative ordering.

Some problems that lend themselves to comparison-based algorithms, such as sorting, are viewed in a different light depending on the comparison model. Using only equality-test operations, there cannot exist an algorithm that sorts a set of numbers by their value, but there can exist an algorithm that partitions a set of numbers into all of its equal subsets. However, some problems can be studied similarly under either model. Of course, the ordered model is more powerful, because it indicates whether or not two elements are equal, but it also indicates their relative order in the event that they are not.

For some problems, there exist algorithms known for a long time under an ordered comparison model, but algorithms under the equality-test comparison model have not yet been discovered, or have only been discovered recently. For example, there have been

polynomial-time algorithms for computing statically optimal binary search trees under the ordered comparison model since 1959 [8] but it took until 2015 before there was a known polynomial time algorithm for the same problem under the equality-test model [2]. In fact, the best known algorithm for the problem under the ordered comparison model is quadratic in the size of the input, but the best known algorithm under the equality-test model is quartic.

1.1.2 Randomized Algorithms

A randomized algorithm is an algorithm that employs randomness in some part of its process, often to produce results using fewer operations than deterministic algorithms for the same problem.

Definition 1.1.4 (randomized algorithm)**.** An algorithm that employs randomness in some part of its process.

There are two classes of randomized algorithms, known as Monte Carlo algorithms and Las Vegas algorithms.

Definition 1.1.5 (Monte Carlo algorithm)**.** A class of randomized algorithms that is allowed to err with a certain (usually small) probability.

A common example of a Monte Carlo algorithm is the Miller-Rabin primality test [11] which aims to determine whether a given number is composite or prime.

Definition 1.1.6 (Las Vegas algorithm)**.** A class of randomized algorithms that always produces a correct result but whose running time is a random variable that depends on the input, though the expectation of the running time must be finite.

This means that some Las Vegas algorithms might never terminate (though the probability of this happening is infinitesimal), but in the expected case they terminate. Additionally, any Las Vegas algorithm that has bounded expected running time but that may never terminate can be modified to guarantee termination by branching to a deterministic method. A common example of a Las Vegas algorithm is the Quicksort algorithm using a randomly chosen pivot. The number of operations this algorithm performs is a random variable whose expected value is $\mathcal{O}(n \log n)$ for an input of size n.

It is worth noting that the expectation of the running time of a Las Vegas algorithm is averaged over possible random choices the algorithm could make on a fixed input, not

over all possible inputs. That is to say, the expected running time of such an algorithm is given by the average over the random choices made while processing the worst possible input for the algorithm.

Randomized algorithms are interesting to study because they often provide more elegant or more efficient solutions for certain problems than deterministic algorithms.

1.1.3 Majority Problem

In this work, we study the majority problem.

Definition 1.1.7 (majority problem)**.** The problem of determining, among an input of n values, whether there exists one value whose frequency is more than $\lfloor n/2 \rfloor$, and if so, producing such a value and the number of times it occurs.

Whereas much prior work on this problem studied it under an equality-test comparison model, we study the majority problem under the ordered comparison model, whereby there is a total ordering on values and a comparison returns not only whether the values are equal, but also their relative ordering if they are not. This problem has applications to voting systems and leader elections in distributed systems, i.e. to determine which candidate, if any, won a majority of the votes. We believe that it is a fundamental comparison-based problem similar to sorting and deserves to be investigated deeply.

1.2 Prior Work

1.2.1 Equality-Test Comparison Model

The majority problem was initially studied under the equality-test comparison model. The majority problem under this model can be formalized as follows: Given a set $S = \{1, 2, \ldots, n\}$ of elements and an equivalence relation `cmp` that determines whether two elements have the same colour, determine whether or not there exists a maximal equivalence class $V \subseteq S$ with $|V| > n/2$, and if there is, produce the cardinality of V and any element $v \in V$. We use "colour" here, as is common in the literature for this problem, to denote that there is no ordering associated with the elements; one can determine that two balls have the same colour, but cannot order them by colour.

Deterministic approaches

The problem of minimizing the number of comparisons required to determine a majority was first studied in 1980 (but not published until 1991) by Boyer and Moore [1], who gave a linear time algorithm that only requires a constant amount of space. Specifically, their algorithm, *MJRTY* (Algorithm 1), uses two passes through the input to achieve an upper bound of $2n - 2$ comparisons, and their result is a prototypical example of a streaming algorithm.[1] This algorithm, and almost all of the work done on this problem thus far, uses an equality-test comparison model, where the result of a comparison is simply "equal" or "not equal."

The algorithm works by making two passes through the data, the first to find a "majority candidate" and the second to determine if this element is actually a majority. The majority candidate element is guaranteed to be the same colour as the majority if there exists a majority in the input.

The first pass works by initializing a counter to zero and a reference to the current majority candidate. It increments the counter whenever it finds a new element that is equal to the current candidate, and decrements otherwise. If the counter reaches zero, the next element that the algorithm finds is set to the candidate. The second pass works by simply counting the number of instances of elements that are equal to the final candidate.

The *MJRTY* algorithm relies on an interesting fact that is very useful for the study of this problem. If one removes from a set two elements that are known to be not equal, if there is a majority before the removal, it is guaranteed to be preserved. This is because at most one of the two elements removed can be equal to a majority element, and so this removes at most one instance of a majority element and at least one instance of an element that is not the majority.

Because this is used very frequently throughout this thesis, we use the term "heterogeneous pair" to refer to a pair of elements that are known to be not equal to each other, and the term "homogeneous pair" to refer to a pair of elements that are known to be equal to each other.

Definition 1.2.1 (heterogeneous pair)**.** A pair of two elements that are known to not be equal.

Definition 1.2.2 (homogeneous pair)**.** A pair of two elements that are known to be equal.

[1] A streaming algorithm is an algorithm that only makes a small number of sequential passes through the input.

