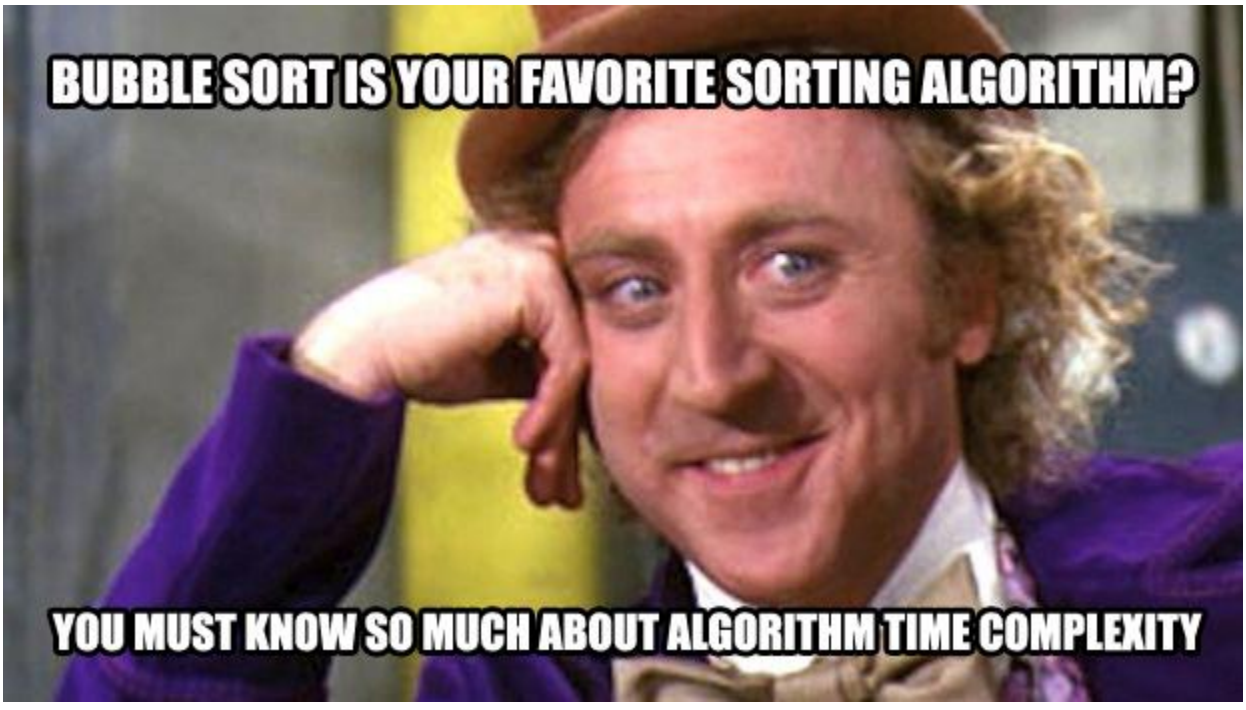
0x1B. C - Sorting algorithms & Big O

CAlgorithmData structure

- By: Alexandre Gautier
- Weight: 2
- Project to be done in teams of 2 people (your team: Emmanuel Ahuron)
- Project over - took place from Jul 13, 2022 6:00 AM to Jul 20, 2022 6:00 AM
- An auto review will be launched at the deadline

In a nutshell…

- **Contribution:** 100.0%
- **Auto QA review:** 0.0/48 mandatory & 0.0/100 optional
- **Altogether: 0.0%**
  - Mandatory: 0.0%
  - Optional: 0.0%
  - Contribution: 100.0%
  - Calculation: 100.0% * (0.0% + (0.0% * 0.0%)) == **0.0%**



Background Context

This project is meant to be done by groups of two students. Each group of two should pair program for at least the mandatory part.

Resources

Read or watch:

- Sorting algorithm
- Big O notation
- Sorting algorithms animations
- 15 sorting algorithms in 6 minutes (***WARNING**: The following video can trigger seizure/epilepsy. It is not required for the project, as it is only a funny visualization of different sorting algorithms*)

Learning Objectives

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

General

- At least four different sorting algorithms
- What is the Big O notation, and how to evaluate the time complexity of an algorithm
- How to select the best sorting algorithm for a given input
- What is a stable sorting algorithm

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
- Unless specified otherwise, you are not allowed to use the standard library. Any use of functions like *printf, puts, …* is totally 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 `sort.h`
- Don't forget to push your header file
- All your header files should be include guarded
- A list/array does not need to be sorted if its size is less than 2.

