MATEUSZ ZAREMBA, 2ND CGAD

DATA STRUCTURES AND ALGORITHMS 1, CMP201

TESTING PLATFORMS

- University computer in audio lab (White Space)
  - Windows 7 Professional Edition Service Pack 1 (Build 7601)
  - Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz
  - 4 GB RAM
- My personal MacBook Pro
  - Windows 10 Education (BootCamp)
  - Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz
  - 16.0 GB RAM
- All tests done using x86 solution platform

RADIX SORT AND QUICKSORT CHARACTERISTIC

- Radix sort
  - Best-case performance - O(n log n)
  - Worst-case performance   O(nw) :
    - n - number of keys
    - w average key length (unsigned long int - 4 bits)
- Quicksort
  - Best-case performance - O(n log n)
  - Worst-case performance - O(n^2)

SORTING ALGORITHMS

- My radix sort
  - Iterative version using queues as buckets
  - LSN - Least Significant Number
- My quicksort
  - Two partition
  - Pivot in the middle of an array - prevents worst-case behaviour - $O(n^2)$ - on already sorted arrays

APPLICATION STRUCTURE

- Populating vector with $N \in [0, 10e8]$
- Sorting
  - radix sort
  - quicksort
  - std::sort
- Displaying results of each sort

TIME COMPLEXITY OF DATA STRUCTURES USED FOR QUICKSORT AND RADIX SORT

- std::queue - FIFO - first in, first out - don't need to insert or randomly access elements
  - push() - O(1)
  - pop() - O(1)
  - front() - O(1)
- std::vector - random access always O(1) - don't need to insert
  - push_back() - O(1)
  - at() - O(1)
  - empty() - O(1)
  - clear() - O(1)
  - size() - O(1)
  - Make a vector with N elements (doesn't affect sorting) - O(n)

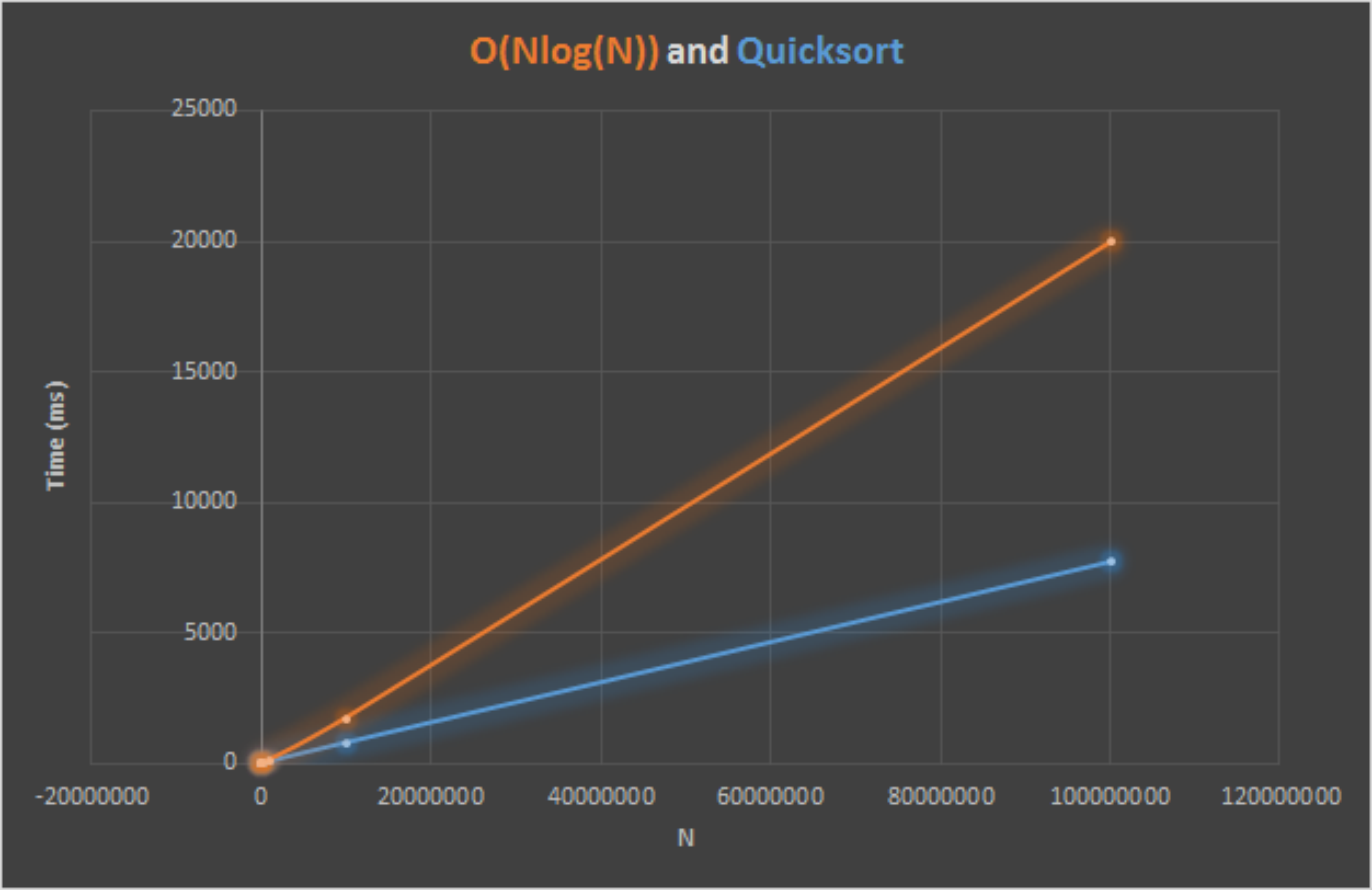
RESULTS TO PRESENT

- Using operator[] and at() function to randomly access a vector makes a slight difference when quicksorting
- In debug mode - up to 100x slower
- In debug mode - inefficient for N > 10e6 for both methods
- Getting rid of warnings - almost 2x faster
- Listening to music through a browser - 1% slower
- For N > 10e8 needs to built using x64 solution platform and requires at least 4GB RAM)
- Constant sorting time (T = 0) for all algorithms when N <= 10e3
- Constant sorting time (T = 0) for quicksort and std::sort when N <= 10e4

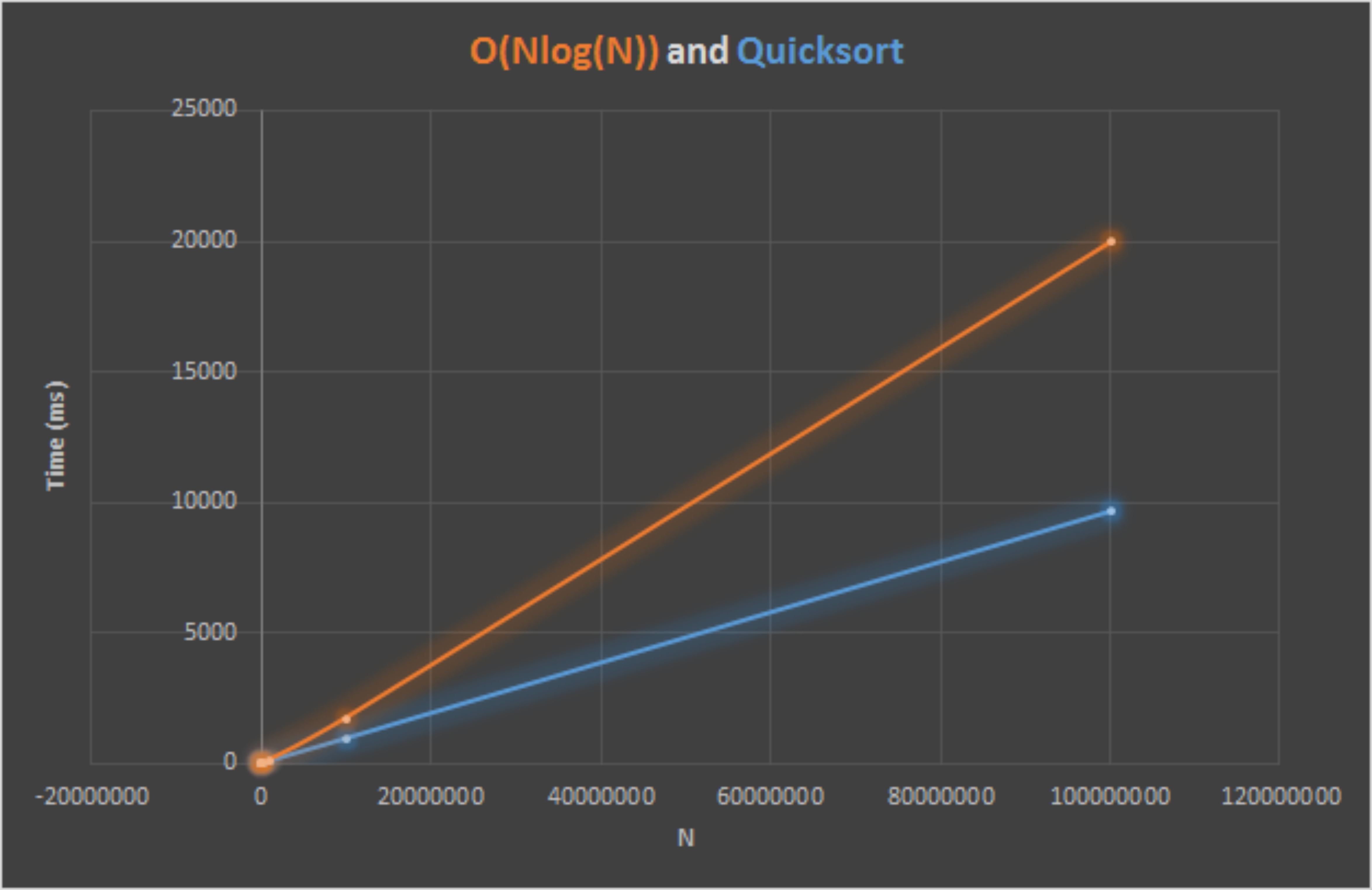
RESULTS - DIFFERENT RANDOM ACCESS METHODS

- Using operator[] and at() function to randomly access a vector makes a slight difference in quicksort

QUICKSORT – RELEASE MODE – USING [] OPERATOR, NO WARNINGS



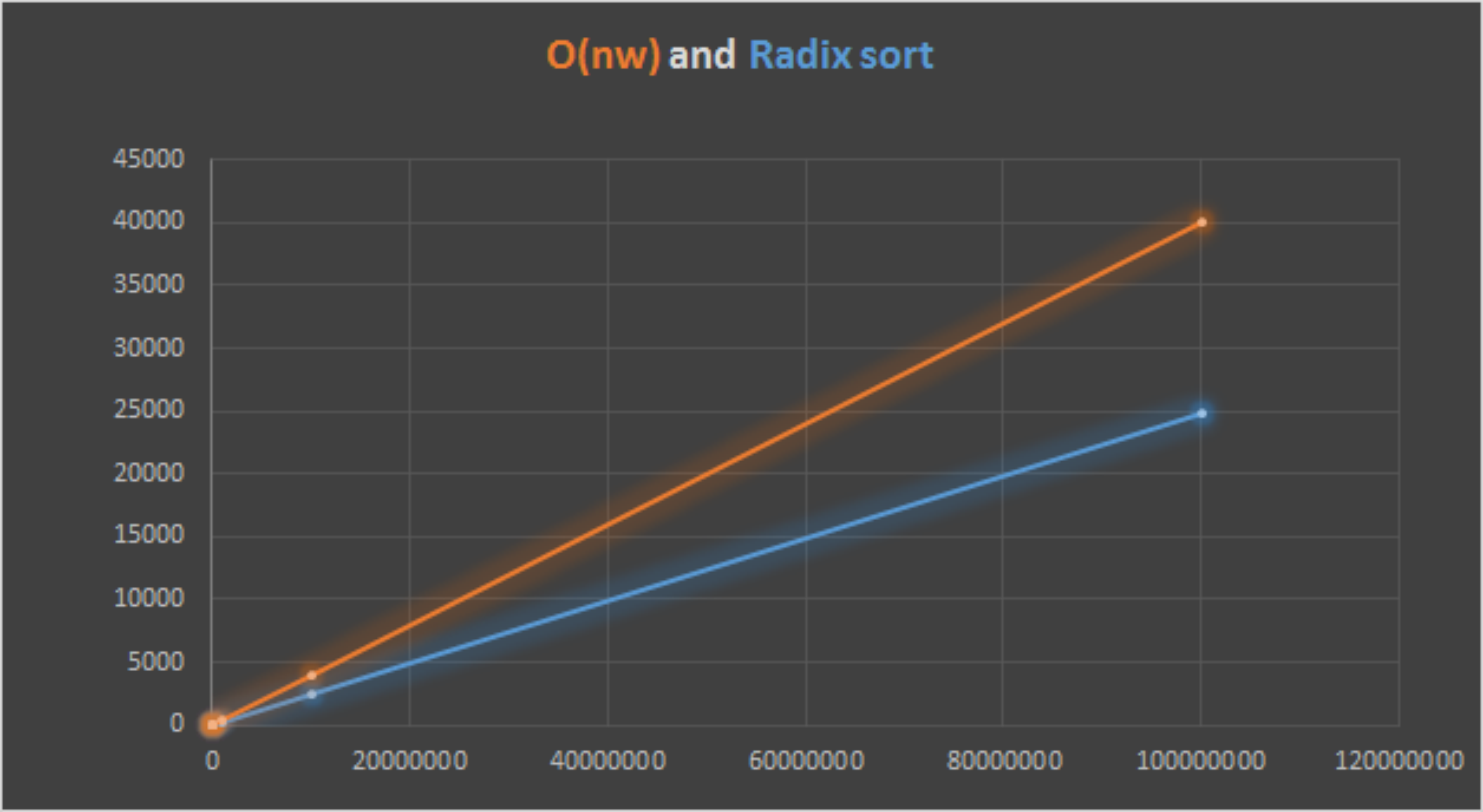
QUICKSORT - RELEASE MODE - USING AT() FUNCTION, NO WARNINGS