Algorithm 1 MJRTY(S)

```
1: cand ← S[1]
2: location ← 1
3: counter ← 0
4: for i = 1 to |S| do                          ▷ Pass 1: determines majority candidate
5:     if counter = 0 then
6:         cand ← S[i]
7:         location ← i
8:     if equal(S[i], cand) then
9:         counter++
10:    else
11:        counter--
12: counter ← 1
13: for i = 1 to |S| do                         ▷ Pass 2: determine if candidate is majority
14:    if i ≠ location and equal(S[i], cand) then
15:        counter++
16: if counter > |S|/2 then
17:    return cand is the majority with multiplicity counter in S
18: else
19:    return there is no majority in S
```

The *MJRTY* algorithm, in essence, is searching for heterogeneous pairs throughout the first pass, and each time it finds an element that compares not equal to the current candidate, it "discards" both values (which is represented by decrementing the counter). The counter keeps track of the number of surplus instances of the candidate seen so far, instances that can be discarded as soon as a non-equal element is found. Thus, after line 9, the candidate is guaranteed to be the majority if there is a majority. Then, the algorithm simply counts the total number of instances of this candidate to determine if it is, in fact, a majority.

In 1981, Fischer and Salzberg [5] improved on *MJRTY*; they proved that $\lceil 3n/2 \rceil - 2$ comparisons are necessary and sufficient for any deterministic algorithm that solves this problem. Interestingly, the lower bound proof is due to Fischer and the algorithm is due to Salzberg, but the results were published in the same paper. Salzberg's algorithm (Algorithm 2) behaves similarly to *MJRTY* but uses a clever optimization to reduce the number of comparisons needed on the second pass. The first pass, like *MJRTY*, finds a candidate for the majority element with the property that if there exists a majority element, it must be this candidate. Unlike *MJRTY*, though, it also partitions and reorganizes the data such that the second pass can spend fewer comparisons. The second pass then counts the number of elements that compare equal to this element in order to determine if the majority candidate found is a majority overall, using the organized data from the first pass.

We discuss the second pass first to give the intuition of how the data is organized in the first pass. The goal of the second pass is to count the number of elements that compare equal to the candidate element determined. In particular, in order to achieve the desired number of comparisons, the algorithm can only spend $\lceil n/2 \rceil - 1$ comparisons so it must be able to remove two elements using only one comparison.

At the beginning of the second pass, the overall set has been partitioned into two subsets: a list of assorted elements with the invariant that no two consecutive elements are the same, and a "bucket" comprising only elements that compare equal to the majority candidate. The algorithm repeatedly compares the last element of the list to the majority candidate. If it is equal, then the algorithm can discard the last two elements of the list (because the penultimate element cannot be equal to the last element, by the invariant). If it is not equal, then the algorithm can discard the last element of the list and one element from the bucket. If the bucket runs out, at most half of the elements remaining in the list can be the majority candidate, and each operation discards exactly one majority candidate and one non-majority candidate, so there cannot be a majority. Thus, the algorithm spends one comparison to discard two elements from the overall set, and this pass uses $\lceil n/2 \rceil - 1$ comparisons.

Now, we explain how the first pass produces the list and bucket with the invariants described above. The algorithm behaves very similarly to the *MJRTY* algorithm, and in fact performs the same comparisons. The aforementioned bucket of elements behaves the same as the counter from *MJRTY*, where adding something to the bucket is equivalent to incrementing the counter, and removing an element is equivalent to decrementing. Similarly, the last element of the list corresponds to the current majority candidate. The first element that the algorithm comes across gets added to the end of the list. After that, the algorithm compares the last element in the list to the next overall element. If it's the same, the algorithm adds the item to the bucket; otherwise, it adds the item to the end of the list and moves an element from the bucket to the end of the list (if there is one to remove from the bucket). Thus, this pass uses $n - 1$ comparisons to produce the list and bucket subject to the invariants above.

Overall, this algorithm uses $\lceil 3n/2 \rceil - 2$ comparisons in the worst case to solve the majority problem under the equality-test comparison model, which is optimal.

An alternative interpretation of the *MJRTY* algorithm leads to a simpler method also minimizing the number of comparisons. The first pass of *MJRTY* can be viewed as organizing the data into "blocks," where each block is a contiguous segment of elements with the same majority candidate. We refer to the majority candidate for a given block (i.e. the first element of the block) as the leader. Each block but the last has exactly half of its elements equal to the leader. The last block gives the only possible candidate (the overall majority candidate) and the margin by which it is the majority in that block.

The first pass performs $n - b$ comparisons, where b is the number of blocks that there are. The second pass compares all elements outside the last block with the final candidate, so potentially $2n - b - l - 1$ comparisons for the entire process, where l is the size of the last block. Unfortunately, MJRTY does not keep track of what it had learned in each block.

A simple twist is to record the size of each block, each block candidate, and (separately) all other elements in the first pass. Pass two now takes $(n - l)/2 + b - 1$ comparisons in the worst case. The total is, then, at most $3n/2 - l/2 - 2$.[2]

Randomized approaches

Gawrychowski et al. [7] were the first to look at this problem in the context of randomized algorithms. They produced a Las Vegas algorithm that solves the problem using $7n/6 + o(n)$ comparisons in the expectation, and in fact, it uses that many comparisons with very high

[2]Note that l can be 0, so this is not an improvement on Salzberg's algorithm.

Algorithm 2 SALZBERG(S)

```
1: L and B are empty stacks that support push, pop, and peek operations
2: for i = 1 to |S| do                                                    ▷ Pass 1
3:     if L is empty or cmp(peek(L), S[i]) = false then
4:         push S[i] to L
5:         if B is not empty then
6:             pop from B and append the element to L
7:     else
8:         push S[i] to B
9: T ← peek(L)
10: while L is not empty do                                               ▷ Pass 2
11:     if T = pop(L) then
12:         if size of L = 0 then
13:             push T to B
14:         else
15:             pop(L)
16:     else
17:         if size of B = 0 then
18:             return there is no majority in S
19:         else
20:             pop(B)
21: if |B| = 0 then
22:     return there is no majority in S
23: else
24:     return T is the majority in S with multiplicity ⌊(|S| + |B|)/2⌋
```

probability.[3] In this paper, note that we use Big-O terms ($\mathcal{O}, o, \Omega$) to mean strictly positive functions, unless there is explicitly a negative symbol in front of the term, in which case it refers to strictly negative functions.

