(/)

Curriculum
SE Foundations
Average: **108.76%**

0x1E. C - Search Algorithms

C **Algorithm**

By: Wilfried Hennuyer

Weight: 1

Project over - took place from Sep 11, 2023 6:00 AM to Sep 14, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 17.0/17 mandatory & 12.0/34 optional
- **Altogether: 135.29%**
  - Mandatory: 100.0%
  - Optional: 35.29%
  - Calculation: 100.0% + (100.0% * 35.29%) == **135.29%**

Resources

Read or watch:

- Search algorithm (/rltoken/ap2kuRv8qrUMyQ0-MY3EXw)
- Space complexity (1) (/rltoken/QK9ENdoTyqGs0d4_M3XE3g)
- Serach Algorithms video playlist (/rltoken/_4-JUPlg6lfKZO2YPHCA7g)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/i0Ru9NIvGBHVAlsg7w5vVg), **without the help of Google**:

Help

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 (https://github.com/alx-tools/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/alx-tools/Betty/blob/master/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: 100.0% (*Checks completed: 100.0%*)

Write a function that searches for a value in an array of integers using the Linear search algorithm (/rltoken/17RKhbmvh_u4ebCwaSxCxg)

- 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-ma
in.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 | QA Review

1. Binary search

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that searches for a value in a sorted array of integers using the Binary search algorithm (/rltoken/SnveFJhSDE7o8bEx-kGGpA)

- 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-ma
in.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 | QA Review

2. Big O #0

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

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 | QA Review

3. Big O #1

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

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 | QA Review

4. Big O #2

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

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: `4-0`

(/)

Done! | Help | Check your code | QA Review

5. Big O #3

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

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: `5-0`

Done! | Help | Check your code | QA Review

6. Big O #4

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

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: `6-0`

(/)

Done! | Help | Check your code | QA Review

7. Jump search

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

Write a function that searches for a value in a sorted array of integers using the Jump search algorithm (/rltoken/1Op40kSYMN23JsOu6F3P1A)

- Prototype: `int jump_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
- You can assume that `array` will be sorted in ascending order
- If `value` is not present in `array` or if `array` is `NULL`, your function must return `-1`
- You have to use the square root of the size of the array as the jump step.
- You can use the `sqrt()` function included in `<math.h>` (don't forget to compile with -lm)
- Every time you compare a value in the array to the value you are searching for, you have to print this value (see example)

(/)

```
wilfried@0x1E-search_algorithms$ cat 100-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", 6, jump_search(array, size, 6));
    printf("Found %d at index: %d\n\n", 1, jump_search(array, size, 1));
    printf("Found %d at index: %d\n", 999, jump_search(array, size, 999));
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 100-
main.c 100-jump.c -lm -o 100-jump
wilfried@0x1E-search_algorithms$ ./100-jump
Value checked array[0] = [0]
Value checked array[3] = [3]
Value found between indexes [3] and [6]
Value checked array[3] = [3]
Value checked array[4] = [4]
Value checked array[5] = [5]
Value checked array[6] = [6]
Found 6 at index: 6

Value checked array[0] = [0]
Value found between indexes [0] and [3]
Value checked array[0] = [0]
Value checked array[1] = [1]
Found 1 at index: 1

Value checked array[0] = [0]
Value checked array[3] = [3]
Value checked array[6] = [6]
Value checked array[9] = [9]
Value found between indexes [9] and [12]
Value checked array[9] = [9]
Found 999 at index: -1
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `100-jump.c`

(/)

☑ Done! | Help | Check your code | QA Review

8. Big O #5

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

What is the `time complexity` (average case) of a jump search in an array of size `n`, using `step = sqrt(n)`?

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `101-O`

☑ Done! | Help | Check your code | QA Review

9. Interpolation search

#advanced

Score: 75.0% (*Checks completed: 75.0%*)

Write a function that searches for a value in a sorted array of integers using the Interpolation search algorithm (/rltoken/cswpABHiyyRmGrPkzsMTyw)

- Prototype: `int interpolation_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
- You can assume that `array` will be sorted in ascending order
- If `value` is not present in `array` or if `array` is `NULL`, your function must return `-1`
- To determine the probe position, you can use: `size_t pos = low + (((double)(high - low) / (array[high] - array[low])) * (value - array[low]))`
- 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 102-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, 0, 1, 2, 2, 2, 2, 3, 3, 4, 4, 5, 6, 6, 7, 8, 8, 8, 9, 9
    };
    size_t size = sizeof(array) / sizeof(array[0]);

    printf("Found %d at index: %d\n\n", 3, interpolation_search(array, size, 3));
    printf("Found %d at index: %d\n\n", 7, interpolation_search(array, size, 7));
    printf("Found %d at index: %d\n", 999, interpolation_search(array, size, 999));
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 102-
main.c 102-interpolation.c -o 102-interpolation
wilfried@0x1E-search_algorithms$ ./102-interpolation
Value checked array[6] = [2]
Value checked array[7] = [3]
Found 3 at index: 7

Value checked array[14] = [7]
Found 7 at index: 14

Value checked array[2109] is out of range
Found 999 at index: -1
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `102-interpolation.c`

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

10. Exponential search

#advanced

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

Write a function that searches for a value in a sorted array of integers using the Exponential search algorithm (/rltoken/J7wng_ddosamvEkFl0ekqA)

- Prototype: `int exponential_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
- You can assume that `array` will be sorted in ascending order
- If `value` is not present in `array` or if `array` is `NULL`, your function must return `-1`
- You have to use powers of 2 as exponential ranges to search in your array
- Every time you compare a value in the array to the value you are searching for, you have to print this value (See example)
- Once you've found the good range, you need to use a binary search:
    - Every time you split the array, you have to print the new array (or subarray) you're searching in (See example)

(/)