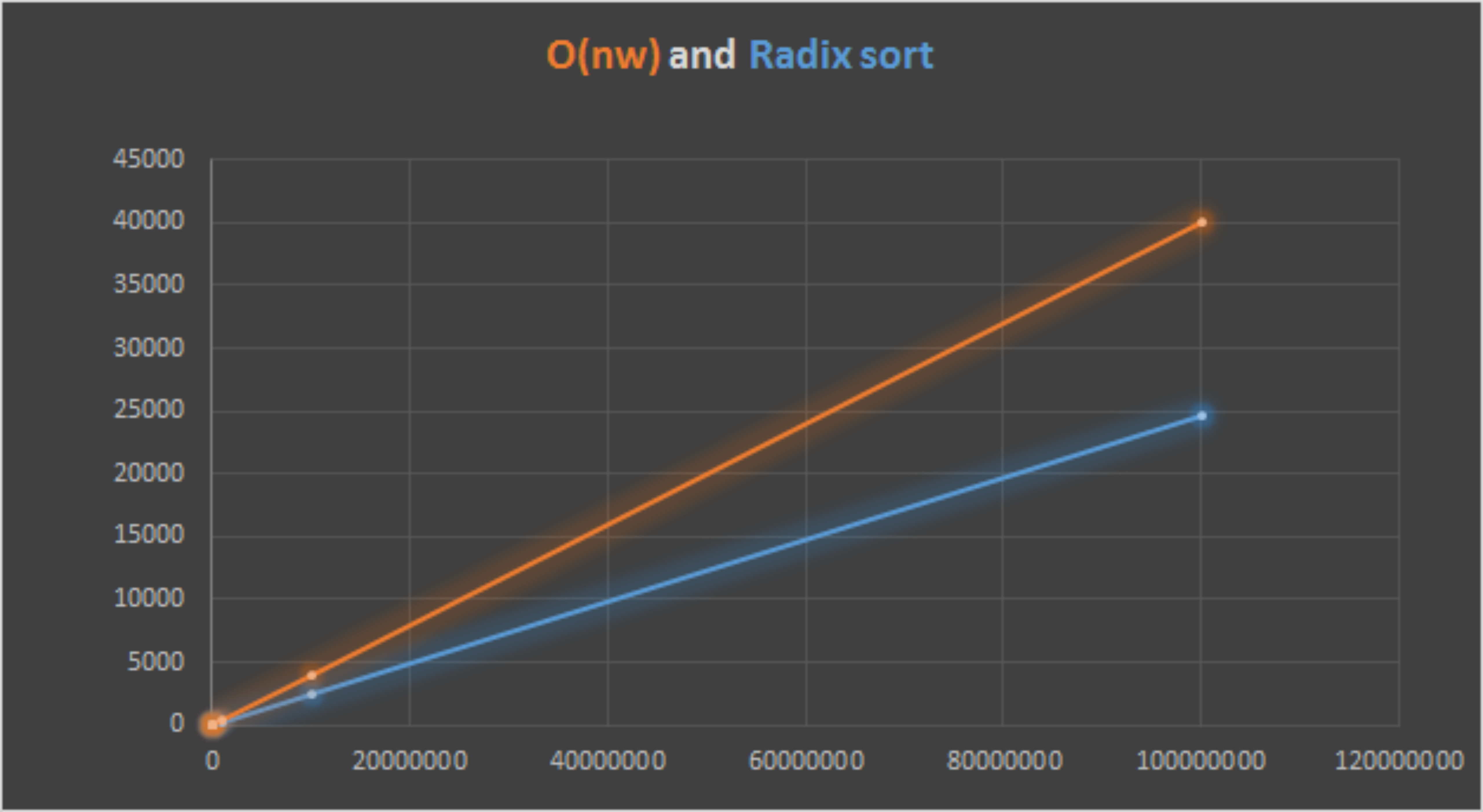
RESULTS - DIFFERENT RANDOM ACCESS METHODS

- Using operator[] and at() function to randomly access a vector makes no difference in radix sort

RADIX SORT- RELEASE MODE – USING [] OPERATOR, NO WARNINGS



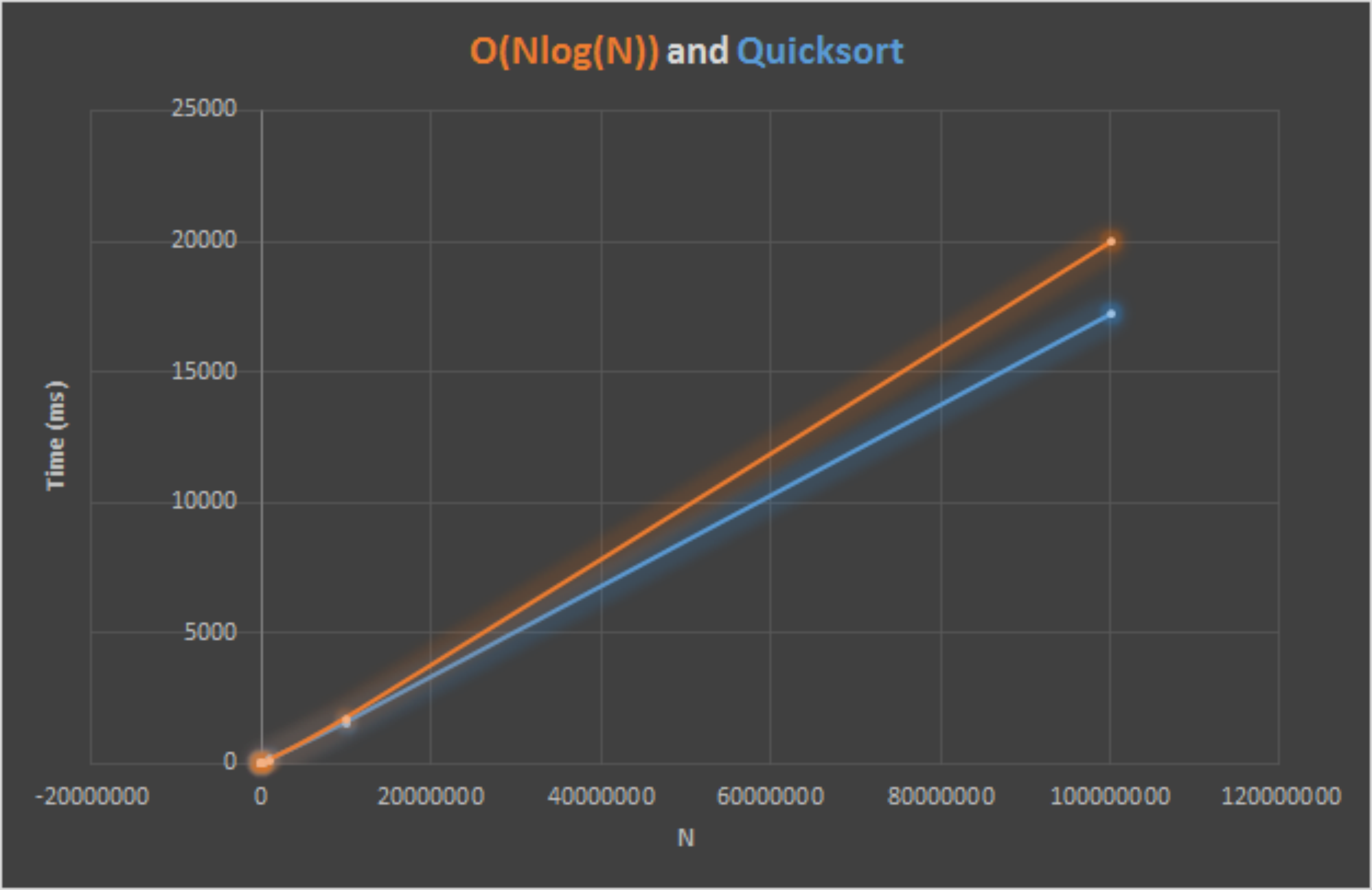
RADIX SORT- RELEASE MODE - USING AT() FUNCTION, NO WARNINGS



