Binary Search of an Integer Array

By vegaseat on Jan 25th, 2005 9:56 am 0

Folks have asked numerous times for a code snippet of a binary search of an array. Here is heavily commented code with a few test printf() included to give you a picture of what is going on.

Related Article: Behaviour of Binary Search Program

Authored by Thinka in the C forum, this related discussion thread currently has 13 replies and was last posted to 5 years ago by Nick Evan. It begins, "Hello once again people; I hope I'm not breaking any rules or being cheeky. But ..."


```
1.  // binary search of an integer array, this search is efficient for large arrays
2.  // tested with PellesC       vegaseat     24jan2005
3.
4.  #include <stdio.h>
5.
6.  int main()
7.  {
8.    int a[20] = {0};
9.    int n, i, j, temp;
10.   int *beg, *end, *mid, target;
11.
12.   printf(" enter the total integers you want to enter (make it less then 20):\n");
13.   scanf("%d", &n);
14.   if (n >= 20) return 0;   // ouch!
15.   printf(" enter the integer array elements:\n" );
16.   for(i = 0; i < n; i++)
17.   {
18.     scanf("%d", &a[i]);
19.   }
20.
21.   // sort the loaded array, a must for binary search!
22.   // you can apply qsort or other algorithms here
23.   for(i = 0; i < n-1; i++)
24.   {
25.     for(j = 0; j < n-i-1; j++)
```

```
    {
      if (a[j+1] < a[j])
      {
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp;
      }
    }
  }
  printf(" the sorted numbers are:");
  for(i = 0; i < n; i++)
  {
    printf("%d ", a[i]);
  }

  // point to beginning and end of the array
  beg = &a[0];
  end = &a[n];  // use n = one element past the loaded array!
  printf("\n beg points to address %d and end to %d",beg, end);  // test

  // mid should point somewhere in the middle of these addresses
  mid = beg += n/2;
  printf("\n mid points to address %d", mid);  // test

  printf("\n enter the number to be searched:");
  scanf("%d",&target);

  // binary search, there is an AND in the middle of while()!!!
  while((beg <= end) && (*mid != target))
  {
    // is the target in lower or upper half?
          if (target < *mid)
          {
      end = mid - 1;     // new end
      n = n/2;
      mid = beg += n/2;  // new middle
    }
    else
    {
      beg = mid + 1;     // new beginning
      n = n/2;
      mid = beg += n/2;  // new middle
    }
  }

  // did you find the target?
  if (*mid == target)
  {
    printf("\n %d found!", target);
  }
  else
  {
    printf("\n %d not found!", target);
  }

  getchar();  // trap enter
  getchar();  // wait
  return 0;
```

```
84.  }
85.
```


samrprog
New bie Poster
1 post since Jul 2009

3 Years Ago — **0**

```
1.  Thanks a lot. This program was a real help.
```

xhongyi
New bie Poster

2 Years Ago — **0**

If beg and end are both real address for a[i], which means they are multiple of 4, should the following calculation revised to mid=beg+=2n?
(The size of int is 4)

xhongyi
New bie Poster
2 posts since May 2010

2 Years Ago — **0**

No there is no problem, sorry guys.

Adak
Nearly a Posting Virtuoso
1,480 posts since Jun 2008

2 Years Ago — **0**

The compiler will handle that, for any data type, automatically.

It's a bit awkward because you're working with mid and other variables, as pointers, instead of as an index.

Also, you repeat the calculation and assignment line of code for mid, twice. Move it to the top of the while loop, and you can just do it one time.

It appears to be good workable code. For a snippet, I would have hoped for a sort better than bubble sort, however. Even though it's an optimized bubble sort, it's still a big step down.

It's ironic that we get way faster computers, only to have the slowest sorter, re-emerge in popularity.

kele matshego
New bie Poster
1 post since Aug 2010

2 Years Ago — **0**

use the following numbers and explain how a binary search works.we want to know if 7 are in this list of items.show all the steps
34 28 4 16 9 8 35 27 7

Adak
Nearly a Posting Virtuoso

2 Years Ago — **0**

1,480 posts
since Jun 2008

A binary search just takes a guess on the answer, and if the guess is too high, it cuts down the top of the search to one element's value below the guess. If the guess is too low, it brings up the bottom value of the search, to one element's value, above your guess.

