GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Ask a Q
- About

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

A Problem in Many Binary Search Implementations

Consider the following C implementation of Binary Search function, is there anything wrong in this?

```
// A iterative binary search function. It returns location of x in
// given array arr[l..r] if present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    while (l <= r)
    {
        // find index of middle element
        int m = (l+r)/2;

        // Check if x is present at mid
```

```
        if (arr[m] == x) return m;

        // If x greater, ignore left half
        if (arr[m] < x) l = m + 1;

        // If x is smaller, ignore right half
        else r = m - 1;
    }

    // if we reach here, then element was not present
    return -1;
}
```

The above looks fine except one subtle thing, the expression "m = (l+r)/2". It fails for large values of l and r. Specifically, it fails if the sum of low and high is greater than the maximum positive int value (2^{31} – 1). The sum overflows to a negative value, and the value stays negative when divided by two. In C this causes an array index out of bounds with unpredictable results.

What is the way to resolve this problem?
Following is one way:

```
int mid = low + ((high - low) / 2);
```

Probably faster, and arguably as clear is (works only in Java, refer this):

```
int mid = (low + high) >>> 1;
```

In C and C++ (where you don't have the >>> operator), you can do this:

```
mid = ((unsigned int)low + (unsigned int)high)) >> 1
```

The similar problem appears in Merge Sort as well.

The above content is taken from google reasearch blog.

Please refer this as well, it points out that the above solutions may not always work.

The above problem occurs when array length is 2^{30} or greater and the search repeatedly moves to second half of the array. This much size of array is not likely to appear most of the time. For example, when we try the below program with 32 bit Code Blocks compiler, we get compiler error.

```
int main()
{
    int arr[1<<30];
    return 0;
}
```

Output:

```
error: size of array 'arr' is too large
```

Even when we try boolean array, the program compiles fine, but crashes when run in Windows 7.0 and Code Blocks 32 bit compiler

```
#include <stdbool.h>
int main()
{
    bool arr[1<<30];
    return 0;
}
```

Output: No compiler error, but crashes at run time.

Sources:
http://googleresearch.blogspot.in/2006/06/extra-extra-read-all-about-it-nearly.html
http://locklessinc.com/articles/binary_search/

This article is contributed by **Abhay Rathi**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- An interesting time complexity question
- Analysis of Algorithms | Set 4 (Analysis of Loops)
- Analysis of Algorithms | Set 3 (Asymptotic Notations)
- NP-Completeness | Set 1 (Introduction)
- Static and Dynamic Libraries | Set 1
- The Ubiquitous Binary Search | Set 1
- Reservoir Sampling
- Analysis of Algorithms | Set 2 (Worst, Average and Best Cases)

Writing code in comment? Please use **ideone.com** and share the link here.

Ram • 3 months ago
You don't really have to use >>> in java, since we are dealing with only positive numbers here. So, you can safely use >> instead of >>>
• Reply • Share ›

Guest • 5 months ago
After the C implementation of Binary Search , the paragraph immediately after the code... in the line 3 it is written that "The sum overflows to a negative value, and the value stays negative when divided by two" ... y does the sum overflows to a negative number?
• Reply • Share ›

Ashutosh Mittal Guest • a month ago

because first addition is done and it might go beyond the range of a data type (say double) , then sum will give us a negative number and dividing it by 2 will also be negative

• Reply • Share ›

Kim Jong-il • 8 months ago

Love this one.!!

• Reply • Share ›

typing.. • 9 months ago

what about (l/2 + r/2) ???

1 • Reply • Share ›

jeeper typing.. • 9 months ago

wrong: 3/2+3/2 != (3+3)/2

4 • Reply • Share ›

Jerry Goyal jeeper • 9 days ago

get your maths straight....3/2 =1.5 and 2x1.5 = 3.. equation is valid. but the problem is r/2 will type cast the result into integer i.e.1 instead of 1.5. solution for this problem is to use typecasting "int m=(float)l/2+(float)r/2;"

• Reply • Share ›

Corey Yanofsky jeeper • 8 months ago

l/2 + r/2 + ((l % 2) + (r % 2))/2

1 • Reply • Share ›

mike0809 • 9 months ago

This one is the preferable solution imo:

int mid = low + ((high - low) / 2);

Picking low, mid, and high as int, that is as signed, is basically limiting the function where it is entirely unnecessay. Neither of these values will ever be negative, so they should be unsigned from the start, if your language allows for it. If chosen unsigned this just moves the problem to the next bit, but also, in the case of unsigned, makes it worse: Now, if you want to do:

((cast_up) low + (cast_up)high >> 1);

You will have to used the next bigger storage unit, which will be an uint64_t, which will require additional memory allocation to hold the values and they

will have to be copied into this location.
You can certainly replace the "/2" part with >> 1, but using the cast on the addition is - imo - very likely slower than using the solution I referred to as the preferred one.

Stack space is limited (usually to something like 8MB or something of comparable magnitude), so you will not be allowed to allocate an array of this size on the stack, but it should be no problem on the heap:

Runtime crash: stack space is limited (~8MB per process usually I think); allocate on the heap.
1 • Reply • Share ›

Guest • 9 months ago
I don't think that (low + high)/2 matters this is because you cannot declare an array of that much size, in general array of size of order 10^8 is allowed in "global scope" so if you talk about the general case it will never overflow.
• Reply • Share ›

v3gA Guest • 9 months ago
It can be an array in the heap. An int array of such size would require 8 GB of space (2**31 * 4 bytes) though, but it's still possible and a good precaution to take.
• Reply • Share ›

Kartik Guest • 9 months ago
it may matter for unbounded binary search, refer http://www.geeksforgeeks.org/f...
1 • Reply • Share ›

- Interview Experiences
- Advanced Data Structures

- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 1 hour ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 1 hour ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 1 hour ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

  @GeeksforGeeks i don't n know what is this long...

  Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 2 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 2 hours ago