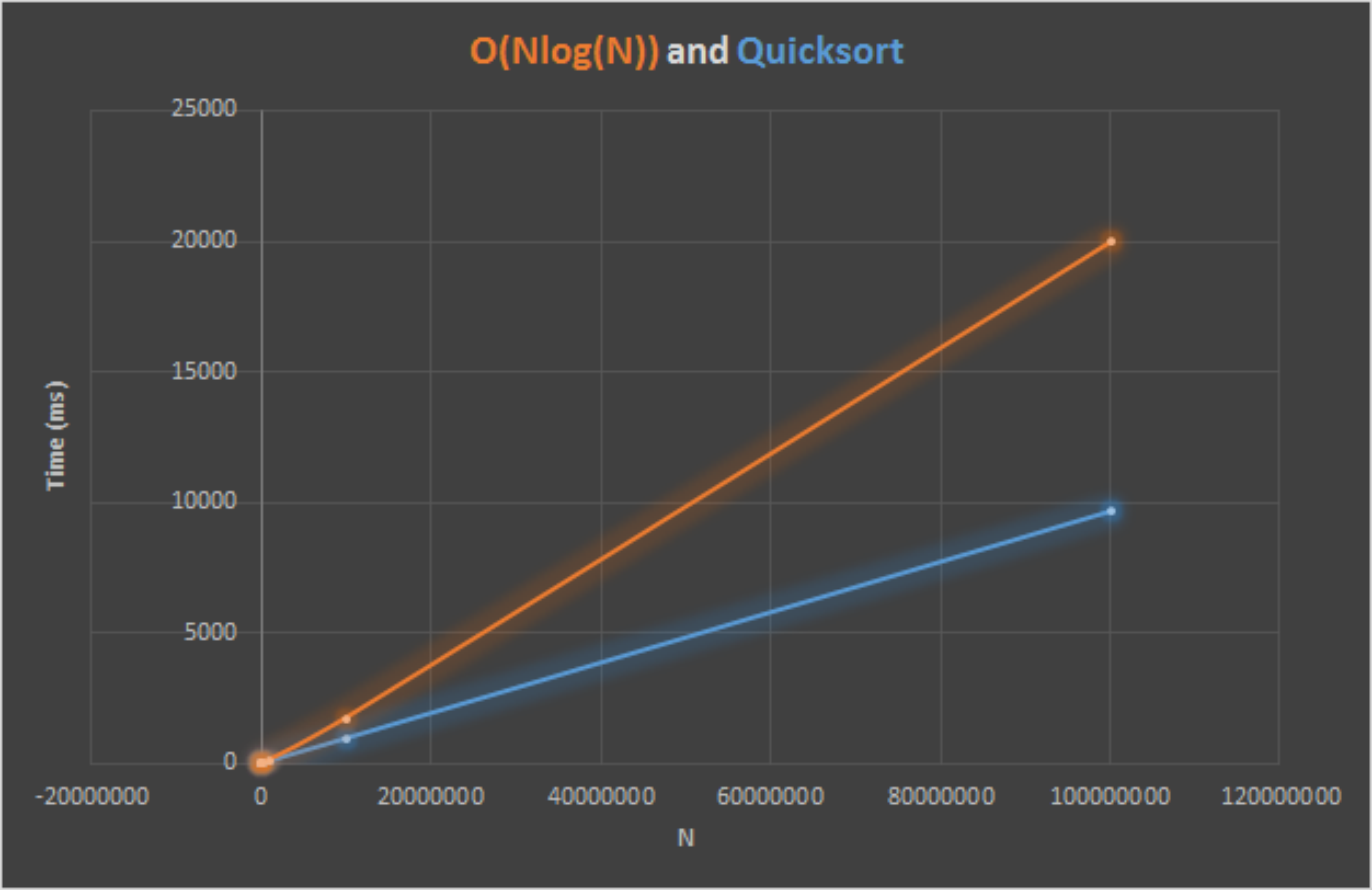
RESULTS - WARNINGS

- Getting rid of warnings - 1.77x faster quick sorting

QUICKSORT - RELEASE MODE - USING AT() OPERATOR, WARNINGS



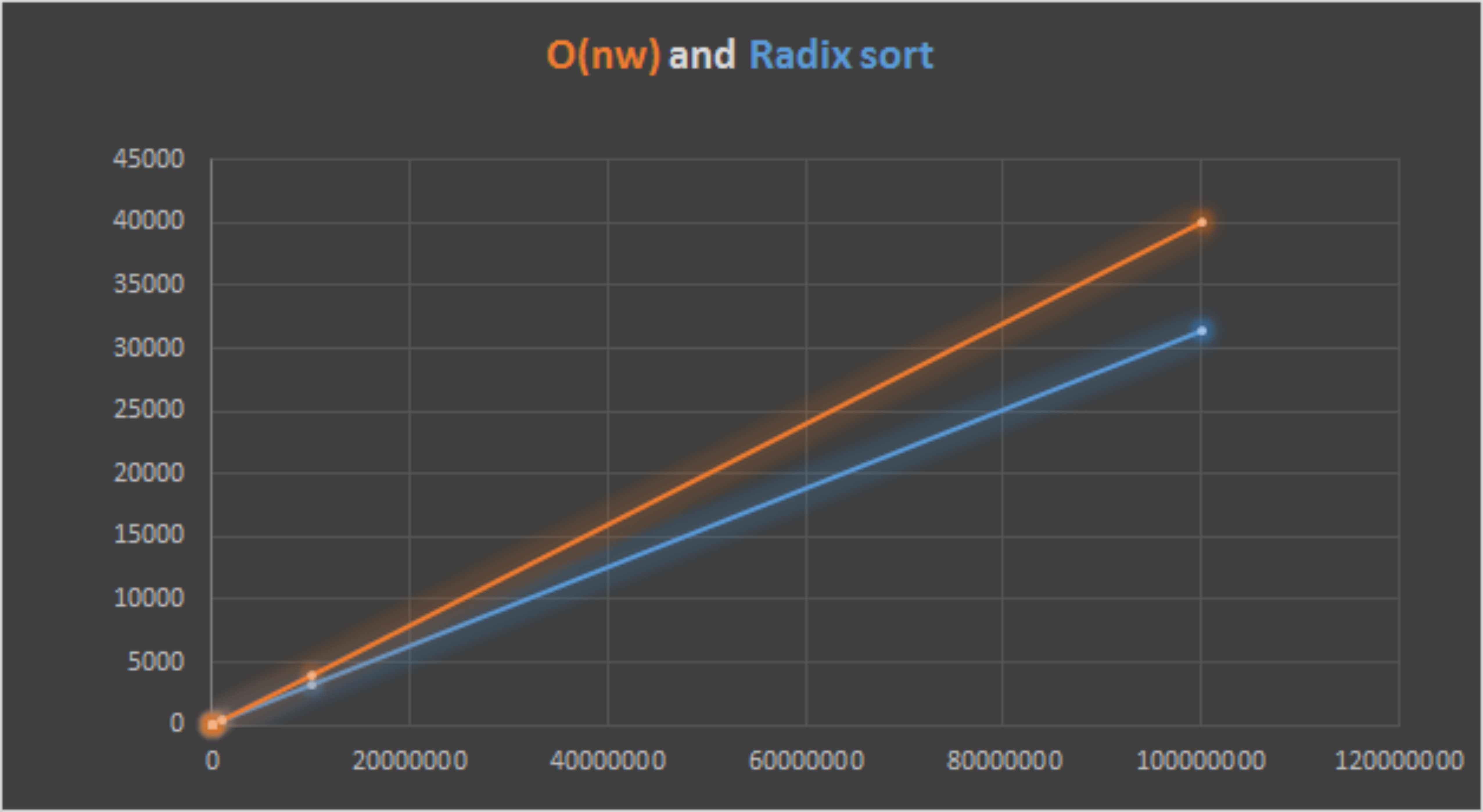
QUICKSORT - RELEASE MODE - USING AT() FUNCTION, NO WARNINGS