```
wilfried@0x1E-search_algorithms$ cat 103-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, 7, 12, 15, 18, 19, 23, 54, 61, 62, 76, 99
    };
    size_t size = sizeof(array) / sizeof(array[0]);

    printf("Found %d at index: %d\n\n", 62, exponential_search(array, size, 62));
    printf("Found %d at index: %d\n\n", 3, exponential_search(array, size, 3));
    printf("Found %d at index: %d\n", 999, exponential_search(array, size, 999));
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 103-
main.c 103-exponential.c -o 103-exponential
wilfried@0x1E-search_algorithms$ ./103-exponential
Value checked array[1] = [1]
Value checked array[2] = [2]
Value checked array[4] = [4]
Value checked array[8] = [18]
Value found between indexes [8] and [15]
Searching in array: 18, 19, 23, 54, 61, 62, 76, 99
Searching in array: 61, 62, 76, 99
Found 62 at index: 13

Value checked array[1] = [1]
Value checked array[2] = [2]
Value found between indexes [2] and [4]
Searching in array: 2, 3, 4
Found 3 at index: 3

Value checked array[1] = [1]
Value checked array[2] = [2]
Value checked array[4] = [4]
Value checked array[8] = [18]
Value found between indexes [8] and [15]
Searching in array: 18, 19, 23, 54, 61, 62, 76, 99
Searching in array: 61, 62, 76, 99
Searching in array: 76, 99
Searching in array: 99
Found 999 at index: -1
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `103-exponential.c`

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

11. Advanced binary search

#advanced

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

You may have noticed that the basic binary search does not necessarily return the index of the *first* value in the array (if this value appears more than once in the array). In this exercise, you'll have to solve this problem.

Write a function that searches for a value in a sorted array of integers.

- Prototype: `int advanced_binary(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
- If `value` is not present in `array` or if `array` is `NULL`, your function must return `-1`
- Every time you split the array, you have to print the new array (or subarray) you're searching in (See example)
- You have to use recursion. You may only use one loop (`while`, `for`, `do while`, etc.) in order to print the array

(7)

```
wilfried@0x1E-search_algorithms$ cat 104-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, 5, 5, 6, 6, 7, 8, 9
    };
    size_t size = sizeof(array) / sizeof(array[0]);

    printf("Found %d at index: %d\n\n", 8, advanced_binary(array, size, 8));
    printf("Found %d at index: %d\n\n", 5, advanced_binary(array, size, 5));
    printf("Found %d at index: %d\n", 999, advanced_binary(array, size, 999));
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 104-
main.c 104-advanced_binary.c -o 104-advanced_binary
wilfried@0x1E-search_algorithms$ ./104-advanced_binary
Searching in array: 0, 1, 2, 5, 5, 6, 6, 7, 8, 9
Searching in array: 6, 6, 7, 8, 9
Searching in array: 8, 9
Found 8 at index: 8

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

Searching in array: 0, 1, 2, 5, 5, 6, 6, 7, 8, 9
Searching in array: 6, 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: `104-advanced_binary.c`

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

12. Jump search in a singly linked list (7)

#advanced

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

You might think that linear search is not as effective as any other algorithm, right? Well, we should see what happens with a singly linked list.

Please define the following data structure in your `search_algos.h` header file:

