CS 3EA3: Optional Assignment - A Survey of Binary Search

James Zhu 001317457

26 April 2017

1 Abstract

This paper presents a general survey of the functional, syntactic and logistical aspects of two variants of the Binary Search Algorithm— one "traditional" variant as introduced by Sedgewick and Wayne in [1], and the other by Musa Al-hassy in [3]. Specifically, a brief overview of the major steps in the derivation of each algorithm will be discussed, followed by an analysis of the structure, operations and implications inherent in each within the context of efficiency both execution and implementation. Lastly, a conclusion will be made as to which algorithm can be considered to be *better* within the scope of these considerations.

2 Introduction

In this paper, a comparison of a instance of the *Binary Search Algorithm* as presented by Sedgewick and Wayne in [1], and the general variant as presented in the course material of the CS 3EA3: Software Specifications and Correctness will be made. Specifically, a brief description of the operation of both variants will be provided, and an analysis of their respective structure and operations on the input data will be conducted. Additionally, the run-time and implementation efficiency of both variations will be dissected, and use cases for both established. Lastly, the ease of memorisation and application of both will be evaluated, and the context for the utilisation for each variant will be outlined.

3 What is Binary Search?

Two varying descriptions and implementations of "Binary Search" are described below, (i) one presented by *Sedgewick and Wayne* in [1], and (ii) the other by Musa Al-hassy within the course material of the CS 3EA3 Course.

3.1 Binary Search as Proposed by Sedgewick and Wayne

Given a list of elements contained in a list which contains a `Comparable` key and an associated value, provided with some desired key, *Binary Search* will return the index of the key within the list if it is present, and `-1` otherwise. [1] Specifically, a key is said to be `Comparable` if each of the elements belonging to the same class has a *natural ordering* which is imposed on it. [2] This allows for the definition of the notion of a comparison between objects — in this case, keys — by some *natural comparison method.*

Additionally, elements are stipulated to be *sorted* according to their respective natural ordering.

To perform the search for a specified element, two variables `lo` and `hi` are maintained, indicating the current *low* and *high* indices of the search bound, respectively. The *lo* value is set prior to the beginning of the search process to the *lowest index* in the list, and the *hi* value to the *highest.* Then, the algorithm continuously makes *comparisons*, while `lo` <= `hi`, utilising the natural comparison method, between the key of the desired element, and that of the *median element* `mid` within the interval of the list bounded by `lo` and `hi`.

The result of this comparison may be one of **three** cases:

- The *key* of the desired element is ***smaller*** than that of the `mid` element. In this case, the algorithm will *cut* the interval size in (approximately half), by setting the value of `hi` to the index value which is *one below* that of `mid`. The search then continues in the same manner described above within this interval.
- The *key* of the desired element is ***larger*** than that of the `mid` element. In this case, the algorithm will *cut* the interval size in (approximately) half, by setting the value of `lo` to the index value which is *one above* that of `mid`. The search then continues in the same manner described above within this interval.
- The *key* of the desired element is ***exactly equal to*** than that of the `mid` element. In this case, the `mid` element is the desired element, and the value of `mid` is returned, thus indicating the *index value* of the *searched-for element within the list.* The algorithm then terminates.

At the termination of this *while loop*, should no index be found, then a `-1` value is returned, indicating that the desired element was *not found* in the list.

3.2 Binary Search as Proposed by Musa Al-hassy

The ***General Binary Search*** algorithm presented by Musa Al-Hassy stipulates that:

Given: $a < b$, such that a $\boldsymbol{Z}$ b, where $\boldsymbol{Z}$ is some ***co-transitive relation***, such that $\forall$ x, y, m : *integers*, x $\boldsymbol{Z}$ m $\vee$ m $\boldsymbol{Z}$ y $\Leftarrow$ x $\boldsymbol{Z}$ y, to find:

Required: two neighbours x and *x+1*, such that it is *bounded* by the interval set by a and b from above: $a \leq x < b$. [3].

The proposed algorithm firstly assigns an `x` and `y` to be equal to the *lower* and *upper* bound of the interval. Then, while ensuring the *invariant* $a \leq x < y \leq b \wedge x\mathbf{Z}y$ holds, ***while*** $x + 1 \neq y$***, a median*** `m` ***is found***. Then, as this implies that $x < m < y$, there exists *two cases*:

- If **m Z y** then assign `x` to `m`
- If **x Z m** then assign `y` to `m`

The above two cases has the effect at each iteration of either:

- Moving the ***lower bound*** in the *search interval* to be the **median**; *or*
- Moving the ***upper bound*** in the *search interval* to be the **median**.