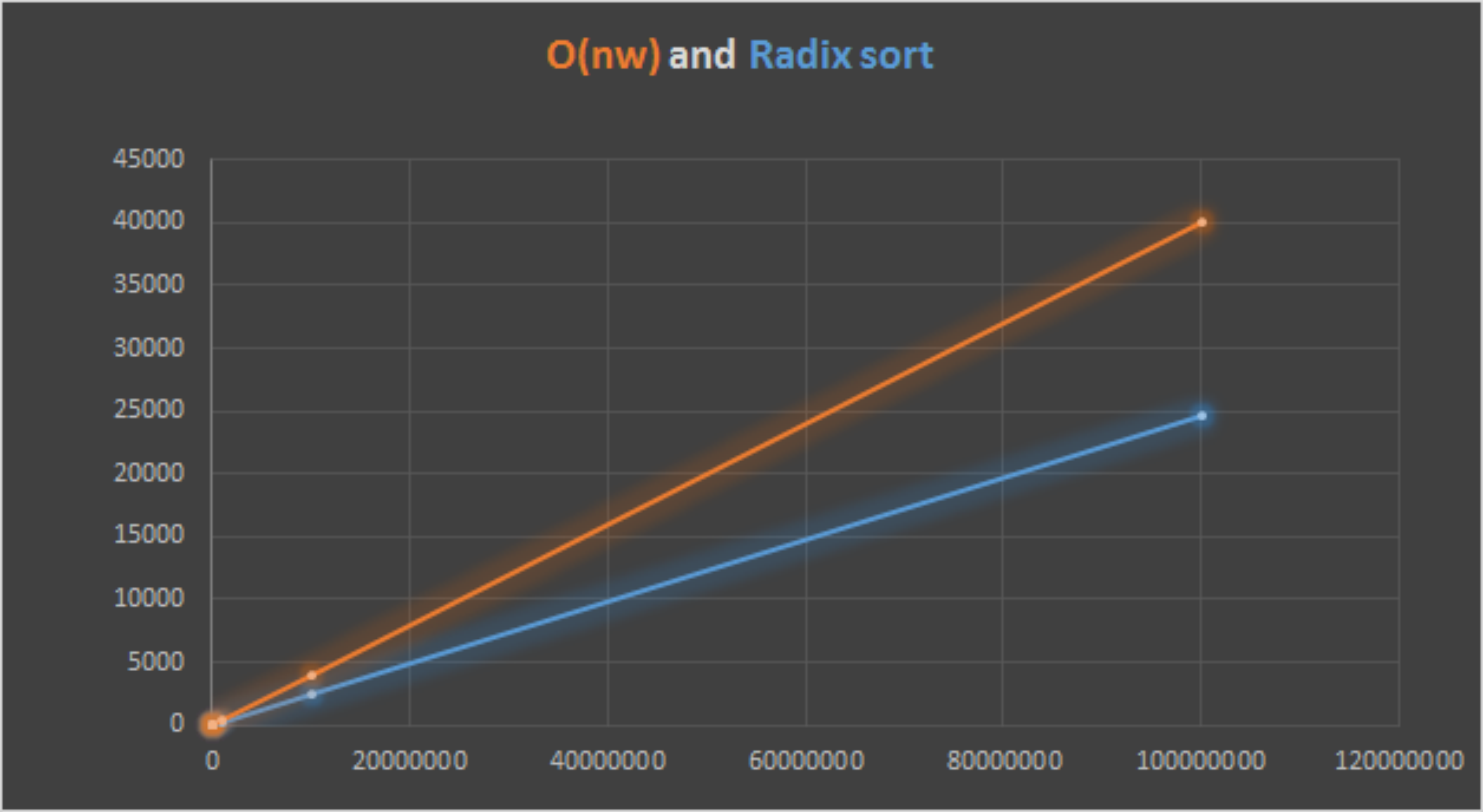
RESULTS - WARNINGS

- Getting rid of warnings - almost 1.27x faster radix sorting

RADIX SORT- RELEASE MODE – USING AT() OPERATOR, WARNINGS



RADIX SORT- RELEASE MODE - USING AT() FUNCTION, NO WARNINGS



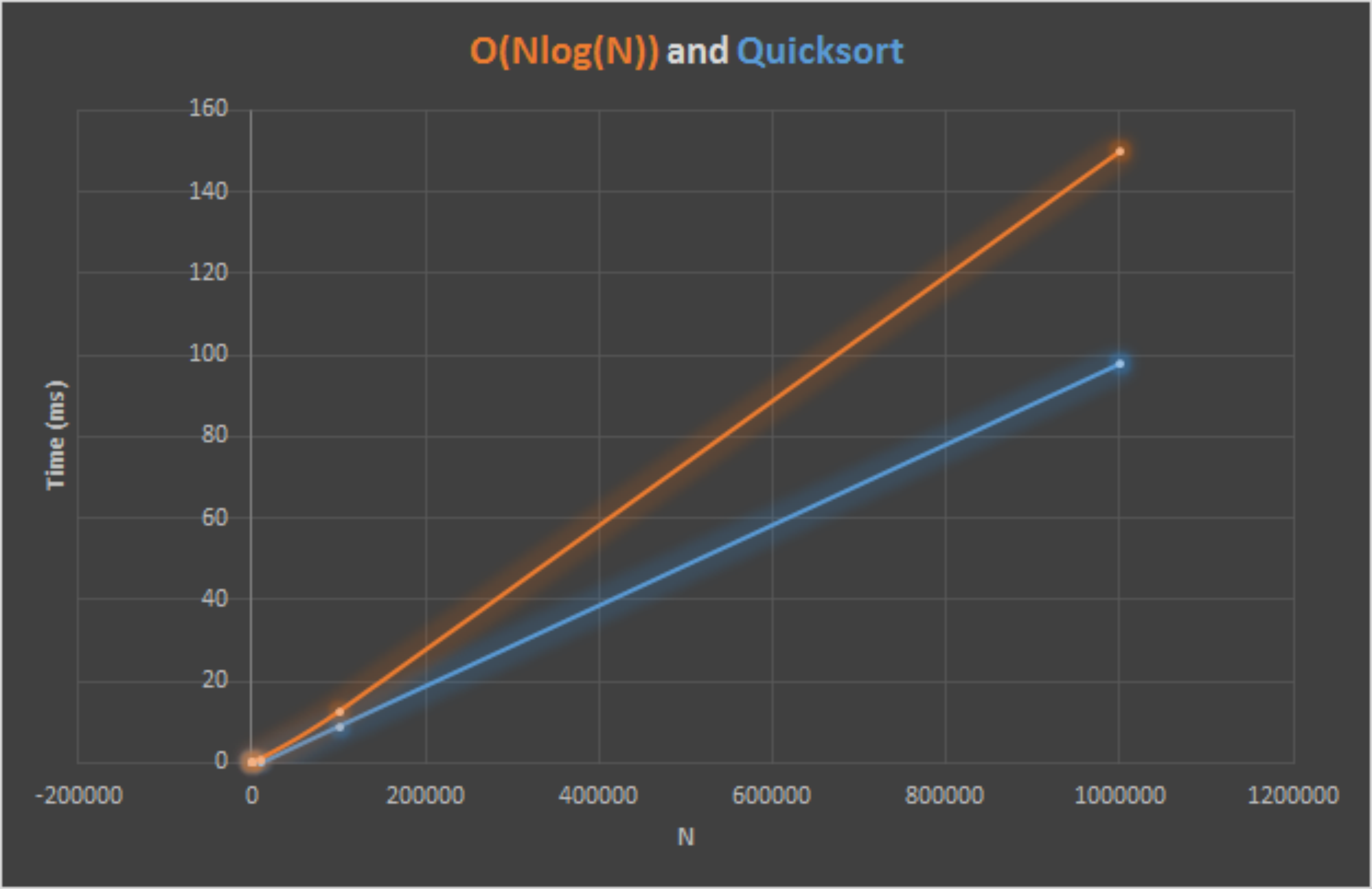
SORTING IN DEBUG MODE

- Up to 100x slower
- Inefficient for N > 10e6 for both methods

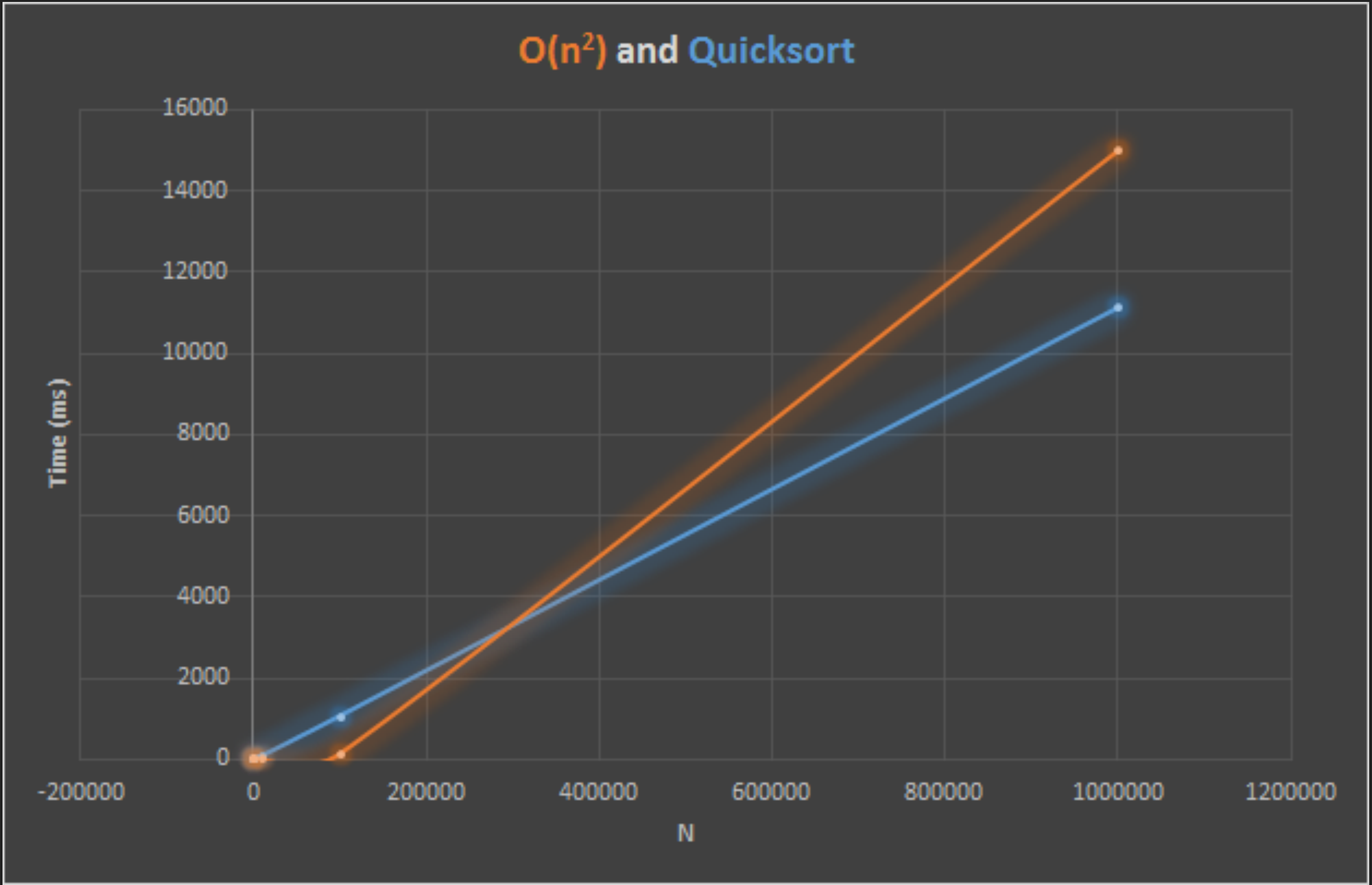
SORTING IN DEBUG MODE

- Quicksort
- Release mode
- Using at() operator
- No warnings
- N ∈ [0, 10E6]

QUICKSORT – RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, N ∈ [0, 10E6]



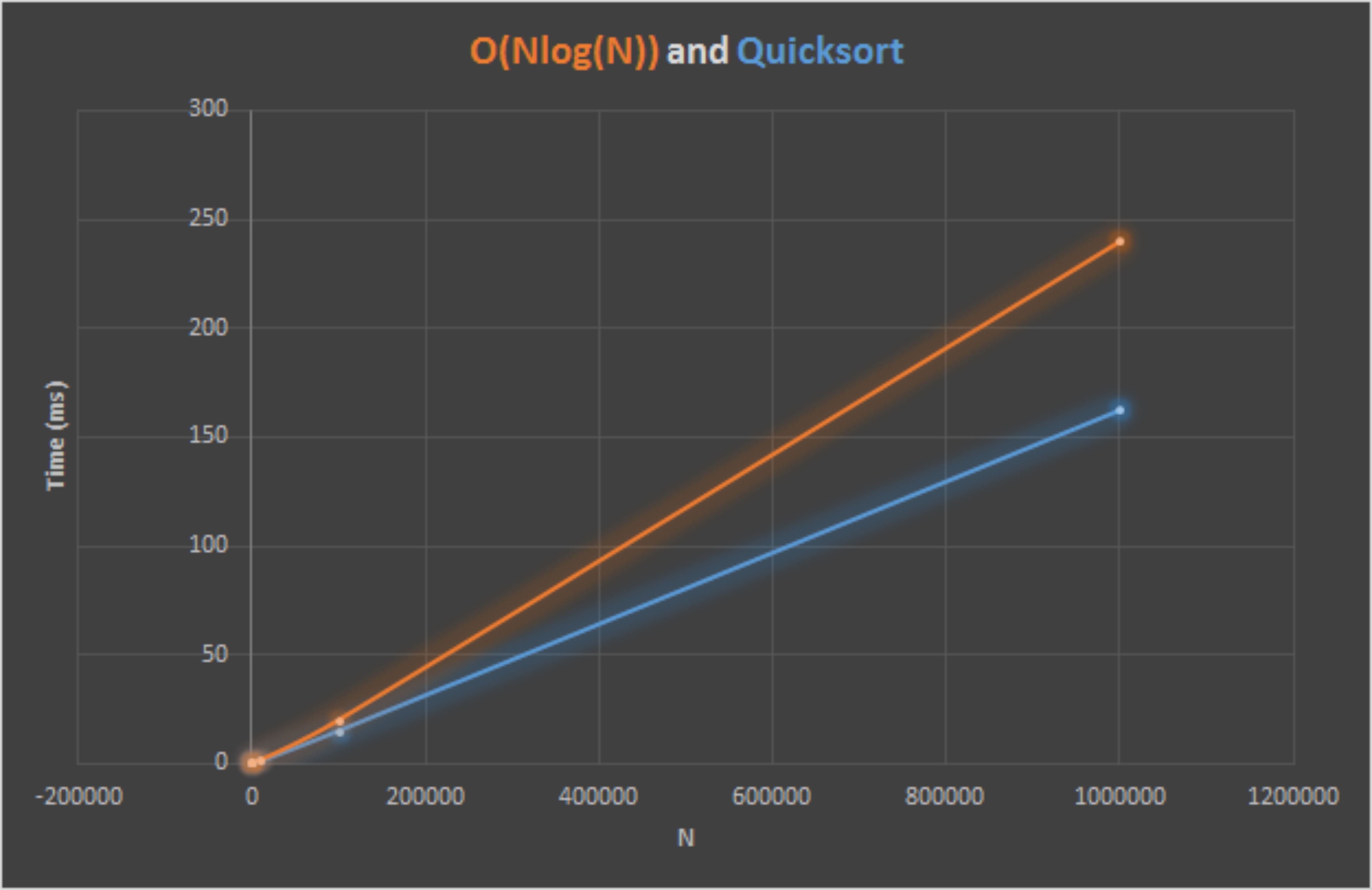
QUICKSORT- DEBUG MODE – USING AT() FUNCTION, NO WARNINGS, N ∈ [0, 10E6]



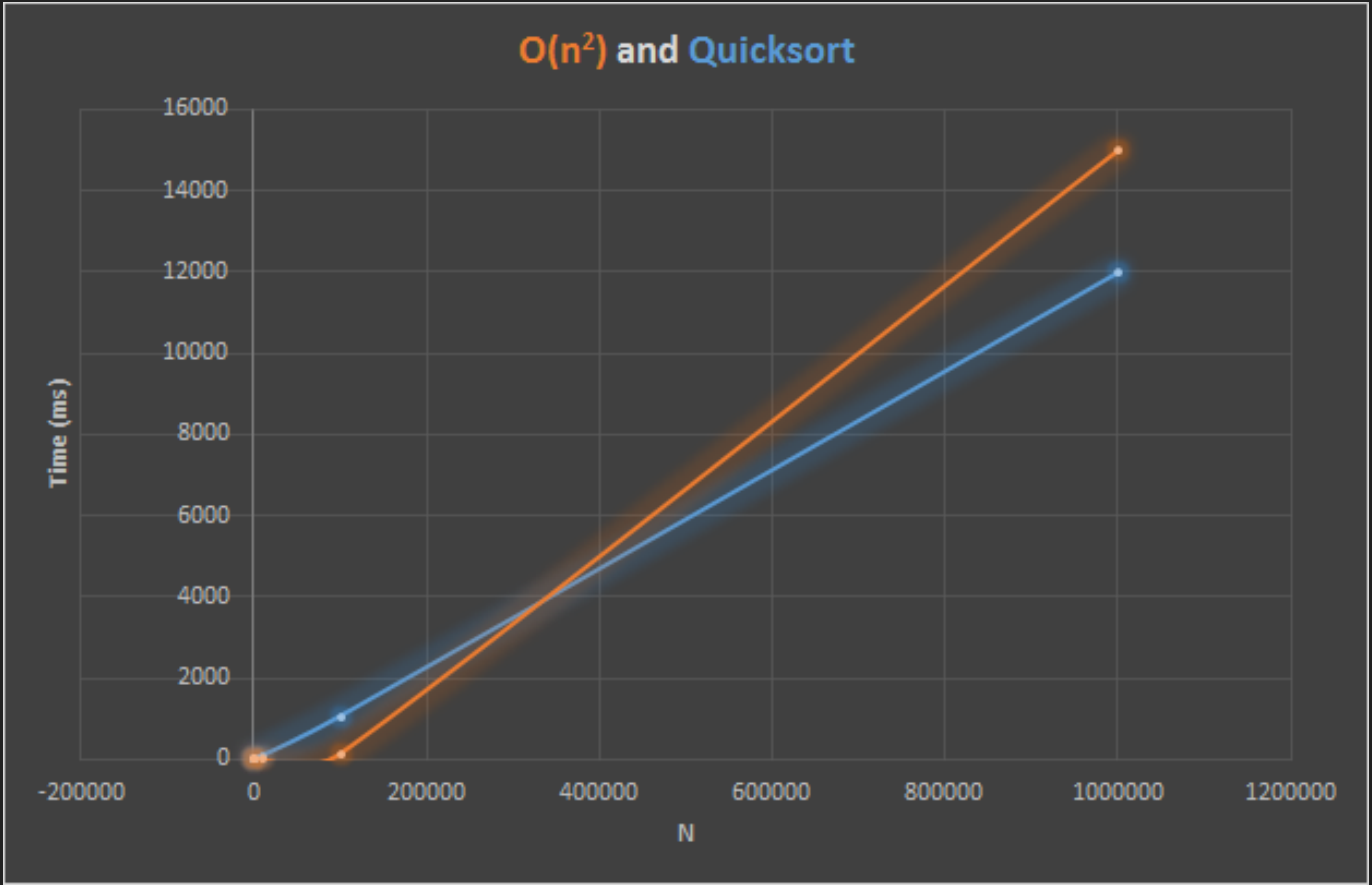
SORTING IN DEBUG MODE

- Quicksort-
- Release mode
- Using at() operator
- Warnings
- N ∊ [0, 10e6]