Their algorithm works by selecting a random subset of elements without replacement large enough to be representative of the overall set but small enough that they can compute the most frequent two elements and their frequencies in the subset. Then, it proceeds with one of three subroutines depending on the distribution of the most frequent elements in the sample, which gives a hypothesis on the majority status. Each of the three subroutines aims to verify that the distribution found in the sample matches that of the overall set, using as few comparisons as possible. Since the sample was large enough to be representative, with high probability each subroutine simply validates the correctness of the assumption based on the sample. With low probability, sometimes a subroutine might report that the hypothesis was incorrect. In this case, the algorithm simply runs one of the deterministic algorithms for the problem, but since this happens infrequently, the average case is still better than the deterministic algorithm.

The three hypothesis distributions are that there is no majority (if no element in the sample had frequency above 45%), there is possibly a majority and it's one of two elements (if both elements had frequency close to 50% in the sample), or there is possibly a majority and it could only be one element (if this element had frequency above 45% in the sample).

Gawrychowski et al. also produced a lower bound for Las Vegas algorithms for this problem of $1.019n$ under the equality-test comparison model.

1.3 Ordered Majority Problem

Surprisingly, not much seems to be known about studying this problem under an ordered comparison model. This is a natural extension to consider because many common data types used, such as integers and strings, lend themselves nicely to ordered comparisons and computers generally return the ordered information when a comparison is performed.

The majority problem under this model can be formalized as follows: Given a set $S = \{1, 2, \ldots, n\}$ of elements with hidden values and a comparison function $\mathbf{cmp}(a, b)$ that determines whether the hidden value of a is less than, equal to, or greater than that of b, determine whether or not there exists a maximal subset $V \subseteq S$ with $|V| > n/2$ and all elements in V compare equal to each other. If such a set V exists, produce its cardinality and any element $v \in V$.

[3] The formal definition of very high probability will follow.

It is known that the lower bound of $\lceil 3n/2 \rceil - 2$ comparisons in the worst case due to [5] still holds [12]. However, the lower bound of $1.019n$ for the expected number of comparisons used by a Las Vegas randomized algorithm for the problem due to Gawrychowski et al. [7] applies only to the case of equality-test comparisons, and in fact our algorithm achieves a better expected number of comparisons.

Our main contribution is a Las Vegas-style randomized algorithm that solves the ordered majority problem using just $n + o(n)$ comparisons, specifically $n + \mathcal{O}(n^{1-f})$ for small positive f, in the expectation, beating the lower bound for the equality-test comparison model. In our case, we use $f = 1/20$, but this term could certainly be optimized.

Chapter 2

Randomized Algorithm for Ordered Majority Problem

First, we describe how our algorithm works at a high level, then we specify more details and parameters. The algorithm begins by producing a sample uniformly at random without replacement, with the intention that the sample is sufficiently large such that it is representative of the overall set. Then, it determines the frequencies within the sample of the most common elements. Based on these relative frequencies, it calls one of three subroutines, each of which is designed to validate a hypothesis about the overall set using few comparisons.

The three subroutines are each intended to quickly validate that the distribution in the overall set matches that of the sample, which occurs with high probability. Here are the three candidate distributions:

1. There are two elements which each comprise around half the sample. In this case, the most likely scenarios are that one of those two elements is a majority, or that they both narrowly miss the majority threshold and there is no majority.

2. There is exactly one element which comprises at least a little less than half the sample. The most likely scenarios for this case are that this element is a majority, or there is no majority.

3. There are no elements close to comprising half the sample. In this case, the most likely scenario is that there is no majority.

As long as the sampling phase and the appropriate one of these subroutines runs in $n + \mathcal{O}(n^{19/20})$ expected comparisons, our results hold. The lower-order term in the running time is somewhat arbitrary; the important part is that it is of the form $\mathcal{O}(n^{1-f})$ for some small positive f, and we chose parameters for our algorithm to achieve this running time with high probability.

2.1 Model

We denote a set of balls $S = \{1, 2, \ldots, n\}$ and use $\textbf{value}(v)$ to refer to the value of a ball v.[1] We use $\textbf{cmp}(v_1, v_2)$ to return one of $\{=, <, >\}$. We define $\textbf{less}(v_1, v_2)$ (and $\textbf{equal}(v_1, v_2)$) to return true if the value of v_1 is less than (resp. equal to) that of v_2.

2.2 Algorithm construction

Given an input set S of size n, the algorithm begins by constructing a random subset $S' \subseteq S$ uniformly at random without replacement. We use $|S'| = \delta(n)$ (definition to follow) for our sample size. Next, it computes statistics about S'; most notably, it determines the median, and the frequency of the two most frequent elements.

Let $v_1, v_2, \ldots$ be the values from S sorted by frequency of the elements within S' such that v_1 is the most frequent.

Let q_i be the fraction corresponding to the proportion of elements of S' that compare equal to v_i. Note that this means $q_i|S'|$ refers to the number of instances of elements that compare equal to v_i in S'.

Let m be the median of S'.

The constants that follow are set in order to give concrete values and to simplify the analysis, but a range of values works for these parameters.

- Let $\delta(n)$ be $n^{9/10} \log^2 n$.[2]

[1] We depart from the "colour" terminology used by previous papers on the subject because "colour" implies a lack of ordering.

[2] Note in particular that we chose a sample size larger than $\sqrt{n}$, which is a larger sample than an $\{=, \neq\}$ comparison algorithm could afford, because with ordered comparisons we can compute the necessary statistics of a sample of size x using $x \log x$ comparisons trivially by sorting, whereas an $\{=, \neq\}$ algorithm must perform x^2 comparisons. Also note that this sample size simplifies the analysis but is somewhat arbitrary.

- Let E be $n^{11/20}$.
- Let ϵ be $\frac{E}{n}$.
- Let b be $n^{1/20}$.
- Let c be $n^{2/5}$.

Then, the algorithm calls one of three different subroutines based on which distribution the algorithm supposes the input is closest to.

