



Curriculum

SE Foundations ^

Average: 137.49% v

You have a captain's log due before 2024-04-21 (in 1 day)! Log it now!
(/captain_logs/5596018/edit)

0x1B. C - Sorting algorithms & Big O

C

Algorithm

Data structure

Weight: 2

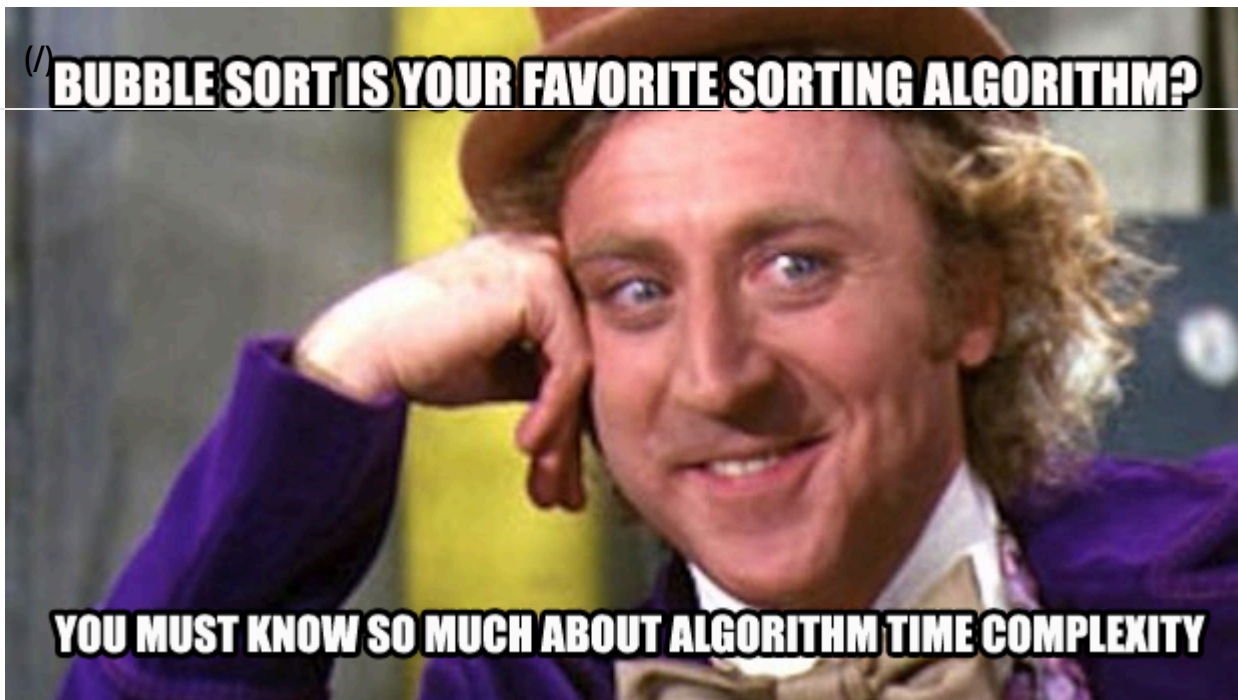
Project to be done in teams of 2 people (your team: Mohamed Madian, Deiaa Elzyat)

 Project over - took place from Nov 15, 2023 6:00 AM to Nov 22, 2023 6:00 AM☒ An auto review will be launched at the deadline

In a nutshell...

- **Contribution:** 100.0%
- **Auto QA review:** 48.0/48 mandatory & 59.0/100 optional
- **Altogether: 159.0%**
 - Mandatory: 100.0%
 - Optional: 59.0%
 - Contribution: 100.0%
 - Calculation: $100.0\% * (100.0\% + (100.0\% * 59.0\%)) == 159.0\%$





Background Context

This project is meant to be done by groups of two students. Each group of two should pair program (/rltoken/glcHRL9I7i1lw2CTAI137A) for at least the mandatory part.