GitHub

There should be one project repository per group. If you clone/fork/whatever a project repository with the same name before the second deadline, you risk a 0% score.

More Info

Data Structure and Functions

- For this project you are given the following `print_array`, and `print_list` functions:

```
#include <stdlib.h>
#include <stdio.h>


/**
 * print_array - Prints an array of integers
```

```
 *
 * @array: The array to be printed
 * @size: Number of elements in @array
 */
void print_array(const int *array, size_t size)
{
    size_t i;

    i = 0;
    while (array && i < size)
    {
        if (i > 0)
            printf(", ");
        printf("%d", array[i]);
        ++i;
    }
    printf("\n");
}
#include <stdio.h>
#include "sort.h"

/**
 * print_list - Prints a list of integers
 *
 * @list: The list to be printed
 */
void print_list(const listint_t *list)
{
    int i;

    i = 0;
    while (list)
    {
```

```
        if (i > 0)
            printf(", ");
        printf("%d", list->n);
        ++i;
        list = list->next;
    }
    printf("\n");
}
```

- Our files `print_array.c` and `print_list.c` (containing the `print_array` and `print_list` functions) will be compiled with your functions during the correction.
- Please declare the prototype of the functions `print_array` and `print_list` in your `sort.h` header file
- Please use the following data structure for doubly linked list:

```
/**
 * struct listint_s - Doubly linked list node
 *
 * @n: Integer stored in the node
 * @prev: Pointer to the previous element of the list
 * @next: Pointer to the next element of the list
 */
typedef struct listint_s
{
    const int n;
    struct listint_s *prev;
    struct listint_s *next;
} listint_t;
```

Please, note this format is used for Quiz and Task questions.

- `O(1)`
- `O(n)`
- `O(n!)`
- n square -> `O(n^2)`
- log(n) -> `O(log(n))`
- n * log(n) -> `O(nlog(n))`

- n + k -> `O(n+k)`
- ...

Please use the "short" notation (don't use constants). Example: `O(nk)` or `O(wn)` should be written `O(n)`. If an answer is required within a file, all your answers files must have a newline at the end.

Tests

Here is a quick tip to help you test your sorting algorithms with big sets of random integers: Random.org

Quiz questions

Great! You've completed the quiz successfully! Keep going! (Show quiz)

Tasks

0. Bubble sort

mandatory

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

Write a function that sorts an array of integers in ascending order using the Bubble sort algorithm

- Prototype: `void bubble_sort(int *array, size_t size);`
- You're expected to print the `array` after each time you swap two elements (See example below)

Write in the file `0-O`, the big O notations of the time complexity of the Bubble sort algorithm, with 1 notation per line:

- in the best case
- in the average case
- in the worst case

```
alex@/tmp/sort$ cat 0-main.c
#include <stdio.h>
#include <stdlib.h>
#include "sort.h"

/**
 * main - Entry point
```

```
 *
 * Return: Always 0
 */
int main(void)
{
    int array[] = {19, 48, 99, 71, 13, 52, 96, 73, 86, 7};
    size_t n = sizeof(array) / sizeof(array[0]);


    print_array(array, n);
    printf("\n");
    bubble_sort(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic  -std=gnu89 0-bubble_sort.c 0-mai
n.c print_array.c -o bubble
alex@/tmp/sort$ ./bubble
19, 48, 99, 71, 13, 52, 96, 73, 86, 7


19, 48, 71, 99, 13, 52, 96, 73, 86, 7
19, 48, 71, 13, 99, 52, 96, 73, 86, 7
19, 48, 71, 13, 52, 99, 96, 73, 86, 7
19, 48, 71, 13, 52, 96, 99, 73, 86, 7
19, 48, 71, 13, 52, 96, 73, 99, 86, 7
19, 48, 71, 13, 52, 96, 73, 86, 99, 7
19, 48, 71, 13, 52, 96, 73, 86, 7, 99
19, 48, 13, 71, 52, 96, 73, 86, 7, 99
19, 48, 13, 52, 71, 96, 73, 86, 7, 99
19, 48, 13, 52, 71, 73, 96, 86, 7, 99
19, 48, 13, 52, 71, 73, 86, 96, 7, 99
19, 48, 13, 52, 71, 73, 86, 7, 96, 99
19, 13, 48, 52, 71, 73, 86, 7, 96, 99
19, 13, 48, 52, 71, 73, 7, 86, 96, 99
```

```
13, 19, 48, 52, 71, 73, 7, 86, 96, 99
13, 19, 48, 52, 71, 7, 73, 86, 96, 99
13, 19, 48, 52, 7, 71, 73, 86, 96, 99
13, 19, 48, 7, 52, 71, 73, 86, 96, 99
13, 19, 7, 48, 52, 71, 73, 86, 96, 99
13, 7, 19, 48, 52, 71, 73, 86, 96, 99
7, 13, 19, 48, 52, 71, 73, 86, 96, 99

7, 13, 19, 48, 52, 71, 73, 86, 96, 99
alex@/tmp/sort$
```

