

CENG 315

Algorithms

Fall 2025-2026

Take-Home Exam 2

Due date: 2 November 2025, Sunday, 23.59

1 Problem Definition

In this homework, you will implement a radix sort algorithm that supports an arbitrary base X. In the scope of THE2, base values are restricted to the power of 2 for efficient digit calculation. Your goal is to understand how radix sort operates under different base values and to analyse how the choice of base values affects the performance of the algorithm in practice and return the number of iterations required in the counting sorts to observe the complexity of the designed algorithm.

At each iteration, the algorithm sorts the array according to a single digit (you should calculate the necessary digits according to given base X, because integers in the array correspond to decimal values.) using counting sort starting from the least significant digit (LSD) until all digits have been processed.

The base x directly determines:

- The number of buckets used in each iteration (memory requirement),
- The number of passes required to sort all digits (algorithmic complexity).

```
long base_x_radix_sort(unsigned int* arr,  
                      const size_t size,  
                      const unsigned int base2);
```

This function should sort the array arr in ascending order using a base X radix sort and return the total number of iterations during the counting sorts for each digit.

Parameters:

- `unsigned int* arr`: pointer to the array that will be sorted in place.
- `size_t size`: number of elements in the array.
- `int base2`: indicates the base of the algorithm. The base will be defined as 2^{base2} . For example, if `base2 = 4`, then `base = 24 = 16`

2 Example I/O

Example 1

```
Size: 1
Base2: 2, Base: 4
Array elements: {506}
Iteration: 30
Sorted array elements: {506}
```

Example 2

```
Size: 8
Base2: 1, Base: 2
Array elements: {1083, 1581, 2983, 1969, 3009, 130, 1524, 1084}
Iteration: 300
Sorted array elements: {130, 1083, 1084, 1524, 1581, 1969, 2983, 3009}
```

Example 3

```
Size: 10
Base2: 3, Base: 8
Array elements: {2331, 99, 1758, 1833, 3255, 2496, 3443, 3266, 1050, 2104}
Iteration: 148
Sorted array elements: {99, 1050, 1758, 1833, 2104, 2331, 2496, 3255, 3266, 3443}
```

3 Specifications

- You will implement your solutions in the `the2.cpp` file.
- You are free to add other functions to `the2.cpp`.
- Do not change the first line of `the2.cpp`, which is `#include "the2.h"`
- Do not change the arguments and the return value of the function `base_x_radix_sort()` in the file `the2.cpp`.
- Do not include any other library or write include anywhere in your `the2.cpp` file (not even in comments).
- You are given a `test.cpp` file to test your work on ODTUCLASS or your locale. You can and you are encouraged to modify this file to add different test cases.
- You can test your `the2.cpp` on the virtual lab environment. If you click run, your function will be compiled and executed with `test.cpp` with random I/O. If you click evaluate, you will get feedback for your current work and your work will be temporarily graded with a limited number of inputs.
- The grade you see in VPL is not your final grade, your code will be reevaluated with more inputs after the exam.
- If you want to test your work and see your outputs on your locale you can use the following commands:

```
> g++ test.cpp the2.cpp -Wall -std=c++11 -o test  
> ./test
```

4 Constraints, Limits and Hints

- The system has the following limits to test the complexity of your solutions:
 - a maximum execution time of 1 minute
 - a 256 MB maximum memory limit
 - a stack size of 64 MB for function calls (ie. recursive solutions)
- Solutions with longer running times will not be graded.
- There are some important points to keep in mind:
 - The array elements will be unsigned integers in its decimal form. You should convert each element to its baseX form and extract its digits in each counting sort iteration. (Because the base is restricted to the powers of 2, think about fast bitwise digit extraction.)
 - It will be easier to follow the count of iterations if you implement your solution by modifying the pseudocodes given in your book.
 - Different from the Radix Sort algorithm in your book, it is not guaranteed that the numbers in the array will always have the same length.
 - Different than the algorithm for Counting Sort in your book, initialize the count array as `int* C = new int[k]` and use the fourth loop for copying the array back. That means, you shouldn't count iterations during initialization, but you should count iterations during copying array back. Otherwise, the return value of the function (`iterations`) will not be evaluated as correct.
 - You should count loop iterations in four different loops.
 - You can make sure that the size for the array `arr` and group size will always be given to stay in the limitations of the VPL environment. Since the complexity of your implementation will be checked by your returned number of iterations, we will not test your code with such edge cases.

5 Regulations

- **Implementation and Submission:** The template files are available in the Virtual Programming Lab (VPL) activity called “THE2” on ODTUCLASS. At this point, you have two options:
 - You can download the template files, complete the implementation, and test it with the given sample I/O on your local machine. Then submit the same file through this activity.
 - You can directly use the editor of the VPL environment by using the auto-evaluation feature of this activity interactively. Saving the code is equivalent to submitting a file.

Please make sure that your code runs on ODTUCLASS. There is no limitation in running your code online. The last save/submission will determine your final grade.

- **Programming Language:** You must code your program in C++11. Your submission will be tested on the VPL environment in ODTUCLASS, hence you are expected to make sure your code runs successfully there.

- **Cheating:** This assignment is designed to be worked on individually. Additionally, the use of any LLMs (ChatGPT, Copilot, the other one that you are thinking about...) and copying code directly from the internet for implementations is strictly forbidden. Your work will be evaluated for cheating, and disciplinary action may be taken if necessary.
- **Evaluation:** Your program will be evaluated automatically using “black-box” testing, so make sure to obey the specifications. No erroneous input will be used. Therefore, you don’t have to worry about invalid cases.
Important Note: The given sample I/O’s are only to ease your debugging process and NOT official. Furthermore, it is not guaranteed that they cover all the cases of required functions. As a programmer, it is your responsibility to consider such extreme cases for the functions. Your implementations will be evaluated by the official test cases to determine your final grade after the deadline.