Resources

Read or watch:

- Sorting algorithm (/rltoken/-j5MKLBlzZAC2RfJ5DTBIg)
- Big O notation (/rltoken/WRvrE2BaNVQFssHiUATTrw)
- Sorting algorithms animations (/rltoken/ol0P7NbYVb5R31iOv4Q40A)
- 15 sorting algorithms in 6 minutes (/rltoken/_l0aEvhfJ66Xyob6dd9Utw) (**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)
- CS50 Algorithms explanation in detail by David Malan (/rltoken/Ea93HeEYuNkOL7sGb6zzGg)
- All about sorting algorithms (/rltoken/21X_eaj5RGcLIL9mZv2sqw)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/b-QhraVUoSGmQ1C4QfNoFw), **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` (<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
- 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.

- $O(1)$
- $O(n)$
- $O(n!)$
- $n \text{ square} \rightarrow O(n^2)$
- $\log(n) \rightarrow O(\log(n))$
- $n * \log(n) \rightarrow O(n \log(n))$
- $n + k \rightarrow O(n+k)$
- ...

Please use the "short" notation (don't use constants). Example: $O(nk)$ or $O(w_n)$ 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 (/rltoken/YR-VWQbICB59wZs1eAal3w)

Quiz questions

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



Tasks

Score: 100.0% (Checks completed: 100.0%)



Write a function that sorts an array of integers in ascending order using the Bubble sort (/rltoken/awhP8BhtkGi-lwmMc2-KAw) 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 `0-0` , 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-main.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`, `0-0`

☒ Done!

Check your code

 Get a sandbox

QA Review

1. Insertion sort

mandatory

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



Write a function that sorts a doubly linked list of integers in ascending order using the Insertion sort (`/rltoken/GocxRKbPdsmERXeOHMCO2w`) 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 `1-0` , 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>
#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`, `1-0`

☒ Done!

[Check your code](#)

[>_ Get a sandbox](#)

[QA Review](#)

2. Selection sort

mandatory

Score: 100.0% (Checks completed: 100.0%)



(/)



Write a function that sorts an array of integers in ascending order using the Selection sort (/rltoken/SEbg0fBEraioQcl-igvUSw) 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 2-0 , 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`, `2-0`

☒ Done!

Check your code

 Get a sandbox

QA Review



3. Quick sort

mandatory

Score: 100.0% (Checks completed: 100.0%)

(/)



Write a function that sorts an array of integers in ascending order using the Quick sort (/rltoken/_pBTrH0Xyo4BRmQn4CtnMg) 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 3-0 , 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`, `3-0`

☒ Done!

Check your code

 Get a sandbox

QA Review



4. Shell sort - Knuth Sequence

#advanced

Score: 100.0% (Checks completed: 100.0%)

Write a function that sorts an array of integers in ascending order using the Shell sort (Knuth sequence) algorithm, using the Knuth sequence

- Prototype: `void shell_sort(int *array, size_t size);`
- You must use the following sequence of intervals (a.k.a the Knuth sequence):
 - $n+1 = n * 3 + 1$
 - 1, 4, 13, 40, 121, ...
- You're expected to print the array each time you decrease the interval (See example below).

No big O notations of the time complexity of the Shell sort (Knuth sequence) algorithm needed - as the complexity is dependent on the size of array and gap

```
alex@/tmp/sort$ cat 100-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");
    shell_sort(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 100-main.c 100-shell_sort.c
print_array.c -o shell
alex@/tmp/sort$ ./shell
19, 48, 99, 71, 13, 52, 96, 73, 86, 7

13, 7, 96, 71, 19, 48, 99, 73, 86, 52
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: `100-shell_sort.c`



 Done!

Check your code

 Get a sandbox

QA Review

5. Cocktail shaker sort

#advanced

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

Write a function that sorts a doubly linked list of integers in ascending order using the Cocktail shaker sort (/rltoken/bwa4mHfUbbWTB8J2OIHvpA) algorithm

- Prototype: `void cocktail_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 `101-0`, the big O notations of the time complexity of the Cocktail shaker sort algorithm, with 1 notation per line:

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




```
alex@tmp/sort$ cat 101-main.c
```

```
#include <stdio.h>
#include <stdlib.h>
#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");
}
```