- If $q_1, q_2 \in [\frac{1}{2} - c\epsilon, \frac{1}{2} + c\epsilon]$, then it defers to TwoCandidates(S, v_1, v_2)
- Else if $q_1 \geq \frac{1}{2} - b\epsilon$, then it defers to OneCandidate(S, v_1)
- Else, it defers to NoCandidates(S, m)

See Algorithm 3.

Algorithm 3 Majority(S)

if $|S| = 1$ **then return** $S[1]$ is the majority with multiplicity 1 in S
randomly sample without replacement $S' \subseteq S$ such that $|S'| = \delta(n)$
let $v_1, v_2, \ldots, v_k$ be the representatives of the equivalence classes of values in S'
let $q_i|S'|$ be the number of elements that compare equal to v_i in S', where $q_1 \geq q_2 \geq \cdots \geq q_k$
let m be the median of S'
if $q_1, q_2 \in [\frac{1}{2} - c\epsilon, \frac{1}{2} + c\epsilon]$ **then**
    return TwoCandidates(S, v_1, v_2)
else if $q_1 \geq \frac{1}{2} - b\epsilon$ **then**
    return OneCandidate(S, v_1)
else
    return NoCandidates(S, m)

2.2.1 TwoCandidates

The TwoCandidates(S, α, β) subroutine takes as input the overall set S and two representative elements whose frequencies are both around half in the subset, α and β. The

subroutine first pairs elements, compares each pair, and divides the pairs into homogeneous pairs and heterogeneous pairs. For the heterogeneous pairs, it puts the smaller element first. Then, it recurses on the elements in the homogenous pairs. Note that if there exists a majority in the subproblem, it must be the majority of the overall set if there is one, and the absence of a majority in the subproblem implies there does not exist a majority in the overall set either.

If it does return that there was a majority in the subproblem, we now must validate whether it is the majority of the overall set. The subroutine returns the multiplicity of the candidate in the subproblem, so we only have to look at the heterogeneous pairs. In fact, most pairs will comprise the two frequent elements, and their relative ordering indicates which element of the pair to check first, so we can often avoid extra comparisons.

This algorithm is used with modification from [7].

See Algorithm 4.

2.2.2 OneCandidate

The OneCandidate(S, α) subroutine takes as input the overall set S and a representative value for a very frequent element expected to either be the majority or narrowly miss being the majority, according to the sample. It then iterates through S, comparing each element to α. For each element that it finds equal, it increments a counter; for each element less than α, it places it in a set X; for each element greater than α, it places it in a set Y. If the counter is greater than half the size of S, then it returns that α is the majority. If neither X nor Y contain more than $|S|/2$ elements, then it returns that there is no majority, as all elements of a majority would have to be in exactly one of those sets.

Otherwise, it attempts to disprove the existence of a majority by iterating through pairs in the larger of the two non-equal sets. We will call the larger set A. If the algorithm finds sufficiently many heterogeneous pairs, there cannot exist a majority. It compares consecutive elements in A until it has found enough heterogeneous pairs to disprove a majority. If it still cannot disprove a majority, it falls back to Salzberg's algorithm. The OneCandidate(S, α) subroutine is used with slight modification from [7].

We note here that using ordered comparisons here does not improve the behaviour of the algorithm for certain inputs. The algorithm works and achieves the correct running time bound if we simply put elements into "equal" and "not equal" sets, rather than partitioning them into three sets. As written, we use fewer comparisons in certain cases (if both sets are smaller than $|S|/2$, for example, there cannot be a majority). However,

Algorithm 4 TwoCandidates(S, v_1, v_2)

randomly shuffle S
$X \leftarrow [], Y \leftarrow []$
for $i = 1$ **to** $|S|/2$ **do**
    switch cmp$(S[2i-1], S[2i])$ **do**
        case $=$
            append $S[2i]$ to X
        case $<$
            append $S[2i-1], S[2i]$ to Y
        case $>$
            append $S[2i], S[2i-1]$ to Y
run Majority(X)
if there is no majority in X **then return** no majority in S
let **value**(v) be the majority value with multiplicity k in X
cnt $\leftarrow 2k$
Check1stElemFirst $\leftarrow$ (**equal**(v, v_1) and **less**(v_1, v_2)) or (**equal**(v, v_2) and **less**(v_2, v_1))
for $i = 1$ **to** $|Y|/2$ **do**
    if Check1stElemFirst **then**
        $i_1 \leftarrow 2i - 1$
        $i_2 \leftarrow 2i$
    else
        $i_1 \leftarrow 2i$
        $i_2 \leftarrow 2i - 1$
    if equal$(v, Y[i_1])$ or **equal**$(v, Y[i_2])$ **then** ▷ The "or" short circuits
        cnt $\leftarrow$ cnt $+ 1$
if cnt $\leq |S|/2$ **then**
    return no majority in S
else
    return value(v) is the majority value with multiplicity cnt in S

this does not improve the average-case running time, because there exist inputs where the algorithm behaves the same way in either case.[3]

See Algorithm 5.

Algorithm 5 OneCandidate(S, α)

```
X ← [], Y ← [], Z ← []
for i = 1 to |S| do
    switch cmp(S[i], α) do
        case <
            append S[i] to X
        case =
            append S[i] to Y
        case >
            append S[i] to Z
if |Y| > n/2 then return value(α) is the majority value with multiplicity |Y| in S
Let A be the larger set of {X, Z}
if |A| ≤ n/2 then return no majority in S
k ← |A| − n/2
randomly shuffle A
for i = 1 to |A|/2 do
    if not equal(A[2i − 1], A[2i]) then k ← k − 1
    if k = 0 then return no majority in S
return Salzberg(S)
```

2.2.3 NoCandidates

The NoCandidates(S, m) subroutine takes as input the overall set S and the median of S'. It then compares every element in S to m and partitions S into three subsets, X, Y, and Z, whose elements are less than, equal to, and greater than m, respectively.[4] If none of the subsets have cardinality greater than $\frac{n}{2}$, it declares that there is no majority.

[3] For example, if all elements in S compared less than or equal to α, the subroutine would produce three sets, but all of the "not equal" elements would be in the "less than" set, so it would be isomorphic to having only "equal" and "not equal" sets.

