A sometimes-accurate O(1) search algorithm

A College Freshman, Quite Useful Institute of Technology (QUIT), Somewhere, USA

***Abstract* – Search algorithms are used to find things. That's what they're for, it's kinda in the name. The computer has to do work to find the things with search algorithms, and we want the computer to do less work. We thought of a way the computer could do a very small amount of work. Unfortunately this also makes the computer very bad at the searching part. Fortunately, what the algorithm lacks in accuracy and general search functionality, it makes up for in speed and simplicity.**

I. INTRODUCTION

Do I really have to explain search algorithms again? It's not like anyone reading this doesn't know what they are. Fine.

Search algorithms are when the computer tries to find a specific value inside of another thing. Like an array or something like that. For an unsorted array, the best search algorithm is a linear search, which has to step through every element of the array, and, shockingly, runs in linear time $O(n)$.

Our method strives to improve this algorithm by sacrificing just a bit of functionality to obtain constant-time searching, regardless of the size of the inputted container.

II. THE (VERY COMPLICATED) ALGORITHM

Running a search algorithm in O(1) time might seem logically impossible at first glance. But we've put a lot of thought into this problem, and determined it is in fact possible, even easy. The reason why the task seemed impossible before is because computer scientists are too concerned with the search algorithm being actually functional. How foolish.

We've devised a technique that bypasses any former limitations of search algorithms, such as the need to pick through multiple elements of the container, the need for the algorithm to result in the correct answer, or even an answer at all.

In contrast to other search algorithms, the algorithm devised in this paper is incredibly simple; rather than "search" through the array, it simply checks if the first element is the target. If it is, the algorithm returns that spot in the array, otherwise, reports that the item is not found.

Now, you might be thinking, "Wait, this will only work for length-1 arrays, or if the element is in the first index. How is this even a search algorithm? Of course it runs in O(1) time, you–" and that's all you'll get out before I hit you with a baseball bat.[1]

III. METHODOLOGY

At first, the implementation of this was difficult, taking many arduous hours and hundreds of attempts to get the functionality working. While initial implementations measured hundreds of lines long, our final working implementation was able to be greatly simplified. The C implementation is shown below:

```c
int search(int* array, int target) {
    int index = 0;
    int searching = array[index];
    int difference = searching - target;

    if (difference != 0)
        return -1;
    else
        return index;
}

```

Note that despite the input array type being int, this algorithm works with any datatype, and the concepts discussed in this paper can easily be applied to a variety of data structures, including Trees, Heaps, Piles[2], and Sand dunes.

IV. RESULTS

Array Size	Time (ns)
1	0
5	0
10	0
10^2	0
10^3	0
10^4	0
10^5	0
10^6	0

TABLE I
Showing the speed of the algorithm, run on an 8-core AMD Ryzen CPU. As you can see, it's very fast, and the time does not increase with the size of the array. Or my code was broken, which is also likely.

V. CONCLUSIONS

The algorithm works, and runs in constant time; when called with arrays of various sizes according to **TABLE 1**, the search time stayed the same. Whether or not the result was correct is irrelevant, because it was fast.

Of course, this approach does have some drawbacks; namely if, for some reason, you *do* want your searches to actually "work". Future research should look into applying this technique in ways that will lead to a higher success rate, such as checking the first two elements, or perhaps even the first three.

ACKNOWLEDGEMENTS

I would like to thank some people for supporting me in the arduous writing of this article, some other people for making me aware SIGBOVIK exists, and I would also like to thank this very serious conference for existing.

[1] Don't worry, it won't hurt. I'm a CS student.

[2] So apparently piles are a real data structure. Huh. I would like to reiterate that as a first-year undergraduate student, I know nothing.