```

    cocktail_sort_list(&list);
(/) printf("\n");
    print_list(list);
    return (0);
}
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 101-main.c 101-cocktail_sort
_list.c print_list.c -o cocktail
alex@/tmp/sort$ ./cocktail
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, 71, 13, 52, 96, 73, 7, 86, 99
19, 48, 71, 13, 52, 96, 7, 73, 86, 99
19, 48, 71, 13, 52, 7, 96, 73, 86, 99
19, 48, 71, 13, 7, 52, 96, 73, 86, 99
19, 48, 71, 7, 13, 52, 96, 73, 86, 99
19, 48, 7, 71, 13, 52, 96, 73, 86, 99
19, 7, 48, 71, 13, 52, 96, 73, 86, 99
7, 19, 48, 71, 13, 52, 96, 73, 86, 99
7, 19, 48, 13, 71, 52, 96, 73, 86, 99
7, 19, 48, 13, 52, 71, 96, 73, 86, 99
7, 19, 48, 13, 52, 71, 73, 96, 86, 99
7, 19, 48, 13, 52, 71, 73, 86, 96, 99
7, 19, 13, 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: 101-cocktail_sort_list.c, 101-0

☒ Done!

[Check your code](#)

[>_ Get a sandbox](#)

[QA Review](#)

6. Counting sort

#advanced



Score: 100.0% (Checks completed: 100.0%)

Write a function that sorts an array of integers in ascending order using the Counting sort (/rltoken/ChcoDSCqnJHGC-qrSPEGHQ) algorithm

- Prototype: `void counting_sort(int *array, size_t size);`
- (/). You can assume that `array` will contain only numbers ≥ 0
- You are allowed to use `malloc` and `free` for this task
- You're expected to print your counting array once it is set up (See example below)
 - This array is of size $k + 1$ where k is the largest number in `array`

Write in the file 102-0 , the big O notations of the time complexity of the Counting sort algorithm, with 1 notation per line:

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


[illegible]

Repo:

- GitHub repository: `sorting_algorithms`
- (/). File: `102-counting_sort.c`, `102-0`

☒ Done!

Check your code

 Get a sandbox

QA Review

7. Merge sort

#advanced

Score: 100.0% (Checks completed: 100.0%)

Write a function that sorts an array of integers in ascending order using the Merge sort (/rltoken/8sZ3nAhd_YLNzHCgNbbf8A) algorithm

- Prototype: `void merge_sort(int *array, size_t size);`
- You must implement the `top-down` merge sort algorithm
 - When you divide an array into two sub-arrays, the size of the left array should always be \leq the size of the right array. i.e. `{1, 2, 3, 4, 5}` -> `{1, 2}`, `{3, 4, 5}`
 - Sort the left array before the right array
- You are allowed to use `printf`
- You are allowed to use `malloc` and `free` only once (only one **call**)
- Output: see example

Write in the file `103-0`, the big O notations of the time complexity of the Merge sort algorithm, with 1 notation per line:

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



```
alex@/tmp/sort$ cat 103-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");
    merge_sort(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
```

```
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 103-main.c 103-merge_sort.c
print_array.c -o merge
```

```
alex@/tmp/sort$ ./merge
```

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

```
Merging...
```

```
[left]: 19
```

```
[right]: 48
```

```
[Done]: 19, 48
```

```
Merging...
```

```
[left]: 71
```

```
[right]: 13
```

```
[Done]: 13, 71
```

```
Merging...
```

```
[left]: 99
```

```
[right]: 13, 71
```

```
[Done]: 13, 71, 99
```

```
Merging...
```

```
[left]: 19, 48
```

```
[right]: 13, 71, 99
```

```
[Done]: 13, 19, 48, 71, 99
```

```
Merging...
```

```
[left]: 52
```

```
[right]: 96
```

```
[Done]: 52, 96
```

```
Merging...
```

```
[left]: 86
```

```
[right]: 7
```

```
[Done]: 7, 86
```

```
Merging...
```



```
[left]: 73
[right]: 7, 86
[Done]: 7, 73, 86
Merging...
[left]: 52, 96
[right]: 7, 73, 86
[Done]: 7, 52, 73, 86, 96
Merging...
[left]: 13, 19, 48, 71, 99
[right]: 7, 52, 73, 86, 96
[Done]: 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: `103-merge_sort.c`, `103-0`

☒ Done!

Check your code

 Get a sandbox

QA Review

8. Heap sort

#advanced

Score: 0.0% (Checks completed: 0.0%)

Write a function that sorts an array of integers in ascending order using the Heap sort (`/rltoken/YKYRdSdomaVkNrtNv1KS6Q`) algorithm

- Prototype: `void heap_sort(int *array, size_t size);`
- You must implement the `sift-down` heap sort algorithm
- You're expected to print the `array` after each time you swap two elements (See example below)

Write in the file `104-0`, the big O notations of the time complexity of the Heap sort algorithm, with 1 notation per line:

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