QUICK SORT- RELEASE MODE - USING AT() FUNCTION, WARNINGS, N ∈ [0, 10E6]



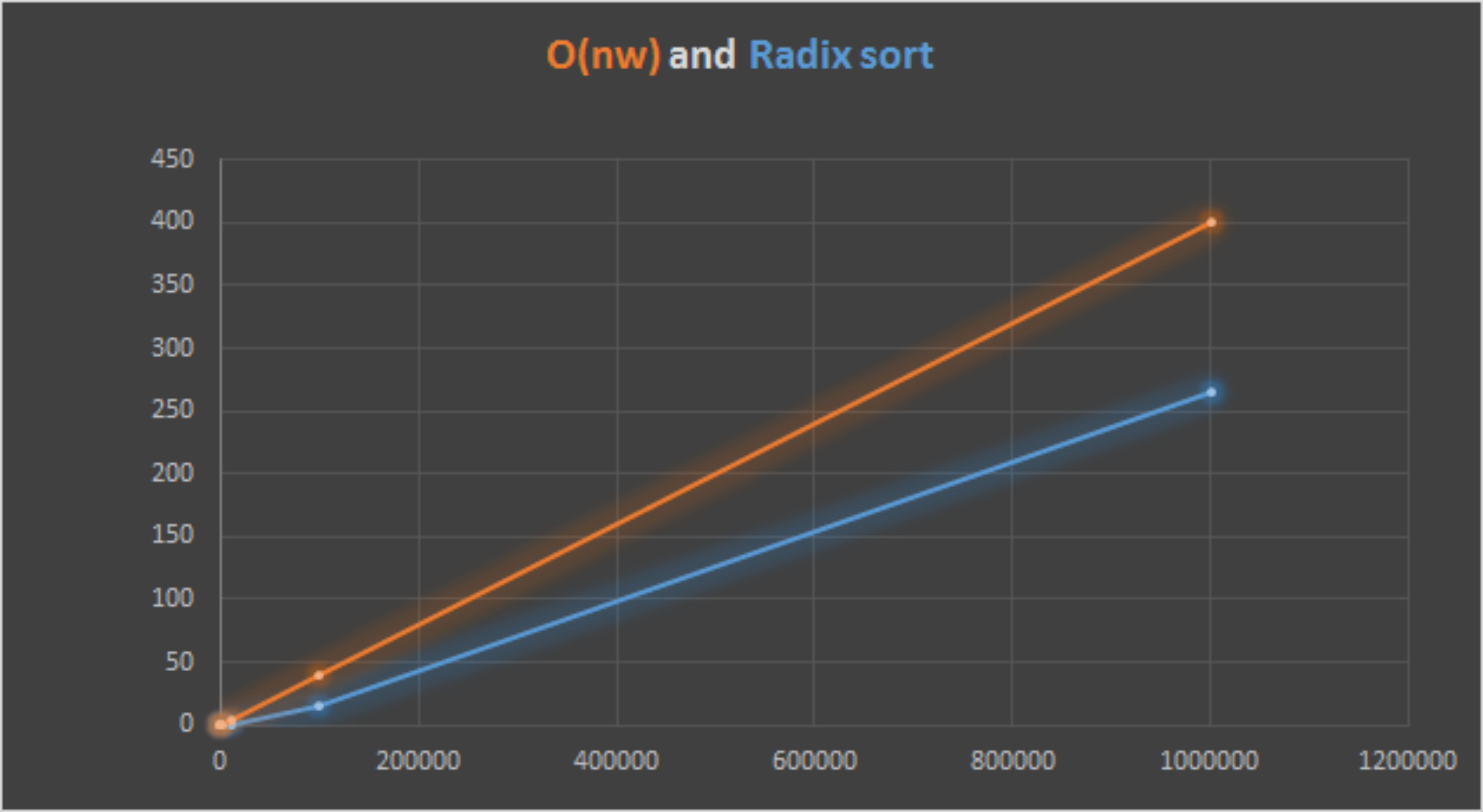
QUICK SORT- DEBUG MODE - USING AT() FUNCTION, WARNINGS, N ∈ [0, 10E6]



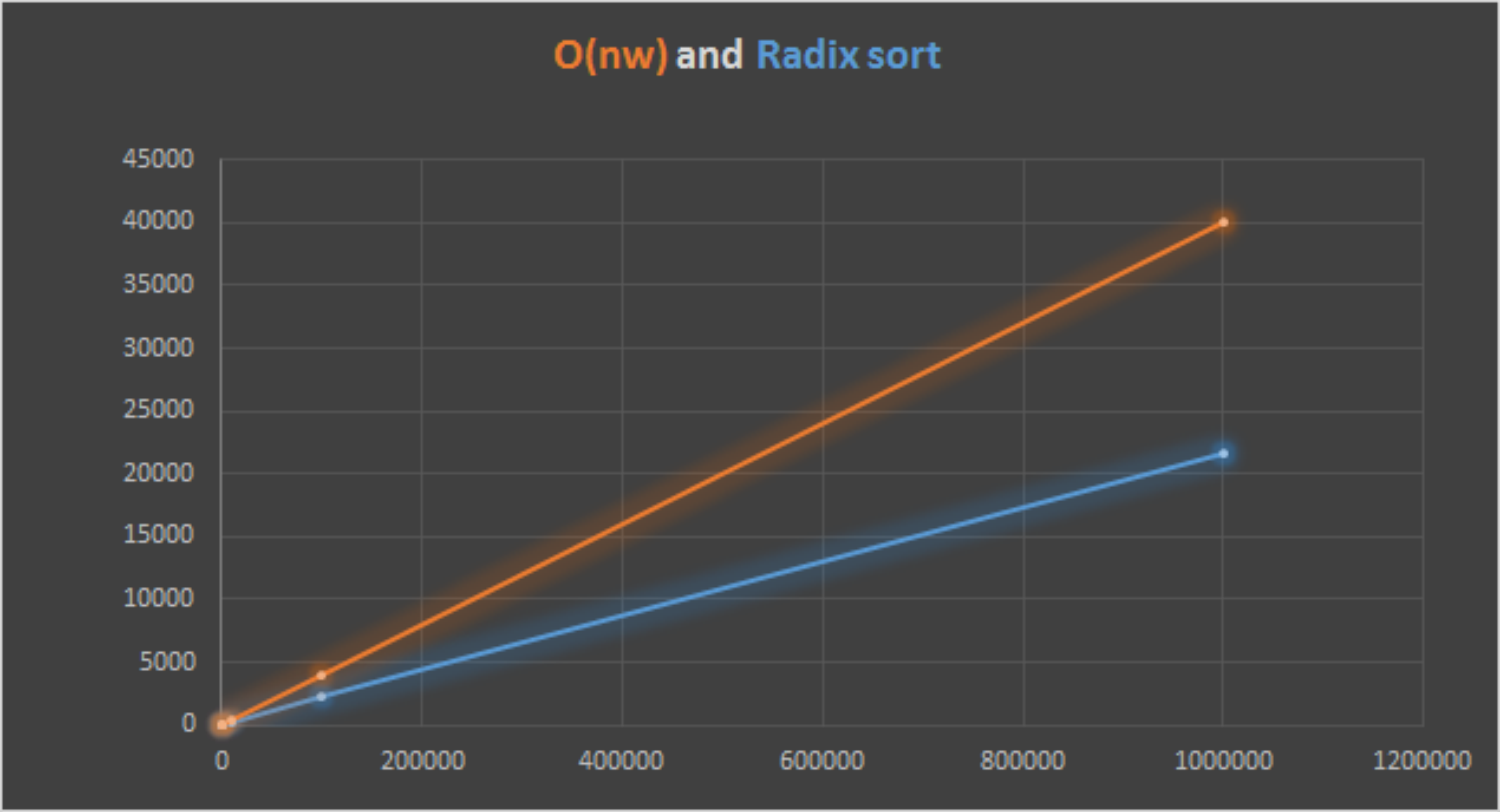
SORTING IN DEBUG MODE

- Radix sort
- Release mode
- Using at() operator
- No warnings
- N ∊ [0, 10e6]

RADIX SORT- RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, N ∈ [0, 10E6]



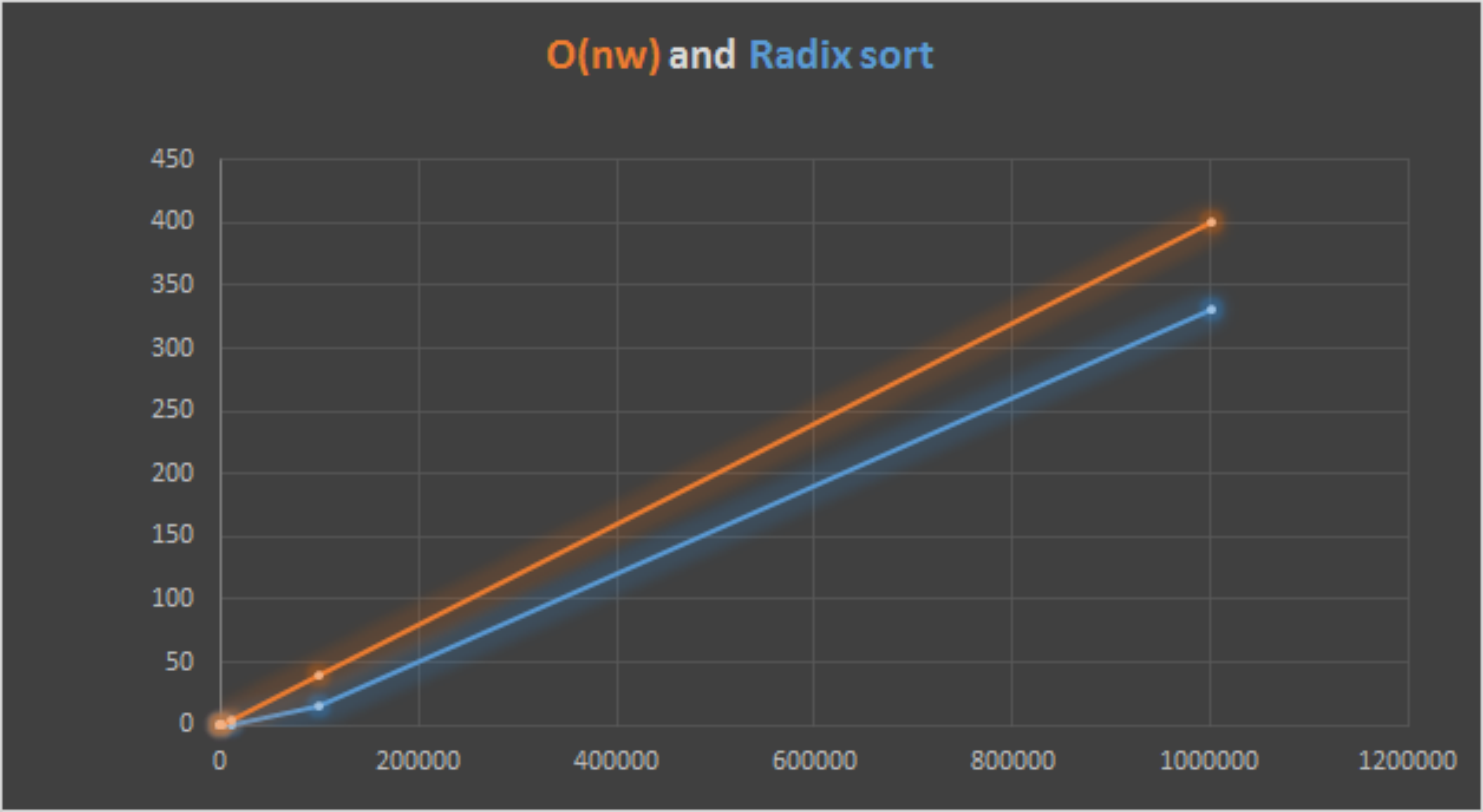
RADIX SORT- DEBUG MODE - USING AT() FUNCTION, NO WARNINGS, N ∈ [0, 10E6]



