Binary search

Searching is a very common operation in Computer Science. We search for a word in a dictionary. We google for some information. There is a book on sorting and searching.

How would you search in a dictionary if the words are not ordered?

If elements of our array are ordered, can we do a faster search? Can we start from the middle of the array? If the element in the middle of the array matches what we are looking for, search is over. Otherwise we can discard one half of the array based on the comparison and search in the other half. We have halved the problem size.

If we compare elements from left to right (or right to left) in a sequential manner, we require 1000 comparisons if there are 1000 elements in the worst case.

If there are 1000 elements in a sorted array, we require 11 comparisons in the worst case if we go to the middle of the section of the array each time and discard half of the remaining array. This way of searching is called binary search. Binary search works only on sorted array.

This is an iterative implementation of binary search – this searches for an element x in the array a from position l to position r.

We assume that the element is not found so far – make pos equal to -1.

We keep searching when pos is -1 and there are more elements in the section of the array. l <= r indicates that there are one or more elements in the section. l > r indicates that there are no more elements in the array section.

If the element is found at the mid point m, pos is updated.

Otherwise based on the relative values of a[m] and x, one half of the array is discarded. We change l to m + 1 or r to m – 1.

```
int bsearch(int a[], int l, int r, int x)
{
        int m;
        int pos = -1;
        // when the element could still exist and is not found do
        while(l <= r && pos == -1)
        {
                m = (l + r) / 2;
                if(a[m] == x)
```

```
		{
			pos = m;
		}
		else if(a[m] > x)
		{
			r = m - 1;
		}
		else
		{
			l = m + 1;
		}
	}
	return pos;
}
```

```
#include <stdio.h>
#include "1_search.h"

int main()
{
	int a[] = {10, 15, 25, 30, 40, 45, 60, 70, 75};
	int n = 9;
	printf("search : %d\n", bsearch(a, 0, n - 1, 30));
	printf("search : %d\n", bsearch(a, 0, n - 1, 65));


}
```

```
#ifndef SEARCH_H
#define SEARCH_H
int bsearch(int a[], int l, int r, int x);
#endif
```

```
#include <stdio.h>
#include "1_search.h"
#if 0
int bsearch(int a[], int l, int r, int x)
{
	int m;
	int pos = -1;
	// when the element could still exist and is not found do
	while(l <= r && pos == -1)
	{
		m = (l + r) / 2;
		if(a[m] == x)
		{
			pos = m;
		}
		else if(a[m] > x)
		{
			r = m - 1;
		}
		else
		{
			l = m + 1;
		}
	}
	return pos;
}
#endif

int bsearch(int a[], int l, int r, int x)
{
	int m = (l + r) / 2;
	if(l > r)
	{
		return -1;
	}
	else if(a[m] == x)
	{
		return m;
	}
	else if(a[m] > x)
	{
		return bsearch(a, l, m - 1, x);
	}
	else
	{
		return bsearch(a, m + 1, r, x);
	}
}
```

This next example shows binary search on an array of structures ordered based on some key and search is based on a predicate.

Observe the changes in the bsearch function from 1_event_array.c.

This parameter compare is used to compare the given element with the element in position m. This comparison decides whether the element is found or which portion of the array should be discarded.

```
int (*compare)(const event_t *lhs, const event_t *rhs)

int bsearch(event_array_t a, int l, int r, event_t x,
	int (*compare)(const event_t *lhs, const event_t *rhs))
{
	int m;
	int pos = -1;
	int res;
	while(pos == -1 && l <= r)
	{
		m = (l + r) / 2;
		res = compare(&a[m], &x);
		//printf("%d %d %d %d\n", l, m, r, res);
		if(res == 0)
		{
			pos = m;
		}
		else if(res > 0)
		{
			r = m - 1;
		}
		else
		{
			l = m + 1;
		}
	}
	return pos;
}


#include <stdio.h>
#include "1_struct.h"
#include "1_struct.h"
#include "1_event.h"
#include "1_event_array.h"
#include <string.h>

int compare_detail(const event_t* lhs, const event_t* rhs)
{
	return strcmp(lhs->detail, rhs->detail);
}
int main()
{
	event_array_t events;
	int n;
	scanf("%d", &n);
	read_event_array(events, n);
	disp_event_array(events, n);

	event_t e;
	printf("reading event : ");
	read_event(&e);
	disp_event(&e);

	int res = bsearch(events, 0, n - 1,  e, compare_detail);
	printf("res : %d\n", res);

	read_event(&e);
	res = bsearch(events, 0, n - 1,  e, compare_detail);
	printf("res : %d\n", res);
```

```
}
```

What should binary search return if the element is not found? We returned -1. Can we return an indication where it would have been if the element were found? Make the result to indicate that the element is not found. Make the magnitude such that it gives the position where the element would have been found. Think about it.