```
alex@/tmp/sort$ cat 104-main.c
```

```
(7)
#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");
    heap_sort(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
```

```
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 104-main.c 104-heap_sort.c p
rint_array.c -o heap
```

```
alex@/tmp/sort$ ./heap
```

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

```
19, 48, 99, 86, 13, 52, 96, 73, 71, 7
19, 86, 99, 48, 13, 52, 96, 73, 71, 7
19, 86, 99, 73, 13, 52, 96, 48, 71, 7
99, 86, 19, 73, 13, 52, 96, 48, 71, 7
99, 86, 96, 73, 13, 52, 19, 48, 71, 7
7, 86, 96, 73, 13, 52, 19, 48, 71, 99
96, 86, 7, 73, 13, 52, 19, 48, 71, 99
96, 86, 52, 73, 13, 7, 19, 48, 71, 99
71, 86, 52, 73, 13, 7, 19, 48, 96, 99
86, 71, 52, 73, 13, 7, 19, 48, 96, 99
86, 73, 52, 71, 13, 7, 19, 48, 96, 99
48, 73, 52, 71, 13, 7, 19, 86, 96, 99
73, 48, 52, 71, 13, 7, 19, 86, 96, 99
73, 71, 52, 48, 13, 7, 19, 86, 96, 99
19, 71, 52, 48, 13, 7, 73, 86, 96, 99
71, 19, 52, 48, 13, 7, 73, 86, 96, 99
71, 48, 52, 19, 13, 7, 73, 86, 96, 99
7, 48, 52, 19, 13, 71, 73, 86, 96, 99
52, 48, 7, 19, 13, 71, 73, 86, 96, 99
13, 48, 7, 19, 52, 71, 73, 86, 96, 99
48, 13, 7, 19, 52, 71, 73, 86, 96, 99
48, 19, 7, 13, 52, 71, 73, 86, 96, 99
13, 19, 7, 48, 52, 71, 73, 86, 96, 99
19, 13, 7, 48, 52, 71, 73, 86, 96, 99
7, 13, 19, 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: `104-heap_sort.c`, `104-0`

☐ Done?

Check your code

Ask for a new correction

 Get a sandbox

QA Review

9. Radix sort

#advanced

Score: 0.0% (Checks completed: 0.0%)

Write a function that sorts an array of integers in ascending order using the Radix sort (/rltoken/pBsj4j_AF_mJAgNZWmX3VQ) algorithm

- Prototype: `void radix_sort(int *array, size_t size);`
- You must implement the `LSD` radix sort algorithm
- You can assume that `array` will contain only numbers `>= 0`
- You are allowed to use `malloc` and `free` for this task
- You're expected to print the `array` each time you increase your `significant digit` (See example below)




```
alex@/tmp/sort$ cat 105-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");
```

```
    radix_sort(array, n);
```

```
    printf("\n");
```

```
    print_array(array, n);
```

```
    return (0);
```

```
}
```

```
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 105-main.c 105-radix_sort.c
```

```
print_array.c -o radix
```

```
alex@/tmp/sort$ ./radix
```

