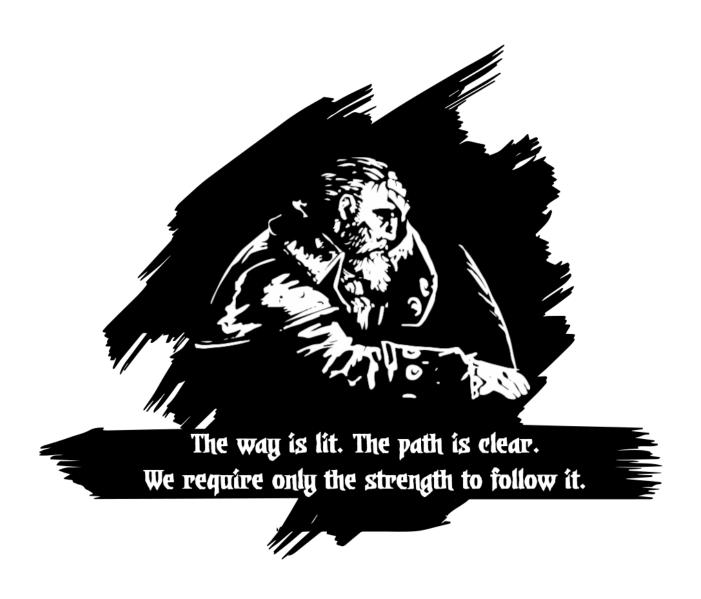


EXERCISES — Binary Search With Pointers

version #7be580532266ed398481e31366afcc24b1950c2a



Copyright

This document is for internal use at EPITA (website) only.

Copyright © 2022-2023 Assistants <assistants@tickets.assistants.epita.fr>

The use of this document must abide by the following rules:

- ▶ You downloaded it from the assistants' intranet.*
- ▶ This document is strictly personal and must **not** be passed onto someone else.
- ▶ Non-compliance with these rules can lead to severe sanctions.

Contents

1	Binary search	3
2	Goal	3
3	Examples	4

^{*}https://intra.assistants.epita.fr

File Tree

```
binary_search_ptr/
bsearch.c (to submit)
bsearch.h (to submit)
```

Authorized headers: You are only allowed to use the functions defined in the following headers

- err.h
- errno.h
- assert.h
- stddef.h

Compilation: Your code must compile with the following flags

-std=c99 -pedantic -Werror -Wall -Wextra -Wvla

Main function: None

1 Binary search

When looking for an element in a sorted array, it is possible to get the result with a logarithmic complexity (which means fast) using *dichotomy*. Here is a quick reminder of how it works:

A header (bsearch.h) containing all the required functions is provided on the intranet.

You **must** add this header to your submission, and you **must not** modify it.

2 Goal

```
int *binary_search(int *begin, int *end, int elt);
```

- begin and end will never be NULL.
- Search elt in the memory range of [begin, end[.
 - begin is a pointer to the first element.
 - end is a pointer **after** the last element.
- All the elements in the memory range of [begin, end[are guaranteed to be sorted in ascending order.

- The array does not contain any duplicate elements.
- An empty range is represented by begin == end.
- If elt is found, return a pointer to the element.
- If elt is not found, return a pointer to the memory location where elt should be inserted to keep the array sorted.
- If the array is empty, return begin.

3 Examples

```
int main(void)
{
    int a[] = { 0, 1, 4, 5, 9, 10, 18, 22, 42, 51, 69 };
    assert(binary_search(a, a + 11, 5) == a + 3);
    assert(binary_search(a, a + 11, 0) == a);
    assert(binary_search(a, a + 11, -1) == a);
    assert(binary_search(a, a + 11, 99) == a + 11);
    assert(binary_search(a, a + 11, 68) == a + 10);
    assert(binary_search(a, a, 5) == a);
}
```

The way is lit. The path is clear. We require only the strength to follow it.