```
/**
 * struct listint_s - singly linked list
 *
 * @n: Integer
 * @index: Index of the node in the list
 * @next: Pointer to the next node
 *
 * Description: singly linked list node structure
 */
typedef struct listint_s
{
    int n;
    size_t index;
    struct listint_s *next;
} listint_t;
```

Write a function that searches for a value in a sorted list of integers using the Jump search algorithm.

- Prototype: `listint_t *jump_list(listint_t *list, size_t size, int value);`
- Where `list` is a pointer to the head of the list to search in
- `size` is the number of nodes in `list`
- And `value` is the value to search for
- Your function must return a pointer to the first node where `value` is located
- You can assume that `list` will be sorted in ascending order
- If `value` is not present in `head` or if `head` is `NULL`, your function must return `NULL`
- You have to use the square root of the size of the list as the jump step.
- You can use the `sqrt()` function included in `<math.h>` (don't forget to compile with -lm)
- Every time you compare a value in the list to the value you are searching, you have to print this value (see example)

NOTE: You can find here (/rltoken/7EwC08L6K_vQyI2wknLvnQ) the functions used in the example. You don't need to push them, we will compile your file with our own implementation during the correction.

(/)

```
wilfried@0x1E-search_algorithms$ cat 105-main.c
#include <stdio.h>
#include <stdlib.h>
#include "search_algos.h"

listint_t *create_list(int *array, size_t size);
void print_list(const listint_t *list);
void free_list(listint_t *list);

/**
 * main - Entry point
 *
 * Return: Always EXIT_SUCCESS
 */
int main(void)
{
    listint_t *list, *res;
    int array[] = {
        0, 1, 2, 3, 4, 7, 12, 15, 18, 19, 23, 53, 61, 62, 76, 99
    };
    size_t size = sizeof(array) / sizeof(array[0]);

    list = create_list(array, size);
    print_list(list);

    res =  jump_list(list, size, 53);
    printf("Found %d at index: %lu\n\n", 53, res->index);
    res =  jump_list(list, size, 2);
    printf("Found %d at index: %lu\n\n", 2, res->index);
    res =  jump_list(list, size, 999);
    printf("Found %d at index: %p\n", 999, (void *) res);

    free_list(list);
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 105-
main.c 105-jump_list.c listint/*.c -lm -o 105-jump
wilfried@0x1E-search_algorithms$ ./105-jump
List :
Index[0] = [0]
Index[1] = [1]
Index[2] = [2]
Index[3] = [3]
Index[4] = [4]
Index[5] = [7]
Index[6] = [12]
Index[7] = [15]
Index[8] = [18]
Index[9] = [19]
Index[10] = [23]
Index[11] = [53]
Index[12] = [61]
```

```
Index[13] = [62]
Index[14] = [76]
Index[15] = [99]

Value checked at index [4] = [4]
Value checked at index [8] = [18]
Value checked at index [12] = [61]
Value found between indexes [8] and [12]
Value checked at index [8] = [18]
Value checked at index [9] = [19]
Value checked at index [10] = [23]
Value checked at index [11] = [53]
Found 53 at index: 11

Value checked at index [4] = [4]
Value found between indexes [0] and [4]
Value checked at index [0] = [0]
Value checked at index [1] = [1]
Value checked at index [2] = [2]
Found 2 at index: 2

Value checked at index [4] = [4]
Value checked at index [8] = [18]
Value checked at index [12] = [61]
Value checked at index [15] = [99]
Value found between indexes [12] and [15]
Value checked at index [12] = [61]
Value checked at index [13] = [62]
Value checked at index [14] = [76]
Value checked at index [15] = [99]
Found 999 at index: (nil)
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `105-jump_list.c`

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

13. Linear search in a skip list

#advanced

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

As you see now, looking for a specific value in a singly linked list always leads to browse every element of the list. A common way to optimize the time complexity of a search in a singly linked list is to modify the list itself by adding an “express lane” to browse it. A linked list with an express lane is called a skip list

(/rltoken/SD8K3P6iYfmYTq39XZzo_Q). This change does not come without consequences. Indeed, the space complexity of a search in this kind of list will grow as `sizeof(skiplist_t) > sizeof(listint_t)` (see example below).