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

```
71, 52, 13, 73, 96, 86, 7, 48, 19, 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: `105-radix_sort.c`

☐ Done?

Check your code

Ask for a new correction

 Get a sandbox

QA Review

10. Bitonic sort

#advanced

Score: 0.0% (Checks completed: 0.0%)



Write a function that sorts an array of integers in ascending order using the Bitonic sort (/rltoken/N-bjAbxm5yr4DoelDz5lLw) algorithm

- Prototype: `void bitonic_sort(int *array, size_t size);`

- You can assume that `size` will be equal to 2^k , where $k \geq 0$ (when `array` is not `NULL` ...)
- (/).• You are allowed to use `printf`
- You're expected to print the `array` each time you swap two elements (See example below)
- Output: see example

Write in the file `106-0`, the big O notations of the time complexity of the Bitonic sort algorithm, with 1 notation per line:

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



alex@/tmp/sort\$ cat 106-main.c

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

/**
 * main - Entry point
 *
 * Return: Always 0
 */
int main(void)
{
    int array[] = {100, 93, 40, 57, 14, 58, 85, 54, 31, 56, 46, 39, 15, 26, 78, 13};
    size_t n = sizeof(array) / sizeof(array[0]);

    print_array(array, n);
    printf("\n");
    bitonic_sort(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
```

alex@/tmp/sort\$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 106-main.c 106-bitonic_sort.
c print_array.c -o bitonic

alex@/tmp/sort\$./bitonic

100, 93, 40, 57, 14, 58, 85, 54, 31, 56, 46, 39, 15, 26, 78, 13

Merging [16/16] (UP):

100, 93, 40, 57, 14, 58, 85, 54, 31, 56, 46, 39, 15, 26, 78, 13

Merging [8/16] (UP):

100, 93, 40, 57, 14, 58, 85, 54

Merging [4/16] (UP):

100, 93, 40, 57

Merging [2/16] (UP):

100, 93

Result [2/16] (UP):

93, 100

Merging [2/16] (DOWN):

40, 57

Result [2/16] (DOWN):

57, 40

Result [4/16] (UP):

40, 57, 93, 100

Merging [4/16] (DOWN):

14, 58, 85, 54

Merging [2/16] (UP):

14, 58

Result [2/16] (UP):

14, 58

Merging [2/16] (DOWN):

85, 54

Result [2/16] (DOWN):



```

85, 54
Result [4/16] (DOWN):
85, 58, 54, 14
Result [8/16] (UP):
14, 40, 54, 57, 58, 85, 93, 100
Merging [8/16] (DOWN):
31, 56, 46, 39, 15, 26, 78, 13
Merging [4/16] (UP):
31, 56, 46, 39
Merging [2/16] (UP):
31, 56
Result [2/16] (UP):
31, 56
Merging [2/16] (DOWN):
46, 39
Result [2/16] (DOWN):
46, 39
Result [4/16] (UP):
31, 39, 46, 56
Merging [4/16] (DOWN):
15, 26, 78, 13
Merging [2/16] (UP):
15, 26
Result [2/16] (UP):
15, 26
Merging [2/16] (DOWN):
78, 13
Result [2/16] (DOWN):
78, 13
Result [4/16] (DOWN):
78, 26, 15, 13
Result [8/16] (DOWN):
78, 56, 46, 39, 31, 26, 15, 13
Result [16/16] (UP):
13, 14, 15, 26, 31, 39, 40, 46, 54, 56, 57, 58, 78, 85, 93, 100

13, 14, 15, 26, 31, 39, 40, 46, 54, 56, 57, 58, 78, 85, 93, 100
alex@/tmp/sort$

```

Repo:

- GitHub repository: `sorting_algorithms`
- File: `106-bitonic_sort.c`, `106-0`

☐ Done?

☐ Check your code

☒ Ask for a new correction

☐ Get a sandbox

☐ QA Review


11. Quick Sort - Hoare Partition scheme

#advanced

Score: 100.0% (Checks completed: 100.0%)

Write a function that sorts an array of integers in ascending order using the Quick sort
(/token/_pBTrH0Xyo4BRmQn4CtnMg) algorithm

- Prototype: `void quick_sort_hoare(int *array, size_t size);`
- You must implement the Hoare 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 107-0 , 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 107-main.c
1)
#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_hoare(array, n);
    printf("\n");
    print_array(array, n);
    return (0);
}
alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 107-main.c 107-quick_sort_hoare.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, 19, 99, 71, 13, 52, 96, 73, 86, 48
7, 19, 13, 71, 99, 52, 96, 73, 86, 48
7, 13, 19, 71, 99, 52, 96, 73, 86, 48
7, 13, 19, 48, 99, 52, 96, 73, 86, 71
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, 86, 73, 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$

```

Another example of output:



(A)
87