This thus cuts the searched interval by (approximately) *half* at each iteration. This `while loop` continues to run until `x` becomes *one less than y*. Then, ***x is returned***. Note that the input data *does not have to be ordered* according to its specified natural ordering.

4 A Comparison of Structure and Operations on Input Data

In terms of the *structure* and *operations on input data* performed by both variants of Binary Search, it may be observed that:

1. **The (negation of) the termination condition of the *while* loop is expressed differently.** For the *traditional version* proposed by Sedgewick and Wayne, this condition is expressed as `while[lo <= hi]`, while for the version General Binary Search implementation, it is: `do x + 1` $\neq$ `y`. This is due to the fact that as the former version always reduced the search interval to a range that is bounded by, *but not including* ***the median***, the case where the *lower bound is* ***equal to*** *— but not higher than the upper bound* must be considered. Contrastingly, for the General Binary Search presented by Musa Al-hassy, as any shrinkage in the interval, while also bounded by the median, *also includes the latter*. Structurally, this allows the algorithm to *terminate* when `x == y`, rather than *one above* with `lo > hi`. Logistically, this also decreases the complexity of the information which must be recalled during implementation, as with regards to the *divider* of the two different *partitions* within the Binary Search instance,

one does not have to recall that the range must be either beginning one entry *before* or *after* the median, as is the case with the instance proposed by Sedgewick and Wayne.

2. **The calculation of the median varies in complexity.** As the General Algorithm as introduced by al-Hassy in [3] utilises an invariant imposed on the lower and upper bounds of the search interval, to ensure that the *the median will always* ***remain valid*** *within this range*. Namely, the invariant `x < m < y` must always hold in each iteration of the `while loop`, prior to the evaluation of any guards. However, as no such invariant is implemented in the version as proposed in [1], an extra *addition* of the value of the index of the *lower bound* must also be made: `mid = lo + [hi - lo]`.

5 A Comparison on the Time and Space Efficiency

It can be observed that in *both* variants of Binary Search, *the search interval is cut approximately by* ***half*** *at every iteration.* Thus, the run-time is **O [log n]**. Assuming, however, that the input data or list is *unsorted*, then it must be done first, thus, necessitating a run-time which is more analogous to **O [n log n]** in this latter case.

This sorting is *required*, as only with ***monotonicity*** introduced by the natural ordering of the elements can it be guaranteed that the *median element which is found through the above method actually represents the median value within the search interval*. The latter fact must be true in order for the Binary Search algorithm to be certain that the desired element is within *either* the *lower* or the *higher* side of the partition. However, with the General Binary Search as presented by Al-hassy, monotonicity is has already been assured through the invariant stated above: $x < m < y$.

6 Debunking the Myth: `return` is better!

Whilst the argument might be put forth that the Traditional variant of Binary Search (as exemplified by Sedgewick and Wayne) may be superior due to the presence of `return` statements which allow the algorithm to end early should the desired element be found (eg. in the case that `key = mid`), it can be shown that this mechanism only goes to introduce extra guards (and therefore complexity), which must be evaluated in each iteration, and does not introduce any *functional* variance in the output.

In the version proposed by Sedgewick and Wayne, it may be seen that upon the satisifiability of `key = mid`), the index of the searched for element is re-

turned, al-Hassy's implementation already *implicitly* accommodates for this, once again through the invariant `x < m < y`, which again, *must be satisfied* before the evaluation of any guard. Otherwise, the loop will terminate in a very short period of time and the algorithm will arrive at the desired *post-condition*. This thus again *lowers* the amount of factors which must be *explicitly defined and accounted for* in implementation, and in extension, makes for a more *concise and elegant* specification.

7 Conclusion

Thus, it can be said that whilst the differences between the two specifications of the Binary Search algorithm are relatively *small*, when the ease of memorisation, derivation and implementation are taken into consideration, then it is very clear from the above comparisons that the **General Binary Search** as proposed by *Musa Al-Hassy* in [3] is far more mathematically concise — both in a syntactic and semantic context.

References

[1] R. Sedgewick, K. Wayne *Algorithms Fourth Edition*. Boston, United States: Addison-Wesley, 2011.

[2] Oracle, "Interface Comparable," [Online document], Available: https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

[3] M. Al-hassy, Class Lecture, Topic: "Binary Search - Revisited" CS 3EA3, Department of Computing and Software, McMaster University, Hamilton, Canada, 8 March 2017.