[4] Note that in this case, unlike in section 2.2.2, there *is* a benefit to using ordered comparisons and three sets (less than, equal to, and greater than) as opposed to two sets (equal to, not equal to), because

If $|Y| > \frac{n}{2}$, then it declares that m is the majority with cardinality $|Y|$. If $|X| > \frac{n}{2}$ or $|Z| > \frac{n}{2}$ it iterates through the larger subset until it finds enough heterogeneous pairs to disprove a majority. If it iterates through the set and finds that there are not enough heterogeneous pairs to disprove a majority (as discussed in the previous section), it falls back to Salzberg's algorithm.

See Algorithm 6.

Algorithm 6 NoCandidates(S, m)

```
X ← [], Y ← [], Z ← []
for i = 1 to |S| do
    switch cmp(S[i], m)) do
        case <
            append S[i] to X
        case =
            append S[i] to Y
        case >
            append S[i] to Z
if |Y| > n/2 then return value(m) is the majority value with multiplicity |Y| in S
if |X| ≤ n/2 and |Z| ≤ n/2 then return no majority in S
Let A be the larger set of {X, Z}
k ← |A| − n/2
randomly shuffle A
for i = 1 to |A|/2 do
    if not equal(A[2i − 1], A[2i]) then k ← k − 1
    if k = 0 then return no majority in S
return Salzberg(S)
```

the element we are using to produce these sets is with high probability close to the median, so there is no distribution where it is forced to be an extreme value.

2.3 Algorithm Analysis

2.3.1 Preliminaries

In this paper, we use Big-O terms $(\mathcal{O}, o, \Omega)$ to mean strictly positive functions, unless there is explicitly a negative symbol in front of the term, in which case it refers to strictly negative functions.

We define an event A to happen with very high probability if $\mathbf{Pr}(A) \geq 1 - \exp(-\Omega(\log^2 n))$.

Definition 2.3.1 (very high probability)**.** The probability of an event is at least $1 - \exp(-\Omega(\log^2 n))$.

Henceforth, we use w.v.h.p. to mean with very high probability.

Note that this is a non-standard definition (though it was used in the Las Vegas randomized algorithm by Gawrychowski et al. [7]). The primary reason for the usage of this definition is that the intersection of polynomially many events that occur with very high probability also occurs with very high probability, which is essential for the correctness of our algorithm analysis.[5]

Lemma 2.3.1 (Sampling)**.** *Let $X \subseteq S$ such that $|X| = j$. We sample uniformly at random $k \leq n$ elements from S without replacement. Let j' denote the number of elements from X in these k elements. Then w.v.h.p. $|j'/k - j/n| \leq k^{-1/2} \log n$.*

This lemma is reprinted with slight modification from [7]. Note that for $k = \delta(n) = n^{9/10} \log^2(n)$, this implies that the absolute difference of j'/k and j/n is bounded above by E.

We use **Hypergeometric(**N, K, n**)** to refer to the hypergeometric distribution that describes the number of red balls drawn in n draws without replacement from a finite set of size N with exactly K red balls. We define the rank of an element to be its position in sorted order of a list that contains it, and for equal values we disambiguate them arbitrarily but consistently.

The following median lemma is a folklore result, but its analysis is similar to some analyses found in Motwani and Raghavan [10].

[5] A standard definition of high probability might be that an event A happens with high probability if $\mathbf{Pr}(A) \geq 1 - n^{-\mathcal{O}(1)}$, but this weaker requirement does not meet this requirement.

Lemma 2.3.2 (Median)**.** *If we take a sample S' of size s and compute its median m, the rank of m in S is between $\frac{n}{2} - E$ and $\frac{n}{2} + E$ with probability at least $1 - 2e^{-2\epsilon^2 s}$.*

Proof. Consider the set Y of elements comprising the first $\frac{n}{2} - E$ elements in the sorted order of S, and the set Z comprising the last $\frac{n}{2} - E$ elements in the sorted order of S. Let us consider the case where we produce a median whose absolute distance to the "true median," that is, the element whose rank in S is $\frac{n}{2}$, is greater than E. In this case, we must have sampled more than $|S'|/2$ elements from Y, or sampled more than $|S'|/2$ elements from Z. Since these cases are symmetric, we can use a union bound to show that the probability of choosing a "bad" median, i.e. an m whose rank in S is not within E of $\frac{n}{2}$, is at most twice the probability of choosing more than $|S'|/2$ elements from Y.

Let us define a random variable X that corresponds to the number of elements chosen from Y. Note that X follows **Hypergeometric(**$\frac{n}{2} - E$, n, s**)**. We can thus use the hypergeometric tail bound [9] given by the following:

$$\mathbf{Pr}\big(X \geq (p+t)|S'|\big) \leq e^{-2t^2|S'|}$$

with $p = \frac{1}{2} - \epsilon$. If we substitute $t = \epsilon$, then we get

$$\mathbf{Pr}\left(X \geq \frac{s}{2}\right) \leq e^{-2\epsilon^2 s}$$

So we produce an appropriate median with probability at least $1 - 2e^{-2\epsilon^2 s}$. □

Note that for $s = \delta(n)$, this occurs with very high probability.

Lemma 2.3.3 (Heterogeneous Pairs)**.** *Consider a partition $\{X_1, X_2\}$ of a set X such that $\frac{|X_1||X_2|}{|X|-1} \in \Omega(\log^2 n)$. We use (X_1, X_2) to refer to a pair of elements comprising one element from X_1 and the other from X_2. We draw pairs from X at random, and we want to find at least r (X_1, X_2) pairs. Using $d = \frac{4(|X|-1)|X|r}{|X_1||X_2|}$ draws, we find at least r (X_1, X_2) pairs with probability at least $1 - e^{-\frac{|X_1||X_2|r}{|X|(|X|-1)}}$.*

Proof. We will reason about drawing random pairs from X by considering the set of pairs of elements P produced by randomly pairing all elements of X. First, we will establish that with very high probability there is a sufficient number of (X_1, X_2) pairs in P. Thus, we can reason about the expected number of such pairs we find after drawing d elements from P.