Repo:

- GitHub repository: `sorting_algorithms`
- File: `0-bubble_sort.c, 0-O`

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

1. Insertion sort

mandatory

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

Write a function that sorts a doubly linked list of integers in ascending order using the Insertion sort algorithm

- Prototype: `void insertion_sort_list(listint_t **list);`
- You are not allowed to modify the integer `n` of a node. You have to swap the nodes themselves.
- You're expected to print the `list` after each time you swap two elements (See example below)

Write in the file `1-O`, the big O notations of the time complexity of the Insertion sort algorithm, with 1 notation per line:

- in the best case
- in the average case
- in the worst case

```
alex@/tmp/sort$ cat 1-main.c
#include <stdio.h>
#include <stdlib.h>
```

```c
#include "sort.h"


/**
 * create_listint - Creates a doubly linked list from an array of integers
 *
 * @array: Array to convert to a doubly linked list
 * @size: Size of the array
 *
 * Return: Pointer to the first element of the created list. NULL on failure
 */
listint_t *create_listint(const int *array, size_t size)
{
    listint_t *list;
    listint_t *node;
    int *tmp;

    list = NULL;
    while (size--)
    {
        node = malloc(sizeof(*node));
        if (!node)
            return (NULL);
        tmp = (int *)&node->n;
        *tmp = array[size];
        node->next = list;
        node->prev = NULL;
        list = node;
        if (list->next)
            list->next->prev = list;
    }
    return (list);
}
```

```
/**
 * main - Entry point
 *
 * Return: Always 0
 */
int main(void)
{
    listint_t *list;
    int array[] = {19, 48, 99, 71, 13, 52, 96, 73, 86, 7};
    size_t n = sizeof(array) / sizeof(array[0]);


    list = create_listint(array, n);
    if (!list)
        return (1);
    print_list(list);
    printf("\n");
    insertion_sort_list(&list);
    printf("\n");
    print_list(list);
    return (0);
}
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic  -std=gnu89 1-main.c 1-insertion_
sort_list.c print_list.c -o insertion
alex@/tmp/sort$ ./insertion
19, 48, 99, 71, 13, 52, 96, 73, 86, 7


19, 48, 71, 99, 13, 52, 96, 73, 86, 7
19, 48, 71, 13, 99, 52, 96, 73, 86, 7
19, 48, 13, 71, 99, 52, 96, 73, 86, 7
19, 13, 48, 71, 99, 52, 96, 73, 86, 7
13, 19, 48, 71, 99, 52, 96, 73, 86, 7
13, 19, 48, 71, 52, 99, 96, 73, 86, 7
13, 19, 48, 52, 71, 99, 96, 73, 86, 7
13, 19, 48, 52, 71, 96, 99, 73, 86, 7
```

```
13, 19, 48, 52, 71, 96, 73, 99, 86, 7
13, 19, 48, 52, 71, 73, 96, 99, 86, 7
13, 19, 48, 52, 71, 73, 96, 86, 99, 7
13, 19, 48, 52, 71, 73, 86, 96, 99, 7
13, 19, 48, 52, 71, 73, 86, 96, 7, 99
13, 19, 48, 52, 71, 73, 86, 7, 96, 99
13, 19, 48, 52, 71, 73, 7, 86, 96, 99
13, 19, 48, 52, 71, 7, 73, 86, 96, 99
13, 19, 48, 52, 7, 71, 73, 86, 96, 99
13, 19, 48, 7, 52, 71, 73, 86, 96, 99
13, 19, 7, 48, 52, 71, 73, 86, 96, 99
13, 7, 19, 48, 52, 71, 73, 86, 96, 99
7, 13, 19, 48, 52, 71, 73, 86, 96, 99


7, 13, 19, 48, 52, 71, 73, 86, 96, 99
alex@/tmp/sort$
```