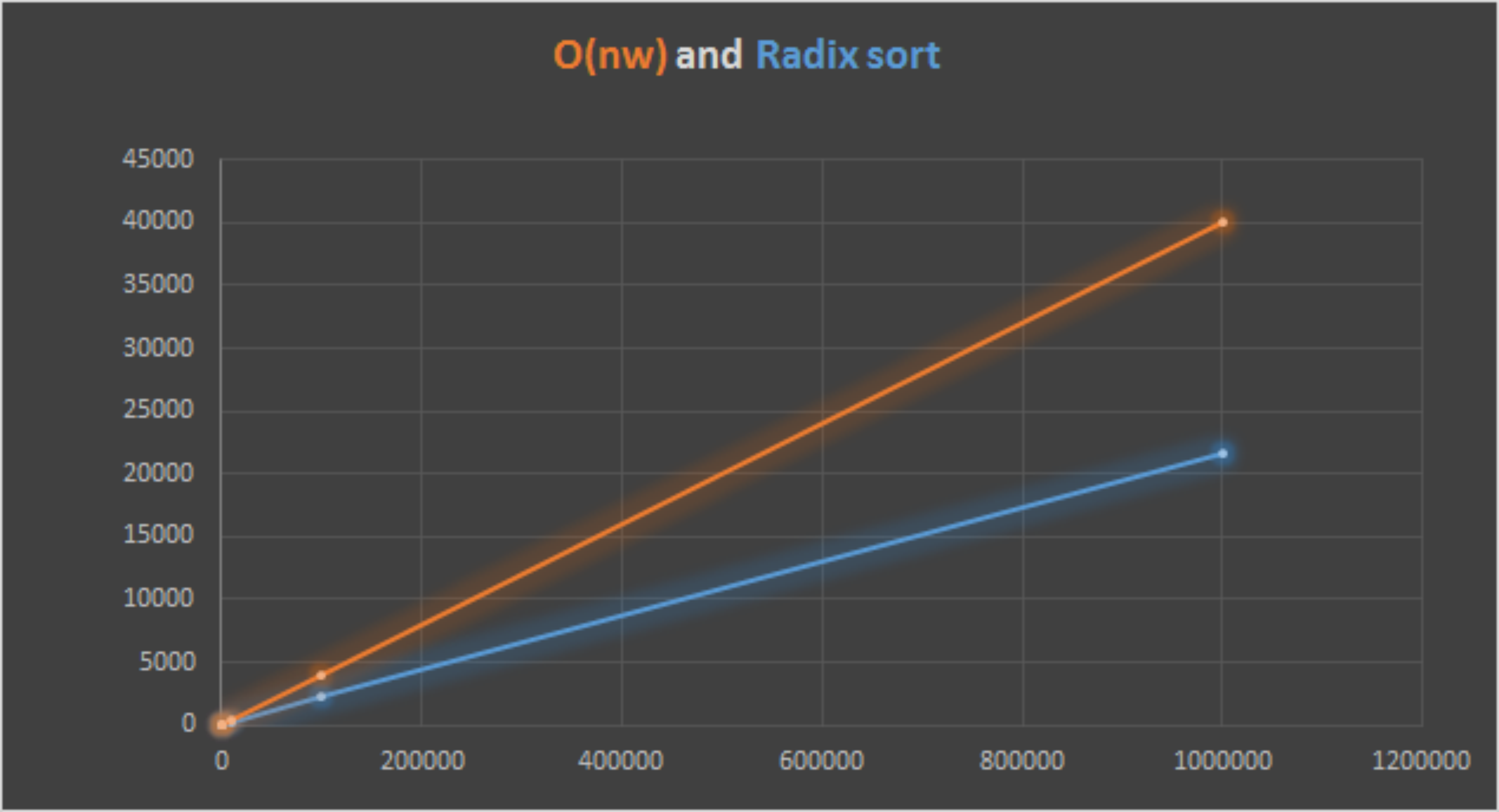
SORTING IN DEBUG MODE

- Radix sort
- Release mode
- Using at() operator
- Warnings
- $N \in [0, 10e6]$

RADIX SORT- RELEASE MODE - USING AT() FUNCTION, WARNINGS, N ∈ [0, 10E6]



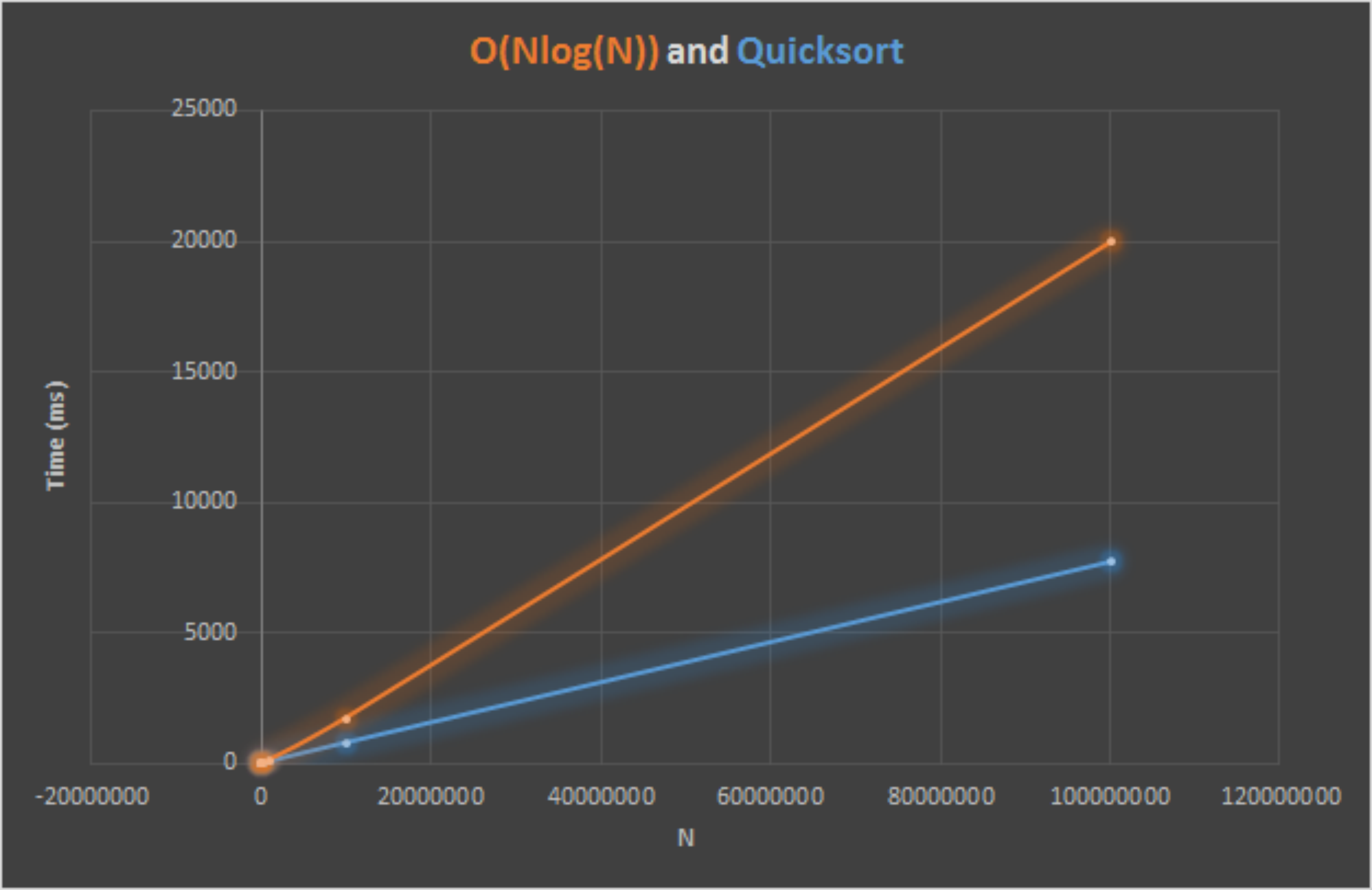
RADIX SORT- DEBUG MODE - USING AT() FUNCTION, WARNINGS, N ∈ [0, 10E6]



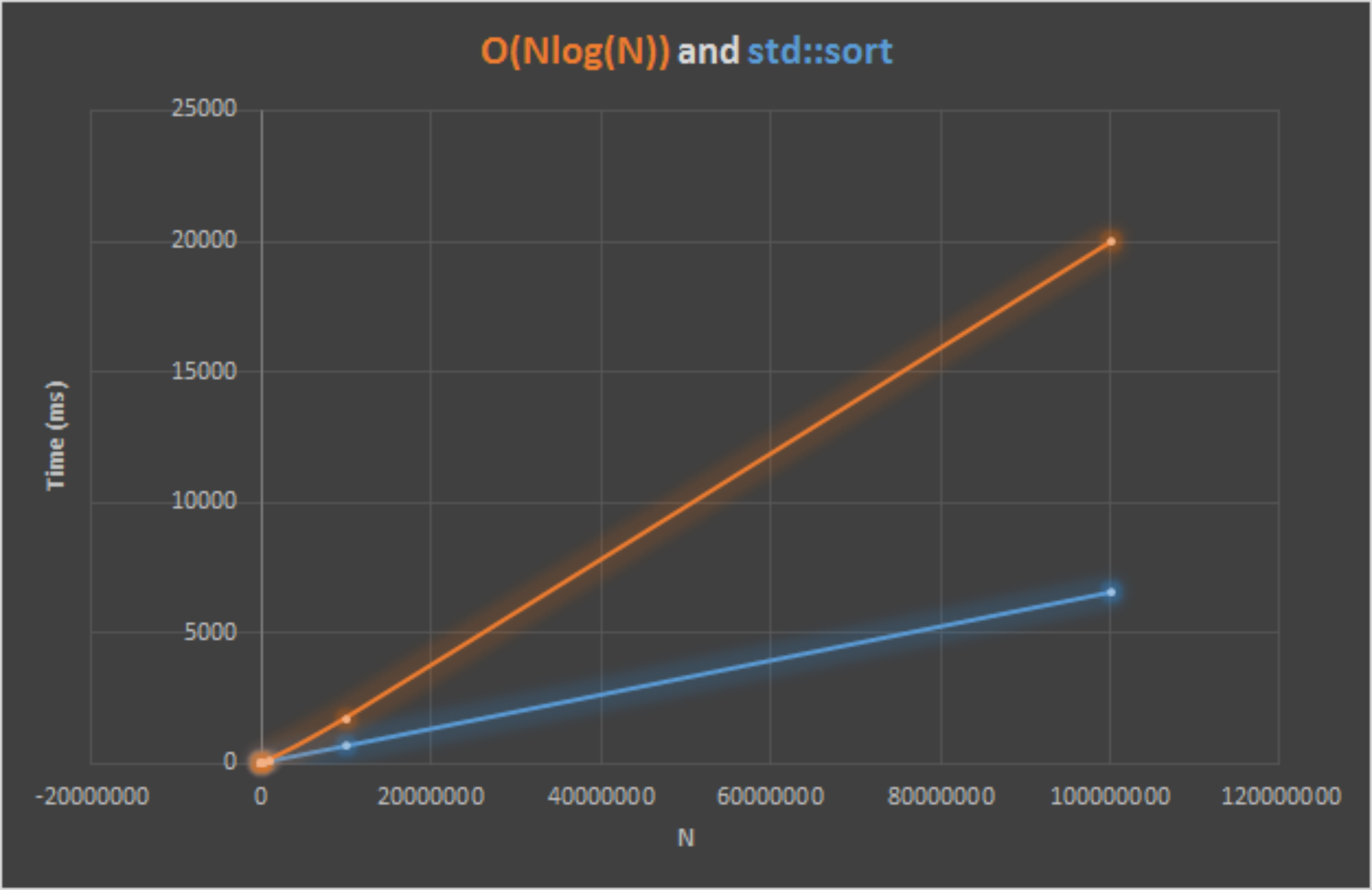
RESULTS - MY BEST RESULTS VS. STD::SORT

- quicksort - release mode - using [] operator, no warnings
- std::sort - release mode - no warnings