Let Q be a random variable representing the number of (X_1, X_2) pairs in P. First, we discuss the expected value of Q. Note that $|P| = |X|/2$.

We can write

$$Q = \sum_{i=1}^{|P|} Q_i$$

where Q_i is an indicator variable that is 1 if the i-th pair has one element from X_1 and the other from X_2, and 0 otherwise. If we consider this event without knowing about other draws, we have

$$\mathbf{Pr}\big(Q_i = 1\big) = \frac{2|X_1||X_2|}{|X|(|X|-1)}$$

(because the pairs can occur in either order).

Thus, $\mathbb{E}(Q) = \frac{|X_1||X_2|}{|X|-1}$. We now want to show that the concentration around this quantity is relatively tight. We set

$$q = \frac{1}{2}\mathbb{E}(Q) = \frac{1}{2}\frac{|X_1||X_2|}{|X|-1}$$

Now, we will show that

$$\mathbf{Pr}\big(Q > q\big) \geq 1 - e^{-q/4}$$

The indicator variables Q_i are not independent; indeed they are negatively correlated, so we apply a Chernoff bound [4]. We have

$$\mathbf{Pr}\big(Q > (1-\delta)\mathbb{E}(Q)\big) \geq 1 - e^{-\delta^2\mathbb{E}(Q)/2}$$

If we set $\delta = \frac{1}{2}$, we get

$$\mathbf{Pr}\big(Q > q\big) \geq 1 - e^{-q/4}$$

Note that since $\frac{|X_1||X_2|}{|X|-1} \in \Omega(\log^2 n)$ (by a condition of this lemma), $Q > q$ with very high probability.

Now, we want to show that drawing d pairs from P will give us at least r (X_1, X_2) pairs with very high probability. Let's call the number of such pairs we find R, assuming that there are q such pairs in P. Note that q is a lower bound on the actual number of (X_1, X_2) pairs.

We note that R follows a hypergeometric distribution since we are drawing elements without replacement, so we can use hypergeometric tail bounds.

We have $\mathbf{Pr}\big(R > (p-t)d\big) \geq 1 - e^{-2t^2 d}$ where $p = \frac{q}{|P|}$ (the fraction of (X_1, X_2) pairs in P) for any $t \in (0, dp)$ by the tail bounds of hypergeometric distributions [9]. We set $t = \frac{p}{2}$ and $d = 2p^{-1}r$.

Thus, we get

$$\mathbf{Pr}\big(R > (p-t)d\big) = \mathbf{Pr}(R > r) \geq 1 - e^{-pr} = 1 - e^{-\frac{|X_1||X_2|r}{|X|(|X|-1)}}$$

Note that if $pr = \frac{|X_1||X_2|r}{|X|(|X|-1)} \in \Omega(\log^2 n)$, this happens with very high probability. □

2.3.2 Algorithm analysis

We argue that our algorithm solves the majority problem using $n + \mathcal{O}(n^{19/20})$ ordered comparisons with very high probability, and in fact uses $n + \mathcal{O}(n^{19/20})$ comparisons in the expectation. Since our algorithm is a Las Vegas algorithm, our running time is a random variable, but our algorithm is always correct. Note that our goal with this algorithm is to produce a Las Vegas algorithm that uses $n + o(n)$ comparisons, and we do not aim to minimize the $o(n)$ term. We prove that our algorithm follows the recurrence relation $T(n) = n + \mathcal{O}(n^{19/20})$ by strong induction on n.

2.3.3 Sampling phase

Our algorithm produces a sample S' by drawing uniformly at random $\delta(n) = n^{9/10} \log^2 n$ elements without replacement. Then, it computes the frequencies of the two most frequent elements within the sample, and determines the median of the sample.

We can clearly compute these statistics in time $\mathcal{O}(|S'| \log |S'|)$ by sorting the sample. Note that we can afford to sort the sample within our desired bound on the number of comparisons spent, which could not be done on a sample this large in a 2-way comparison model.

We note, in passing, that in fact we can tune down the running time of this operation. We can determine the high frequency values required (or their absence) in time linear in the sample size, though it has little bearing on our majority theorem, as we can afford to sort the sample without impacting the worst case analysis. If an element occurs close to $|S'|/2$ times, it must be at least one of the upper quartile, lower quartile, or median values. Using the selection algorithm of Floyd and Rivest [6], we find the median and then apply it again to the elements greater than and less than this median to find the quartiles. With a

3-way comparison, it is straightforward to get the frequencies of any value occurring more than about $|S'|/4$ times. The number of comparisons used will be $3|S'| + o(|S'|)$ with very high probability. Following the expected case median lower bound of Cunto and Munro [3], $3|S'| - o(|S'|)$ comparisons are necessary in the expected case.

2.3.4 TwoCandidates

This subroutine starts by pairing all elements and comparing the elements of consecutive odd-even pairs. This uses $n/2$ comparisons. We construct a set X comprising one element from each homogeneous pair found through the pairing process, and another set Y comprising each heterogeneous pair pair found, with the smaller element first.

The next step of the algorithm is to recurse on X. Note that if there is a majority in S, it must be the majority of X as well. That is, if the recursive call returns that there is no majority in X, then the algorithm reports that there is no majority.

First, we note that we expect $|X|$ to be approximately equal to $|S|/4$ given that our algorithm ended up in this subroutine, since we expect there to be two elements that occur around half the time each in the overall set based on the sample. Precisely,

$$|X| = \left(\frac{1}{4} \pm \mathcal{O}(c\epsilon)\right)n$$

w.v.h.p. [7]. Thus, it follows that w.v.h.p. $T(|X|) \leq \frac{n}{4} + \mathcal{O}(cE) + \mathcal{O}(n^{19/20})$.

If our algorithm returns here, then it has spent only $\frac{3n}{4} + \mathcal{O}(n^{19/29})$ comparisons, which is within our bound. If it continues, it must determine if the majority element of X is a majority in S as well. Note also that the recusive call will return the cardinality of the majority candidate in X, so all that is left to do is determine whether there are enough instances of it in Y to declare a majority.

We can establish that $|Y| = (\frac{1}{2} \pm \mathcal{O}(c\epsilon))n$ w.v.h.p. [7].