Please define the following data structure in your `search_algos.h` header file:

```
/**
 * struct skiplist_s - Singly linked list with an express lane
 *
 * @n: Integer
 * @index: Index of the node in the list
 * @next: Pointer to the next node
 * @express: Pointer to the next node in the express lane
 *
 * Description: singly linked list node structure with an express lane
 */
typedef struct skiplist_s
{
    int n;
    size_t index;
    struct skiplist_s *next;
    struct skiplist_s *express;
} skiplist_t;
```

Write a function that searches for a value in a sorted skip list of integers.

- Prototype: `skiplist_t *linear_skip(skiplist_t *list, int value);`
- Where `list` is a pointer to the head of the skip list to search in
- A node of the express lane is placed every index which is a multiple of the square root of the size of the list (see example below)
- And `value` is the value to search for
- You can assume that `list` will be sorted in ascending order
- Your function must return a pointer on the first node where `value` is located
- If `value` is not present in `list` or if `head` is `NULL`, your function must return `NULL`
- Every time you compare a value in the list to the value you are searching, you have to print this value (see example below)

NOTE: You can find here (/rltoken/Br9jXygWf5gbgGxZI45ukA) the functions used in the example. You don't need to push them, we will compile your file with our own implementation during the correction.

(/)

```
wilfried@0x1E-search_algorithms$ cat 106-main.c
#include <stdio.h>
#include <stdlib.h>
#include "search_algos.h"

skiplist_t *create_skiplist(int *array, size_t size);
void print_skiplist(const skiplist_t *list);
void free_skiplist(skiplist_t *list);

/**
 * main - Entry point
 *
 * Return: Always EXIT_SUCCESS
 */
int main(void)
{
    skiplist_t *list, *res;
    int array[] = {
        0, 1, 2, 3, 4, 7, 12, 15, 18, 19, 23, 53, 61, 62, 76, 99
    };
    size_t size = sizeof(array) / sizeof(array[0]);

    list = create_skiplist(array, size);
    print_skiplist(list);

    res =  linear_skip(list, 53);
    printf("Found %d at index: %lu\n\n", 53, res->index);
    res =  linear_skip(list, 2);
    printf("Found %d at index: %lu\n\n", 2, res->index);
    res =  linear_skip(list, 999);
    printf("Found %d at index: %p\n", 999, (void *) res);

    free_skiplist(list);
    return (EXIT_SUCCESS);
}
wilfried@0x1E-search_algorithms$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 106-
main.c 106-linear_skip.c skiplist/*.c -lm -o 106-linear
wilfried@0x1E-search_algorithms$ ./106-linear
List :
Index[0] = [0]
Index[1] = [1]
Index[2] = [2]
Index[3] = [3]
Index[4] = [4]
Index[5] = [7]
Index[6] = [12]
Index[7] = [15]
Index[8] = [18]
Index[9] = [19]
Index[10] = [23]
Index[11] = [53]
Index[12] = [61]
```

```
Index[13] = [62]
Index[14] = [76]
Index[15] = [99]

Express lane :
Index[0] = [0]
Index[4] = [4]
Index[8] = [18]
Index[12] = [61]

Value checked at index [4] = [4]
Value checked at index [8] = [18]
Value checked at index [12] = [61]
Value found between indexes [8] and [12]
Value checked at index [8] = [18]
Value checked at index [9] = [19]
Value checked at index [10] = [23]
Value checked at index [11] = [53]
Found 53 at index: 11

Value checked at index [4] = [4]
Value found between indexes [0] and [4]
Value checked at index [0] = [0]
Value checked at index [1] = [1]
Value checked at index [2] = [2]
Found 2 at index: 2

Value checked at index [4] = [4]
Value checked at index [8] = [18]
Value checked at index [12] = [61]
Value found between indexes [12] and [15]
Value checked at index [12] = [61]
Value checked at index [13] = [62]
Value checked at index [14] = [76]
Value checked at index [15] = [99]
Found 999 at index: (nil)
```

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `106-linear_skip.c`

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

14. Big O #6

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

What is the `time complexity` (average case) of a jump search in a singly linked list of size `n`, using `step = sqrt(n)`?

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `107-O`

Done! | Help | Check your code | QA Review

15. Big O #7

#advanced

Score: 100.0% (*Checks completed: 100.0%*)

What is the `time complexity` (average case) of a jump search in a skip list of size `n`, with an express lane using `step = sqrt(n)`?

Repo:

- GitHub repository: `alx-low_level_programming`
- Directory: `0x1E-search_algorithms`
- File: `108-O`

Done! | Help | Check your code | QA Review