Repo:

- GitHub repository: `sorting_algorithms`
- File: `1-insertion_sort_list.c, 1-O`

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

2. Selection sort

mandatory

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

Write a function that sorts an array of integers in ascending order using the Selection sort algorithm

- Prototype: `void selection_sort(int *array, size_t size);`
- You're expected to print the `array` after each time you swap two elements (See example below)

Write in the file `2-O`, the big O notations of the time complexity of the Selection sort algorithm, with 1 notation per line:

- in the best case

- in the average case
- in the worst case

```
alex@/tmp/sort$ cat 2-main.c
#include <stdio.h>
#include <stdlib.h>
#include "sort.h"

/**
 * main - Entry point
 *
 * Return: Always 0
 */
int main(void)
{
    int array[] = {19, 48, 99, 71, 13, 52, 96, 73, 86, 7};
    size_t n = sizeof(array) / sizeof(array[0]);

    print_array(array, n);
    printf("\n");
    selection_sort(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic  -std=gnu89
2-main.c 2-selection_sort.c print_array.c -o select
alex@/tmp/sort$ ./select
19, 48, 99, 71, 13, 52, 96, 73, 86, 7

7, 48, 99, 71, 13, 52, 96, 73, 86, 19
7, 13, 99, 71, 48, 52, 96, 73, 86, 19
7, 13, 19, 71, 48, 52, 96, 73, 86, 99
7, 13, 19, 48, 71, 52, 96, 73, 86, 99
```

```
7, 13, 19, 48, 52, 71, 96, 73, 86, 99
7, 13, 19, 48, 52, 71, 73, 96, 86, 99
7, 13, 19, 48, 52, 71, 73, 86, 96, 99

7, 13, 19, 48, 52, 71, 73, 86, 96, 99
alex@/tmp/sort$
```

Repo:

- GitHub repository: `sorting_algorithms`
- File: `2-selection_sort.c, 2-O`

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

3. Quick sort

mandatory

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

Write a function that sorts an array of integers in ascending order using the Quick sort algorithm

- Prototype: `void quick_sort(int *array, size_t size);`
- You must implement the `Lomuto` partition scheme.
- The pivot should always be the last element of the partition being sorted.
- You're expected to print the `array` after each time you swap two elements (See example below)

Write in the file `3-O`, the big O notations of the time complexity of the Quick sort algorithm, with 1 notation per line:

- in the best case
- in the average case
- in the worst case

```
alex@/tmp/sort$ cat 3-main.c
#include <stdio.h>
#include <stdlib.h>
#include "sort.h"

/**
 * main - Entry point
```

```
 *
 * Return: Always 0
 */
int main(void)
{
    int array[] = {19, 48, 99, 71, 13, 52, 96, 73, 86, 7};
    size_t n = sizeof(array) / sizeof(array[0]);


    print_array(array, n);
    printf("\n");
    quick_sort(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic  -std=gnu89 3-main.c 3-quick_sort
.c print_array.c -o quick
alex@/tmp/sort$ ./quick
19, 48, 99, 71, 13, 52, 96, 73, 86, 7


7, 48, 99, 71, 13, 52, 96, 73, 86, 19
7, 13, 99, 71, 48, 52, 96, 73, 86, 19
7, 13, 19, 71, 48, 52, 96, 73, 86, 99
7, 13, 19, 71, 48, 52, 73, 96, 86, 99
7, 13, 19, 71, 48, 52, 73, 86, 96, 99
7, 13, 19, 48, 71, 52, 73, 86, 96, 99
7, 13, 19, 48, 52, 71, 73, 86, 96, 99


7, 13, 19, 48, 52, 71, 73, 86, 96, 99
alex@/tmp/sort$
```

Repo:

- GitHub repository: `sorting_algorithms`
- File: `3-quick_sort.c, 3-O`

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

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