The algorithm then iterates through pairs in Y. Each of the following statements occur with very high probability. Note that Y comprises almost exclusively pairs of both α and β. In particular, at most $\mathcal{O}(cE)$ pairs within Y contain an element that is neither α nor β. This is because there are at most $\mathcal{O}(cE)$ elements in S are neither α nor β.

We then iterate through Y, pair by pair, and declare whether there is a majority based on how many elements we found that were equal to our majority candidate found via recursion. We claim that for all pairs comprising both α and β, we must only spend

one comparison to properly count all instances of our majority candidate. Let us assume without loss of generality that α is the majority candidate and $\alpha < \beta$. For pairs that contain both frequent elements, we only spend one comparison because we compare the first element of each pair to α, and find it, so we don't have to compare the other element to α. (Note that if $\alpha > \beta$ then we would compare the second element instead.)

For pairs that don't contain both α and β, we spend either one or two comparisons, because we might find our candidate on the first comparison, might find it on the second, or might not find it at all. But, as mentioned, w.v.h.p. there are at most $\mathcal{O}(cE)$ pairs not comprising both of the frequent elements. Thus, this loop thus costs at most $2 \times \mathcal{O}(cE) + |Y|/2 \leq \frac{n}{4} + \mathcal{O}(n^{19/20})$ w.v.h.p.

The total number of comparisons thus used by this subroutine is $n + \mathcal{O}(n^{19/20})$ w.v.h.p.

2.3.5 OneCandidate

This subroutine begins by comparing the frequent element passed to it from the main algorithm with every element in the overall set. Since we are in this case, we know that the frequency of the frequent element in the sample, q_1, is at least $\frac{1}{2} - b\epsilon$, so by Lemma 2.3.1 we can say w.v.h.p. that its frequency in the overall set, p_1, is at least $\frac{1}{2} - b'\epsilon$ with $b' = b - 1$. If $p_1 > \frac{1}{2}$, then we terminate on line 10 using only n comparisons, so for the rest of this case we can assume that $p_1 \in [\frac{1}{2} - b'\epsilon, \frac{1}{2}]$, and thus the goal is to show that there does not exist a majority. That is, this frequent element must have narrowly missed being a majority element, and since it was the only element in the sample frequent enough to be a contender for the majority, it must mean that there is no majority w.v.h.p.

Our algorithm has already compared every element to α, so it must spend no more than $\mathcal{O}(n^{19/20})$ comparisons to determine that there cannot be another majority element. Note that if both X and Z are smaller than $n/2$, then there cannot exist a majority; furthermore, all instances of any majority element must be in exactly one of these sets. We call the larger of these sets A, and we use k to refer to the number of elements in A above $n/2$; i.e. $k = |A| - n/2$.

Note that w.v.h.p., k is no more than $b'E$ based on the reasoning about q_1 above. Because we did not end up in the TwoCandidates case, we have that $q_2 < \frac{1}{2} - c\epsilon$, so w.v.h.p. $p_2 n < \frac{n}{2} - c'E$ with $c' = c - 1$. We have already spent n comparisons, but here we argue that the number of extra comparisons spent to disprove a majority is $\mathcal{O}(n^{19/20})$. If we find k heterogeneous pairs in A, that is enough to disprove a majority, so we shuffle A and draw pairs until we meet this condition. Drawing a pair and determining if it is

heterogeneous requires one comparison, so we count the number of pairs we expect to draw in order to find k heterogeneous ones.

Let us consider a partition of A: $\{A_1, A_2\}$ with the requirement that for any value, all balls with that value are in the same part. If we have a pair of elements that comprises one element from A_1 and one element from A_2, then this will necessarily be a heterogeneous pair. Thus, we can lower bound the number of heterogeneous pairs in A drawn at random by the number of pairs of elements where one came from A_1 and the other from A_2. Intuitively, we will get the strongest lower bound on the number of heterogeneous pairs by picking a partition whose components are as close to equal in size as possible. With very high probability, we can say that the number of instances of the most frequent element in A (which corresponds to the second most frequent element in S) is $0.5n - c'E$ where $c' = c - 1$ (by Lemma 2.3.1). We will place all instances of this element in A_1. Thus, the largest A_1 we might require has cardinality $0.5n - c'E$. In this case, $|A_2| = \frac{n}{2} + k - |A_1| = c'E + k$.

We now want to argue that this bound is sufficient such that when we draw some number of pairs $d \in \mathcal{O}(n^{19/20})$ we find at least $b'E$ heterogeneous pairs, which disproves the existence of a majority in S. By Lemma 2.3.3, we will find at least $b'E$ heterogeneous pairs after drawing only $d = \frac{4(|X|-1)(|X|)b'E}{|X_1||X_2|}$ pairs from A with probability at least $1 - e^{-\frac{|X_1||X_2|b'E}{|X|(|X|-1)}}$. Since $\frac{|X_1||X_2|b'E}{|X|(|X|-1)} \in \Omega(\log^2 n)$, this occurs with very high probability. Each pair drawn uses exactly one comparison, so this subroutine thus disproves a majority using $n + d$ comparisons w.v.h.p. Since $d \in \mathcal{O}(n^{19/20})$, that means this subroutine terminates using $n + \mathcal{O}(n^{19/20})$ comparisons w.v.h.p.

2.3.6 NoCandidates

This subroutine starts by comparing every element to the median of the sample, m. From that, it forms a family of sets X, Y, and Z (elements less than, equal to, and greater than m, respectively) that partition the input. Forming the X, Y, and Z subsets clearly takes exactly n comparisons. If none of the subsets have cardinality greater than $\frac{n}{2}$, or if $|Y|$ has such cardinality, then the algorithm terminates there and spends only n comparisons total.

If X or Z have more than $\frac{n}{2}$ elements, then the algorithm has to spend more comparisons, but we argue here that the number of extra comparisons is bounded above by $\mathcal{O}(n^{19/20})$ w.v.h.p. We use A to refer to the larger of X or Z. Let us say that set A has cardinality $|A| = \frac{n}{2} + k$. With very high probability, $k \leq E$ because by Lemma 2.3.2 the rank of the median of S' is expected to not be farther than E off from the median of S. To disprove a majority in S, it suffices to find k heterogeneous pairs in A.

