0x1E. C - Search Algorithms

CAlgorithm

- By: Wilfried Hennuyer
- Weight: 1
- Project over - took place from Oct 3, 2022 6:00 AM to Oct 6, 2022 6:00 AM
- An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 0.0/17 mandatory & 0.0/34 optional
- **Altogether: 0.0%**
  - Mandatory: 0.0%
  - Optional: 0.0%
  - Calculation: 0.0% + (0.0% * 0.0%) == **0.0%**

Resources

Read or watch:

- Search algorithm
- Space complexity (1)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone, **without the help of Google**:

General

- What is a search algorithm
- What is a linear search
- What is a binary search
- What is the best search algorithm to use depending on your needs

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.

- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using gcc, using the options -Wall -Werror -Wextra -pedantic -std=gnu89
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project, is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl and betty-doc.pl
- You are not allowed to use global variables
- No more than 5 functions per file
- You are only allowed to use the `printf` function of the standard library. Any call to another function like `strdup`, `malloc`, … is forbidden.
- In the following examples, the `main.c` files are shown as examples. You can use them to test your functions, but you don't have to push them to your repo (if you do we won't take them into account). We will use our own `main.c` files at compilation. Our `main.c` files might be different from the one shown in the examples
- The prototypes of all your functions should be included in your header file called `search_algos.h`
- Don't forget to push your header file
- All your header files should be include guarded

More Info

You will be asked to write files containing big O notations. Please use this format:

- `O(1)`
- `O(n)`
- `O(n!)`
- n*m -> `O(nm)`
- n square -> `O(n^2)`
- sqrt n -> `O(sqrt(n))`
- log(n) -> `O(log(n))`
- n * log(n) -> `O(nlog(n))`
- …

Tasks

0. Linear search

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that searches for a value in an array of integers using the Linear search algorithm

- Prototype : `int linear_search(int *array, size_t size, int value);`
- Where `array` is a pointer to the first element of the array to search in
- `size` is the number of elements in `array`
- And `value` is the value to search for
- Your function must return the first index where `value` is located
- If `value` is not present in `array` or if `array` is `NULL`, your function must return `-1`
- Every time you compare a value in the array to the value you are searching, you have to print this value (see example below)

```
wilfried@0x1E-search_algorithms$ cat 0-main.c
#include <stdio.h>
#include <stdlib.h>
#include "search_algos.h"

/**
 * main - Entry point
 *
 * Return: Always EXIT_SUCCESS
 */
int main(void)
{
    int array[] = {
        10, 1, 42, 3, 4, 42, 6, 7, -1, 9
    };
    size_t size = sizeof(array) / sizeof(array[0]);

    printf("Found %d at index: %d\n\n", 3, linear_search(array, size, 3));
    printf("Found %d at index: %d\n\n", 42, linear_search(array, size, 42));
```

```
    printf("Found %d at index: %d\n", 999, linear_search(array, size, 999));
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 0-mai
n.c 0-linear.c -o 0-linear
wilfried@0x1E-search_algorithms$ ./0-linear
Value checked array[0] = [10]
Value checked array[1] = [1]
Value checked array[2] = [42]
Value checked array[3] = [3]
Found 3 at index: 3

Value checked array[0] = [10]
Value checked array[1] = [1]
Value checked array[2] = [42]
Found 42 at index: 2

Value checked array[0] = [10]
Value checked array[1] = [1]
Value checked array[2] = [42]
Value checked array[3] = [3]
Value checked array[4] = [4]
Value checked array[5] = [42]
Value checked array[6] = [6]
Value checked array[7] = [7]
Value checked array[8] = [-1]
Value checked array[9] = [9]
Found 999 at index: -1
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `0-linear.c`

Done? Help Check your code Ask for a new correction QA Review

1. Binary search

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

Write a function that searches for a value in a sorted array of integers using the Binary search algorithm

- Prototype : `int binary_search(int *array, size_t size, int value);`
- Where `array` is a pointer to the first element of the array to search in
- `size` is the number of elements in `array`
- And `value` is the value to search for
- Your function must return the index where `value` is located
- You can assume that `array` will be sorted in ascending order
- You can assume that `value` won't appear more than once in `array`
- If `value` is not present in `array` or if `array` is `NULL`, your function must return `-1`
- You must print the array being searched every time it changes. (e.g. at the beginning and when you search a subarray) (See example)

```
wilfried@0x1E-search_algorithms$ cat 1-main.c
#include <stdio.h>
#include <stdlib.h>
#include "search_algos.h"

/**
 * main - Entry point
 *
 * Return: Always EXIT_SUCCESS
 */
int main(void)
{
    int array[] = {
        0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    };
    size_t size = sizeof(array) / sizeof(array[0]);

    printf("Found %d at index: %d\n\n", 2, binary_search(array, size, 2));
    printf("Found %d at index: %d\n\n", 5, binary_search(array, 5, 5));
    printf("Found %d at index: %d\n", 999, binary_search(array, size, 999));
```

```
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 1-mai
n.c 1-binary.c -o 1-binary
wilfried@0x1E-search_algorithms$ ./1-binary
Searching in array: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Searching in array: 0, 1, 2, 3
Searching in array: 2, 3
Found 2 at index: 2

Searching in array: 0, 1, 2, 3, 4
Searching in array: 3, 4
Searching in array: 4
Found 5 at index: -1

Searching in array: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Searching in array: 5, 6, 7, 8, 9
Searching in array: 8, 9
Searching in array: 9
Found 999 at index: -1
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `1-binary.c`

Done? Help Check your code Ask for a new correction QA Review

2. Big O #0

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

What is the `time complexity` (worst case) of a linear search in an array of size `n`?

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `2-O`

Done? Help Check your code Ask for a new correction QA Review

3. Big O #1

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

What is the `space complexity` (worst case) of an iterative linear search algorithm in an array of size `n`?

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `3-O`

Done? Help Check your code Ask for a new correction QA Review

4. Big O #2

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

What is the `time complexity` (worst case) of a binary search in an array of size `n`?

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `4-O`

Done? Help Check your code Ask for a new correction QA Review

5. Big O #3

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

What is the `space complexity` (worst case) of a binary search in an array of size `n`?

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `5-O`

Done? Help Check your code Ask for a new correction QA Review

6. Big O #4

mandatory

Score: 0.0% (*Checks completed: 0.0%*)

What is the space complexity of this function / algorithm?

```
int **allocate_map(int n, int m)
{
     int **map;

     map = malloc(sizeof(int *) * n);
     for (size_t i = 0; i < n; i++)
     {
          map[i] = malloc(sizeof(int) * m);
     }
     return (map);
}
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `6-O`

Done? Help Check your code Ask for a new correction QA Review

Done with the mandatory tasks? Unlock 9 advanced tasks now!