So every time you guess and miss, you cut the search space into an ever smaller search space. Indeed, on each guess it makes, since it guesses the middle number (and unless that guess is correct), the search space is cut in half! ;)

In order to work, the values being searched, must be sorted. Since your numbers aren't sorted yet, that would be the necessary first step.

Instead of posting your assignment, post your assignment AND your attempt to solve it, AND what has you stumped, next time.

And Welcome to the Forum! ;)

surbhi11
New bie Poster

1 post
since Mar 2011

1 Year Ago **0**

Hello
I have an assignment regarding binary search
we have to search for an employee and his/her details from an array of structures
The above program really helps
But i am not getting why in "if(*mid!=target && beg<=end) "....beg<=end condition is required
I ran the program without it but nothing changed

WaltP
Posting Sage w / dash of thyme

Moderator

11,141 posts
since May 2006

1 Year Ago **0**

Maybe you should read this Code Snippet then, as well as the posts.
Follow the code using pencil and paper to understand code in general.

Phil Outram
New bie Poster

3 posts
since Aug 2011

1 Year Ago **0**

Can someone please explain to me why we don't hit memory past the top end of the array under some circumstances with this algorithm?

In a very simple example with an array of 2 numbers, say 5 and 10, and we're looking for a target of 20, which we shouldn't find as it's too big.

First time through beg points at 5, mid points at 10 and end points off the end of the array which is fine, although I'm not sure why we want this? Why not set it to n - 1 so it's at least pointing to the last element in the array.

So, we do the compare and see that target > *mid so we execute beg = mid + 1; but this now means that beg is pointing off the end of the array. And then mid = beg += n/2 sets mid off the end of the array too! So now all three pointers point outside the array. Or am I missing something?

Would just setting beg = mid; fix the problem for a tiny loss of performance?

Any thoughts would be much appreciated.

Thanks.

Phil Outram
New bie Poster

1 Year Ago **0**

3 posts since Aug 2011

Apologies, while problem still stands, I didn't think my suggested solution through - we have to move one of beg or end each time through the loop otherwise we can get into infinite loops. My suggestion fails with example I've just given if the target is 7.

Re-thinking, would setting n = n-1 at the top work? i.e. just before end = &a[n];

Phil Outram
New bie Poster

3 posts since Aug 2011

1 Year Ago **0**

Hmm, on further inspection, the original algorithm also doesn't handle the case where n = 0.

Collecting a few ideas from a few other sites and doing a bit of testing, here's my preferred solution for anyone who's interested (it is slightly more explicit with less pointer shifting, but equally quick or quicker when compiled in the random tests I ran)...

```
1.  int* pBase = &a[0];
2.      int nIdxLeft   = 0;
3.      int nIdxRight  = GetSize() - 1;
4.      int nIdxMiddle = 0;
5.      int nMiddleValue = *pBase;
6.
7.      while (nIdxLeft <= nIdxRight)
8.      {
9.          nIdxMiddle = (nIdxLeft + nIdxRight) / 2;
10.         nMiddleValue = *(pBase + nIdxMiddle);
11.         if (nMiddleValue == dwTarget)
12.             break;
13.         else if (nMiddleValue > dwTarget)
14.             nIdxRight = nIdxMiddle - 1;
15.         else
16.             nIdxLeft = nIdxMiddle + 1;
17.     }
```

Adak
Nearly a Posting Virtuoso

1,480 posts since Jun 2008

1 Year Ago **0**

These "snippets" are not checked by anyone beyond a cursory look. They're typically coded by students, and shouldn't be taken as examples of great functions, in detail. Indeed, some have been dreadful, and I've shown it in detail.

In this case, you can tell it's less than elegant, simply because it repeats lines of code. It's less than efficient, because it repeats the comparison to target twice, in every loop.

It's alright if the pointers run off the boundary of the array, (or the index does), as long as the array is not referenced at that time (or the pointer isn't dereferenced). The idea it seems, is that the while loop test will fail, before that illegal comparison will be made.

I would not use any of these snippets, without thoroughly checking it out and testing them. This one is obviously, poor code. It does get the general idea across. That has some merit. I can't give it high marks otherwise, however.