The algorithm continues by drawing pairs from A until it has disproven a majority. Similar to the previous section, we will consider a partition of A: $\{A_1, A_2\}$ with the requirement that for any value, all balls with that value are in the same part. We will again lower bound the number of heterogeneous pairs by the number of (A_1, A_2) pairs.

Since we ended up in this subroutine, that must mean $q_1 < \frac{1}{2} - b\epsilon$ by Lemma 2.3.1. Note also that this implies that $\forall i \in [1, n], q_i < \frac{1}{2} - b\epsilon$. With very high probability, we can say that the number of instances of the most frequent element in A is $0.5n - b'E$ where $b' = b - 1$. We will place all instances of this element in A_1. Thus, the largest A_1 we might require has cardinality $0.5n - b'E$. In this case, $|A_2| = \frac{n}{2} + k - |A_1| = b'E + k$.

By Lemma 2.3.3, we will find at least E heterogeneous pairs after drawing only $d = \frac{4(|A|-1)(|A|)E}{|A_1||A_2|}$ pairs w.v.h.p. Each pair drawing uses exactly one comparison, so this subroutine disproves a majority using $n + d$ comparisons w.v.h.p. Since $d \in \mathcal{O}(n^{19/20})$, that means this subroutine terminates using $n + \mathcal{O}(n^{19/20})$ comparisons w.v.h.p. □

Chapter 3

Conclusion and Future Work

In this thesis, we presented the first Las Vegas-style randomized algorithm that correctly solves the majority problem under an ordered comparison model using just $n + o(n)$ comparisons in the expected case, and in fact, does so with very high probability.

3.0.1 Optimality

We conjecture that any Las Vegas-style randomized algorithm that correctly solves the majority problem under an ordered comparison model must use at least $n - o(n)$ comparisons in the expectation, which would imply that our result is optimal to a lower order term. We attempted to prove this from the lower bound proof of Gawrychowski et al. [7], but unfortunately their proof technique could not be easily adapted to show a similar result in the ordered comparison model. Their proof technique relied on upper bounding the amount of knowledge that any correct algorithm $\mathcal{A}$ could know at a given stage and showing that in order for $\mathcal{A}$ to declare a majority, it must have made at least a certain number of comparisons. Unfortunately, their technique did not immediately extend to the ordered comparison model because ordered comparisons have transitivity, which violated some of the assumptions of the proof. We think that proving this conjecture is a promising avenue for future work.

3.0.2 Optimizing Lower-Order Term

In this work, we did not make any effort to produce an algorithm with the smallest lower-order term. Our focus was to produce an algorithm with an $n + o(n)$ expected number of

comparisons, as opposed to finding the smallest lower-order term. We are confident that there exist Las Vegas algorithms for the ordered majority problem that correctly solve the problem in $n + \gamma(n)$ comparisons where $\gamma(n) \in \Omega(1) \cap o(n^{19/20})$, and so it could be an avenue of future work to minimize this term.

3.0.3 Comparison Systems

Another possible goal for future work is to study this problem using 2-way comparison models that are not equality-test, such as a model where a comparison tells you $a \leq b$ or $a > b$. Alternatively, an interesting model through which to study this problem could be one that allows any type of 2-way comparison, so one could perform any combination of $\{=, \neq\}$ comparisons or $\{<, \geq\}$ comparisons. We are not aware of any work on this problem using models of either of these types.

References

[1] Robert S. Boyer and J. Strother Moore. *MJRTY—A Fast Majority Vote Algorithm*. Springer Netherlands, Dordrecht, 1991.

[2] Marek Chrobak, Mordecai Golin, J. Munro, and Neal Young. Optimal search trees with equality tests. *ArXiv*, abs/1505.00357, 05 2015.

[3] Walter Cunto and J. Ian Munro. Average case selection. *J. ACM*, 36(2):270–279, 1989.

[4] Devdatt P. Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *BRICS Report Series*, 3(25), Jan. 1996.

[5] M.J. Fischer and Steven Salzberg. Finding a majority among n votes: Solution to problem 81-5. *Journal of Algorithms*, 3, 01 1982.

[6] Robert W. Floyd and Ronald L. Rivest. Algorithm 489: The algorithm select—for finding the i-th smallest of n elements. *Communications of the ACM*, 18(3):173, Mar 1975.

[7] Pawel Gawrychowski, Jukka Suomela, and Przemyslaw Uznanski. Randomized Algorithms for Finding a Majority Element. In Rasmus Pagh, editor, *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[8] E. N. Gilbert and E. F. Moore. Variable-length binary encodings. *The Bell System Technical Journal*, 38(4):933–967, 1959.

[9] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[10] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, England, August 1995.

[11] Michael O Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[12] Siwei Yang. The randomized majority problem. Unpublished thesis draft, University of Waterloo.

Glossary

comparison-based algorithm an algorithm where the algorithm can only gain information about the input through comparing two elements. 1

equality-test comparison model a comparison model where each comparison between two elements returns only whether or not they are equal. 2, 4

heterogeneous pair a pair of two elements that are known to not be equal. 5, 15, 18, 23, 24, 26

homogeneous pair a pair of two elements that are known to be equal. 5, 15, 23

Las Vegas algorithm a class of randomized algorithms that always produces a correct result but whose running time is a random variable that depends on the input, though the expectation of the running time must be finite. 3, 8, 22

majority problem the problem of determining, among an input of n values, whether there exists one value whose frequency is more than $\lfloor n/2 \rfloor$, and if so, producing such a value and the number of times it occurs. 4

Monte Carlo algorithm a class of randomized algorithms that is allowed to err with a certain (usually small) probability. 3

ordered comparison model a comparison model where each comparison between two elements returns whether or not they are equal, and if they are not, it returns their relative ordering. 2, 4, 10

randomized algorithm an algorithm that employs randomness in some part of its process. 3, 8

very high probability the probability of an event is at least $1 - \exp(-\Omega(\log^2 n))$. 19