QUICKSORT – RELEASE MODE – USING [] OPERATOR, NO WARNINGS



STD::SORT - RELEASE MODE - NO WARNINGS



PROFILING - MACBOOK PRO

- Turned off all output for CPU sampling
- Inclusive samples for:
- radix sort: 42.04%
- quicksort: 19.61%
- std::sort: 16.15%
- populating vector with random numbers: 2.13%

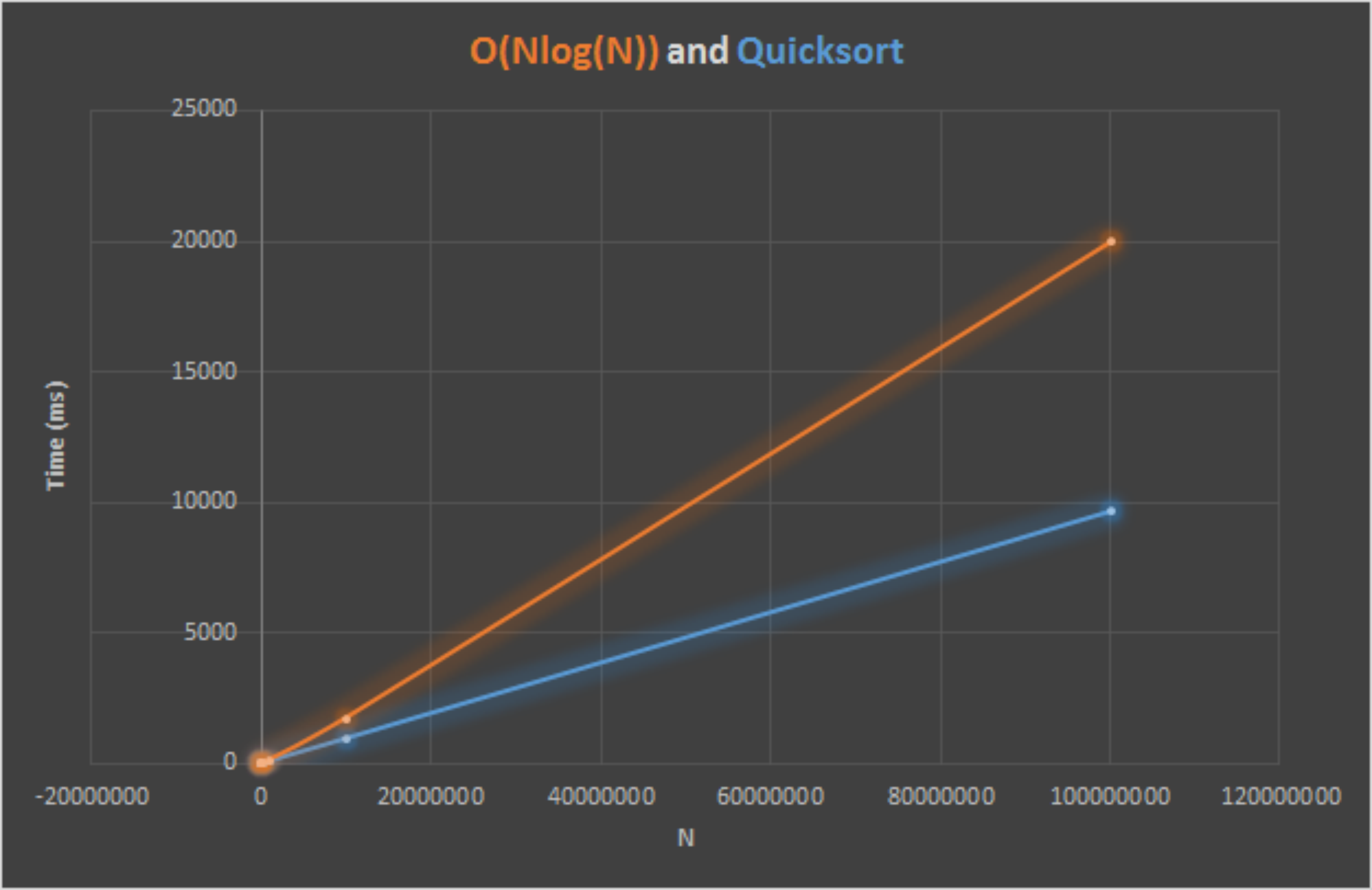
RESULTS – UNI PC VS. MACBOOK PRO

- quicksort - release mode - using at() function, no warnings
- radix sort - release mode - using at() function, no warnings
- std::sort - release mode - no warnings

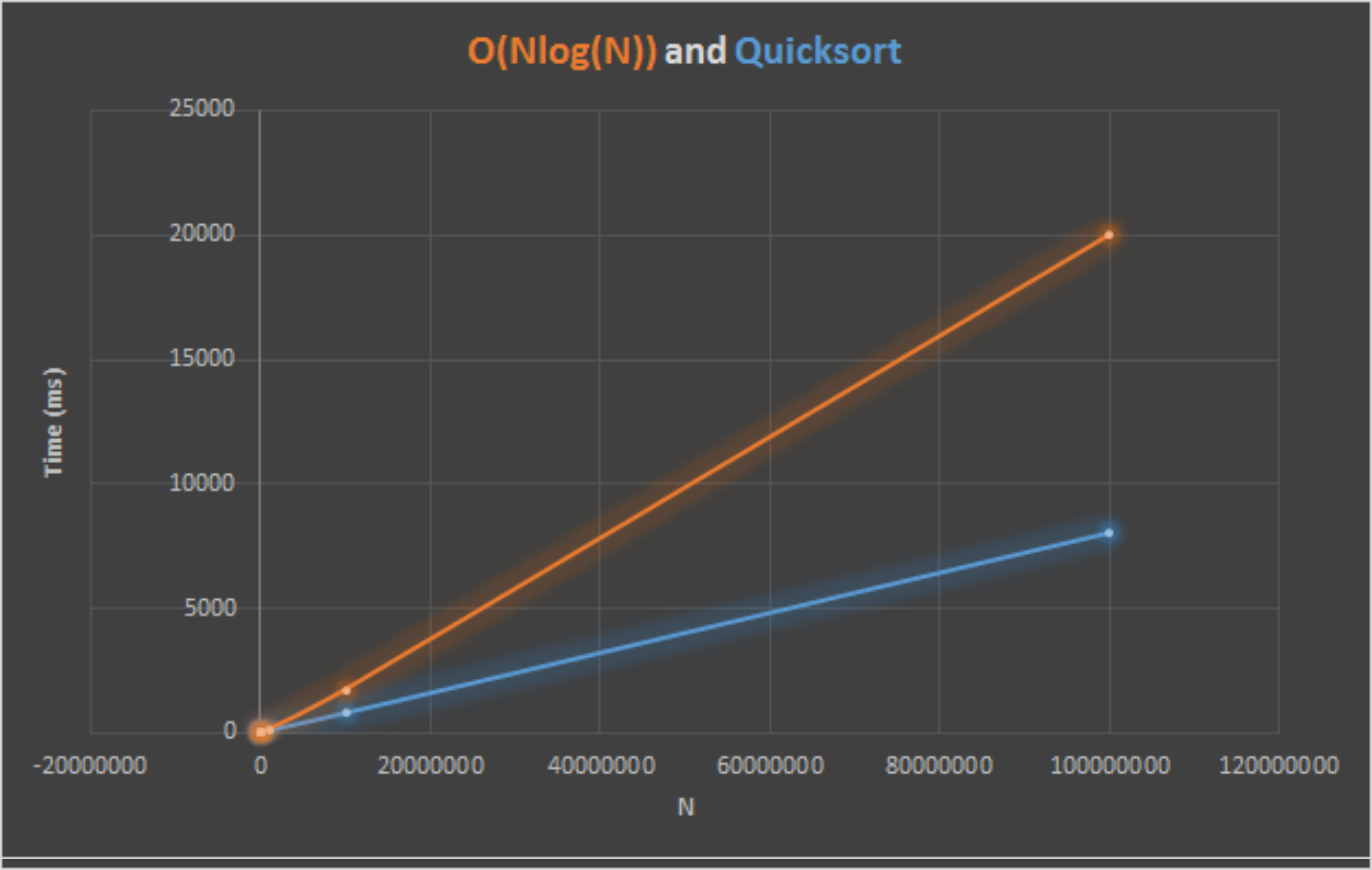
RESULTS – UNI PC VS. MACBOOK PRO

- quicksort - release mode - using at() function, no warnings

QUICKSORT – RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, UNI PC



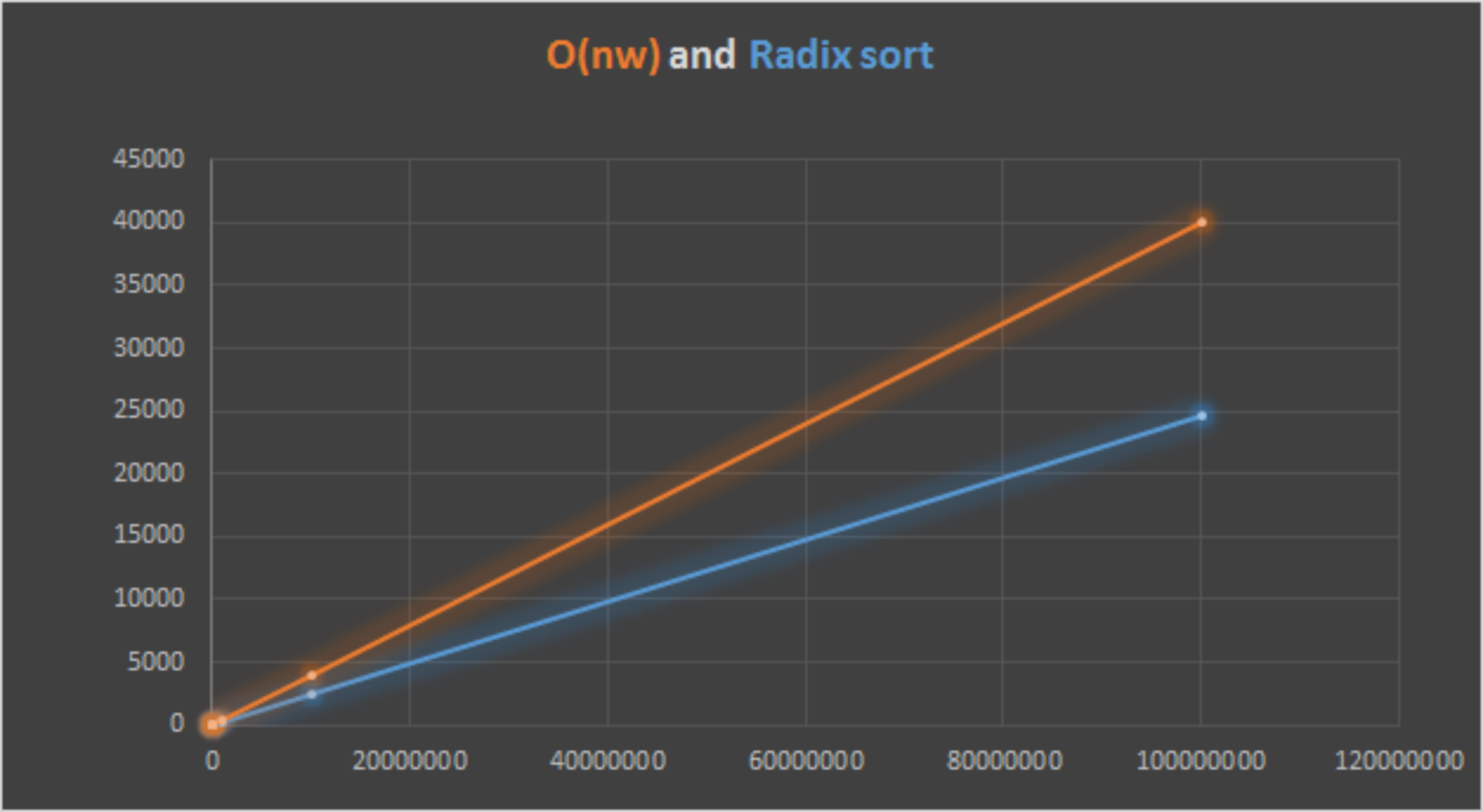
QUICKSORT – RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, MACBOOK PRO