Randika prasad
New bie Poster

1 post since Oct 2011

10 Months Ago **0**

```
1.  // I think u can realize this method......
2.  #include<stdio.h>
3.  int sort(int arr[],int n) // Sorting the array..... here i use insertion sort but u can
```

```
 use any sorting method.....
{
        int i,j,temp;
        for(i=1;i<n;i++)
        {
                temp=arr[i];
                j=i-1;
                while(temp<arr[j] && j>=0)
                {
                        arr[j+1]=arr[j];
                        j=j-1;
                }
                arr[j+1]=temp;
        }
}
int get_data(int arr[],int n) // Gte data for array...
{
        int i;
        printf("Enter data for array :\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&arr[i]);
        }

}
int out_data(int arr[],int n) // Out data in the array....
{
        int i;
        printf("After sorting......\n\n");
        for(i=0;i<n;i++)
        {
                printf("%d ",arr[i]);
        }
}
int bi_search(int a[], int low, int high, int target) // Searching function......
{
    if (high < low)
        return -1;
    int middle = (low + high)/2;
    if (target < a[middle])
        return bi_search(a, low, middle-1, target);
    else if (target > a[middle])
        return bi_search(a, middle+1, high, target);
    else if (target == a[middle])
        return middle;
}
main()
{
        int n=10,arr[20],k,b;
        get_data(arr,n);
        sort(arr,n);
        out_data(arr,n);
        printf("\n\nEnter value to search :");
        scanf("%d",&b);
        k=bi_search(arr,0,n-1,b);
        if(k==-1)
        printf("-->%d is not found......!\n",b);
        else
```

```
61.          printf("\n-->%d is found......!\n\n",b);
62.  }
```

Adak
Nearly a Posting Virtuoso

1,480 posts since Jun 2008

10 Months Ago **0**

I'm always skeptical when I read code that has an index or pointer that runs out of the array bounds. The standard says you MAY be sure that it's OK, but only for 1 element AND you are never to work with the value at that address.

Code with no supporting tests, are another thing I'm cautious about. If you say that a program you post is good at something, I expect at least a little data from your testing, to support that conclusion.

There's a good deal of "hand waving and hot air" regarding programs, and I've read some great examples of it, by notable professors even. In the latest scam, the professor claimed a VERY fast prime number generator -and it was BLAZING fast. Only later did you learn it was so fast, partly because it HAD A LIST OF PRIME NUMBERS, EMBEDDED IN THE CODE! Oh for crying out loud!!

Another common example is someone claiming that their new sorting algorithm is "Faster than Quicksort". Sure, because Quicksort is an old algorithm, and the early versions weren't well studied yet. So you can easily beat the Quicksort versions from 20 years ago, but that's *NOT* beating a modern version of Quicksort!

Well, they get their names in the scholarly publications, and their tenure is quite secure, etc. Still, it's misleading at best.

I don't know anything about this binary search version. When I was a kid, we used to play a "Guess the number I'm thinking of" game, from time to time. My version of the Binary search reflects the way us kids learned to play the game, and it's plenty fast enough, even for a speed fanatic like me. When I see a version that is at least 5% faster, and has data to support it, I'll look into it.

Until then, I'm just reading an interesting post.

codinghavok
New bie Poster

1 post since Dec 2011

8 Months Ago **0**

A description of Binary Search and the algorithm can be found here:
http://codinghavok.wordpress.com/2011/12/31/basics-i-searching/

You

This article has been dead for over three months: Start a new discussion instead

Post: Markdown Syntax: Formatting Help

Bold	*Italic*	Code	Inline Code	Link	Quote	Heading	Sub-Heading	# List	• List	Undo	Redo

Reply to this Discussion (Alt+S)

2d 2d-array a algorithm and array arrays binary by C c++ char conversion convert copy count declaration error exchange fault file files fopen for fork() format fscanf function functions hashtable help how i if in input int integer line linked linkedlist linux list loop main malloc matrix merge multiple mysql number numbers of on order output parallel pointer pointers precedence problem process program programming read reading scanf screen search segmentation server socket sort sorting sscanf str strcat strcpy string strings struct structure structures studio synchronization text threads to tree txt typedef ubuntu using variable vertical visual while write zero 55

Home | About Us | Contact Us | Press Kit | **Advertising Opportunities**
RSS Feed | Terms of Service | Donate | **NYC Events**