```
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 62, 65, 52, 63, 58, 45, 59, 42, 39, 71, 6
(7) 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 65, 52, 63, 58, 45, 59, 42, 62, 71, 6
9, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 62, 52, 63, 58, 45, 59, 42, 65, 71, 6
9, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 62, 52, 42, 58, 45, 59, 63, 65, 71, 6
9, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 59, 52, 42, 58, 45, 62, 63, 65, 71, 6
9, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 45, 52, 42, 58, 59, 62, 63, 65, 71, 6
9, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 45, 42, 52, 58, 59, 62, 63, 65, 71, 6
9, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 42, 45, 52, 58, 59, 62, 63, 65, 71, 6
9, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 42, 45, 52, 58, 59, 62, 63, 65, 69, 7
1, 75, 87, 83, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 42, 45, 52, 58, 59, 62, 63, 65, 69, 7
1, 75, 83, 87, 93, 92, 88
6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 42, 45, 52, 58, 59, 62, 63, 65, 69, 7
1, 75, 83, 87, 88, 92, 93

6, 8, 11, 13, 16, 21, 24, 26, 27, 28, 30, 36, 38, 39, 42, 45, 52, 58, 59, 62, 63, 65, 69, 7
1, 75, 83, 87, 88, 92, 93
alex@/tmp/sort$
```

Repo:

- GitHub repository: `sorting_algorithms`
- File: `107-quick_sort_hoare.c`, `107-0`

☒ Done!

Check your code

QA Review

12. Dealer

#advanced

Score: 0.0% (Checks completed: 0.0%)



(/)



Write a function that sorts a deck of cards.

- Prototype: `void sort_deck(deck_node_t **deck);`
- You are allowed to use the C standard library function `qsort`
- Please use the following data structures:



```

typedef enum kind_e
{
    SPADE = 0,
    HEART,
    CLUB,
    DIAMOND
} kind_t;

/**
 * struct card_s - Playing card
 *
 * @value: Value of the card
 * From "Ace" to "King"
 * @kind: Kind of the card
 */
typedef struct card_s
{
    const char *value;
    const kind_t kind;
} card_t;

/**
 * struct deck_node_s - Deck of card
 *
 * @card: Pointer to the card of the node
 * @prev: Pointer to the previous node of the list
 * @next: Pointer to the next node of the list
 */
typedef struct deck_node_s
{
    const card_t *card;
    struct deck_node_s *prev;
    struct deck_node_s *next;
} deck_node_t;

```

- You have to push you `deck.h` header file, containing the previous data structures definition
- Each node of the doubly linked list contains a card that you cannot modify. You have to swap the nodes.
- You can assume there is exactly 52 elements in the doubly linked list.
- You are free to use the sorting algorithm of your choice
- The deck must be ordered:
 - From Ace to King
 - From Spades to Diamonds
 - See example below



alex@tmp/sort\$ cat 1000-main.c

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

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

```
#include "deck.h"
```

```
void print_deck(const deck_node_t *deck)
```

```
{
```

```
    size_t i;
```

```
    char kinds[4] = {'S', 'H', 'C', 'D'};
```

```
    i = 0;
```

```
    while (deck)
```

```
    {
```

```
        if (i)
```

```
            printf(", ");
```

```
        printf("{%s, %c}", deck->card->value, kinds[deck->card->kind]);
```

```
        if (i == 12)
```

```
            printf("\n");
```

```
        i = (i + 1) % 13;
```

```
        deck = deck->next;
```

```
    }
```

```
}
```

```
deck_node_t *init_deck(const card_t cards[52])
```

```
{
```

```
    deck_node_t *deck;
```

```
    deck_node_t *node;
```

```
    size_t i;
```

```
    i = 52;
```

```
    deck = NULL;
```

```
    while (i--)
```

```
    {
```

```
        node = malloc(sizeof(*node));
```

```
        if (!node)
```

```
            return (NULL);
```

```
        node->card = &cards[i];
```

```
        node->next = deck;
```

```
        node->prev = NULL;
```

```
        if (deck)
```

```
            deck->prev = node;
```

```
        deck = node;
```

```
    }
```

```
    return (deck);
```

```
}
```

```
int main(void)
```

```
{
```

```
    card_t cards[52] = {
```

```
        {"Jack", CLUB}, {"4", HEART}, {"3", HEART}, {"3", DIAMOND}, {"Queen", HEART}, {"5",  
HEART}, {"5", SPADE}, {"10", HEART}, {"6", HEART}, {"5", DIAMOND}, {"6", SPADE}, {"9", HEAR  
T}, {"7", DIAMOND}, {"Jack", SPADE}, {"Ace", DIAMOND}, {"9", CLUB}, {"Jack", DIAMOND}, {"7",
```



```

SPADE}, {"King", DIAMOND}, {"10", CLUB}, {"King", SPADE}, {"8", CLUB}, {"9", SPADE}, {"6", C
(10)B}, {"Ace", CLUB}, {"3", SPADE}, {"8", SPADE}, {"9", DIAMOND}, {"2", HEART}, {"4", DIAMON
D}, {"6", DIAMOND}, {"3", CLUB}, {"Queen", CLUB}, {"10", SPADE}, {"8", DIAMOND}, {"8", HEAR
T}, {"Ace", SPADE}, {"Jack", HEART}, {"2", CLUB}, {"4", SPADE}, {"2", SPADE}, {"2", DIAMON
D}, {"King", CLUB}, {"Queen", SPADE}, {"Queen", DIAMOND}, {"7", CLUB}, {"7", HEART}, {"5", C
LUB}, {"10", DIAMOND}, {"4", CLUB}, {"King", HEART}, {"Ace", HEART},
};

```

```

deck_node_t *deck;

```

```

deck = init_deck(cards);
print_deck(deck);
printf("\n");
sort_deck(&deck);
printf("\n");
print_deck(deck);
return (0);
}

```

```

alex@/tmp/sort$ gcc -Wall -Wextra -Werror -pedantic -std=gnu89 1000-main.c 1000-sort_deck.c
-o deck

```

```

alex@/tmp/sort$ ./deck

```

```

{Jack, C}, {4, H}, {3, H}, {3, D}, {Queen, H}, {5, H}, {5, S}, {10, H}, {6, H}, {5, D}, {6,
S}, {9, H}, {7, D}
{Jack, S}, {Ace, D}, {9, C}, {Jack, D}, {7, S}, {King, D}, {10, C}, {King, S}, {8, C}, {9,
S}, {6, C}, {Ace, C}, {3, S}
{8, S}, {9, D}, {2, H}, {4, D}, {6, D}, {3, C}, {Queen, C}, {10, S}, {8, D}, {8, H}, {Ace,
S}, {Jack, H}, {2, C}
{4, S}, {2, S}, {2, D}, {King, C}, {Queen, S}, {Queen, D}, {7, C}, {7, H}, {5, C}, {10, D},
{4, C}, {King, H}, {Ace, H}

```

```

{Ace, S}, {2, S}, {3, S}, {4, S}, {5, S}, {6, S}, {7, S}, {8, S}, {9, S}, {10, S}, {Jack,
S}, {Queen, S}, {King, S}
{Ace, H}, {2, H}, {3, H}, {4, H}, {5, H}, {6, H}, {7, H}, {8, H}, {9, H}, {10, H}, {Jack,
H}, {Queen, H}, {King, H}
{Ace, C}, {2, C}, {3, C}, {4, C}, {5, C}, {6, C}, {7, C}, {8, C}, {9, C}, {10, C}, {Jack,
C}, {Queen, C}, {King, C}
{Ace, D}, {2, D}, {3, D}, {4, D}, {5, D}, {6, D}, {7, D}, {8, D}, {9, D}, {10, D}, {Jack,
D}, {Queen, D}, {King, D}
alex@/tmp/sort$

```

Repo:

- GitHub repository: `sorting_algorithms`
- File: `1000-sort_deck.c`, `deck.h`



☐ Done?

☐ Check your code

☒ Ask for a new correction

☐ Get a sandbox

☐ QA Review

(/)