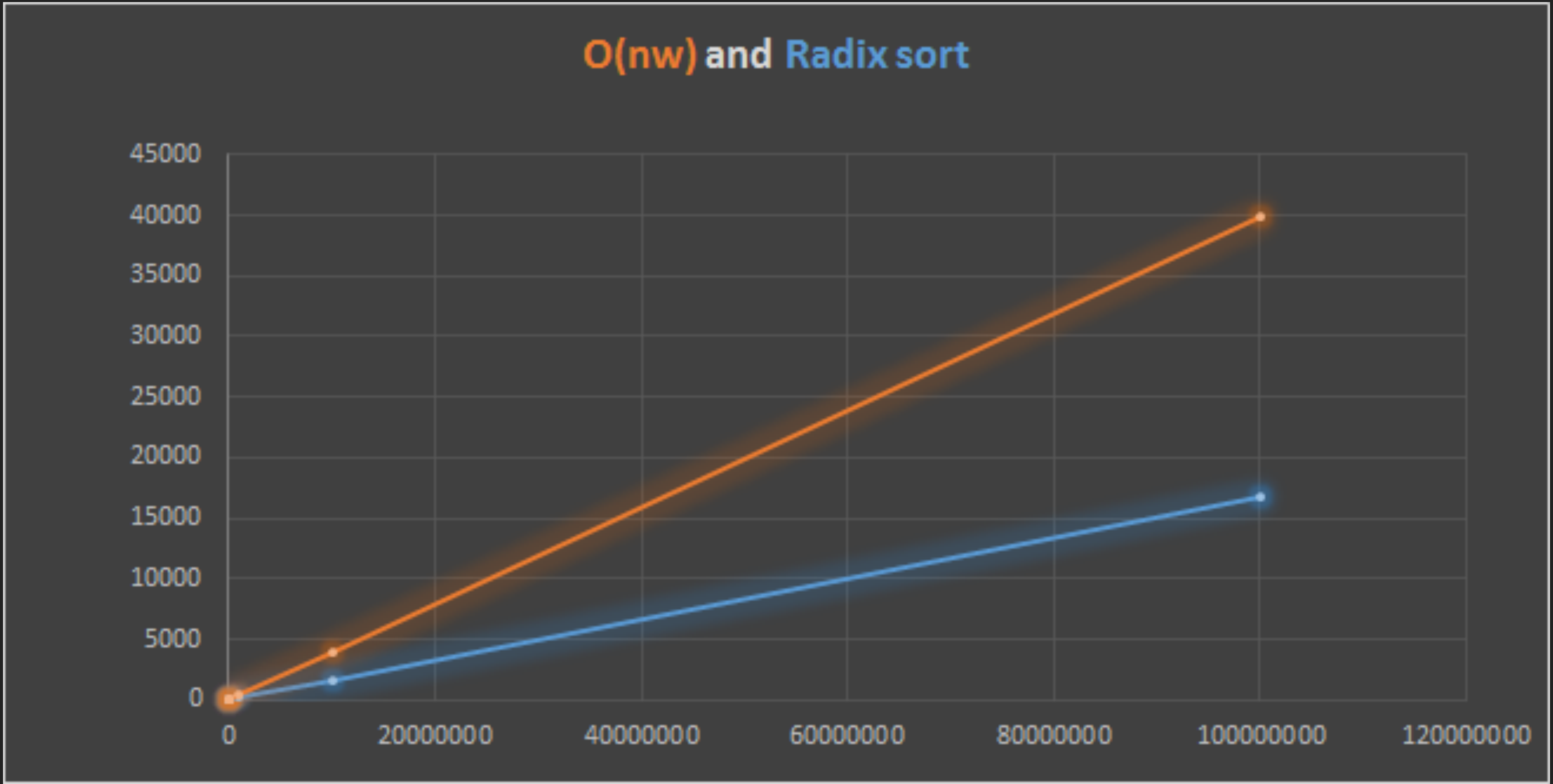
RESULTS – UNI PC VS. MACBOOK PRO

- radix sort - release mode - using at() function, no warnings

RADIX SORT- RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, UNI PC



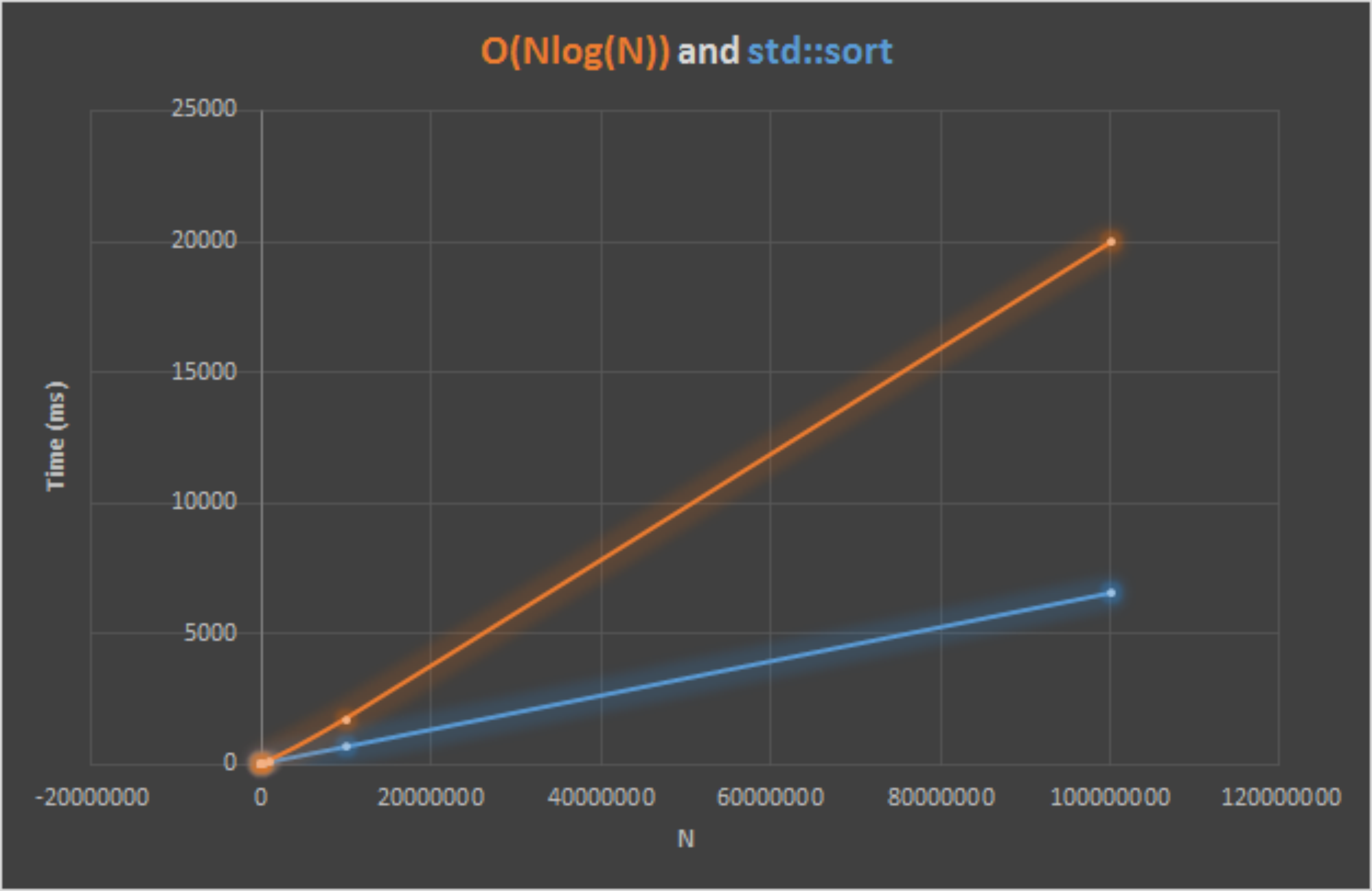
RADIX SORT- RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, MACBOOK PRO



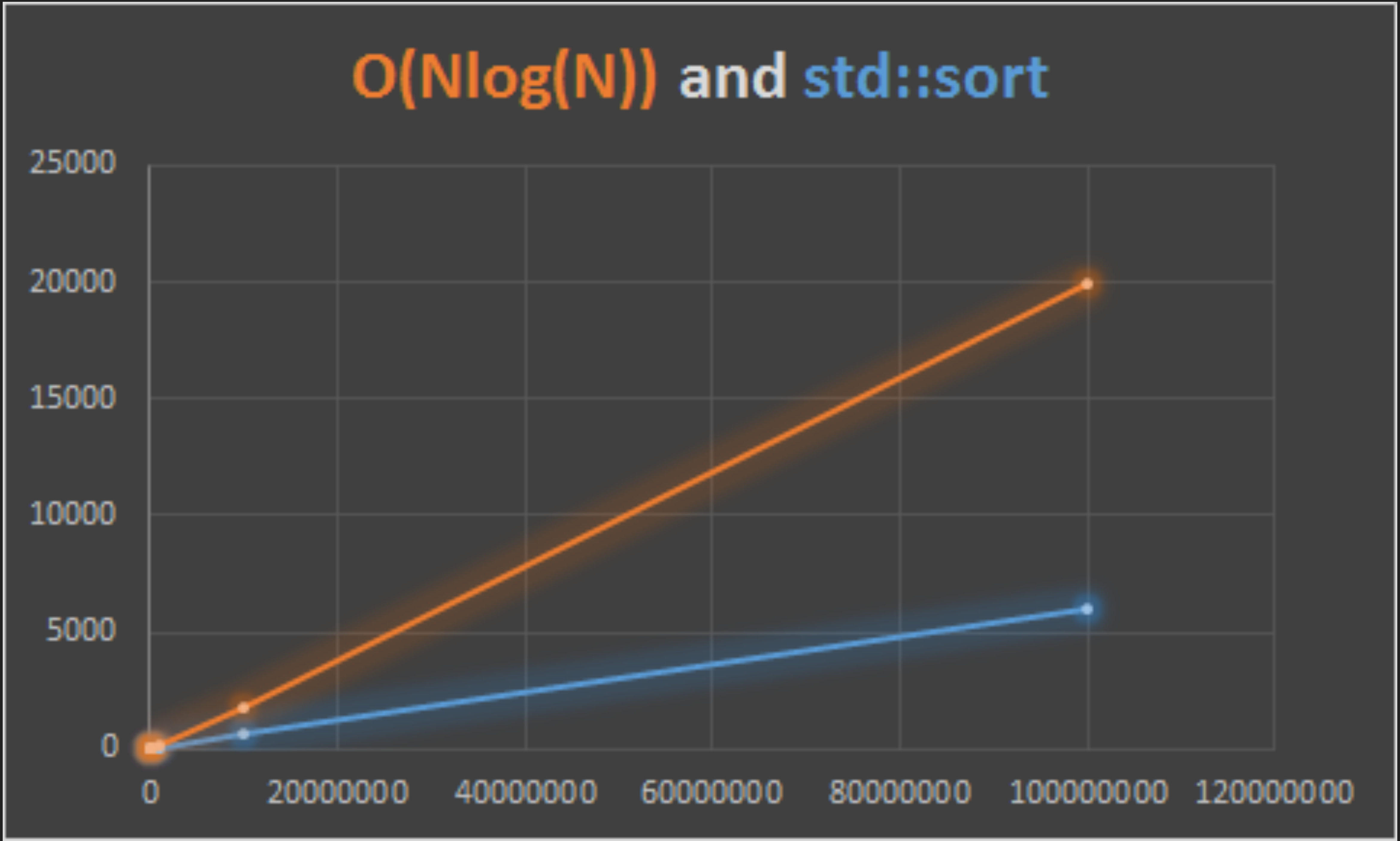
RESULTS – UNI PC VS. MACBOOK PRO

- std::sort - release mode - no warnings

STD::SORT – RELEASE MODE – NO WARNINGS, UNI PC



STD::SORT – RELEASE MODE – NO WARNINGS, MACBOOK PRO



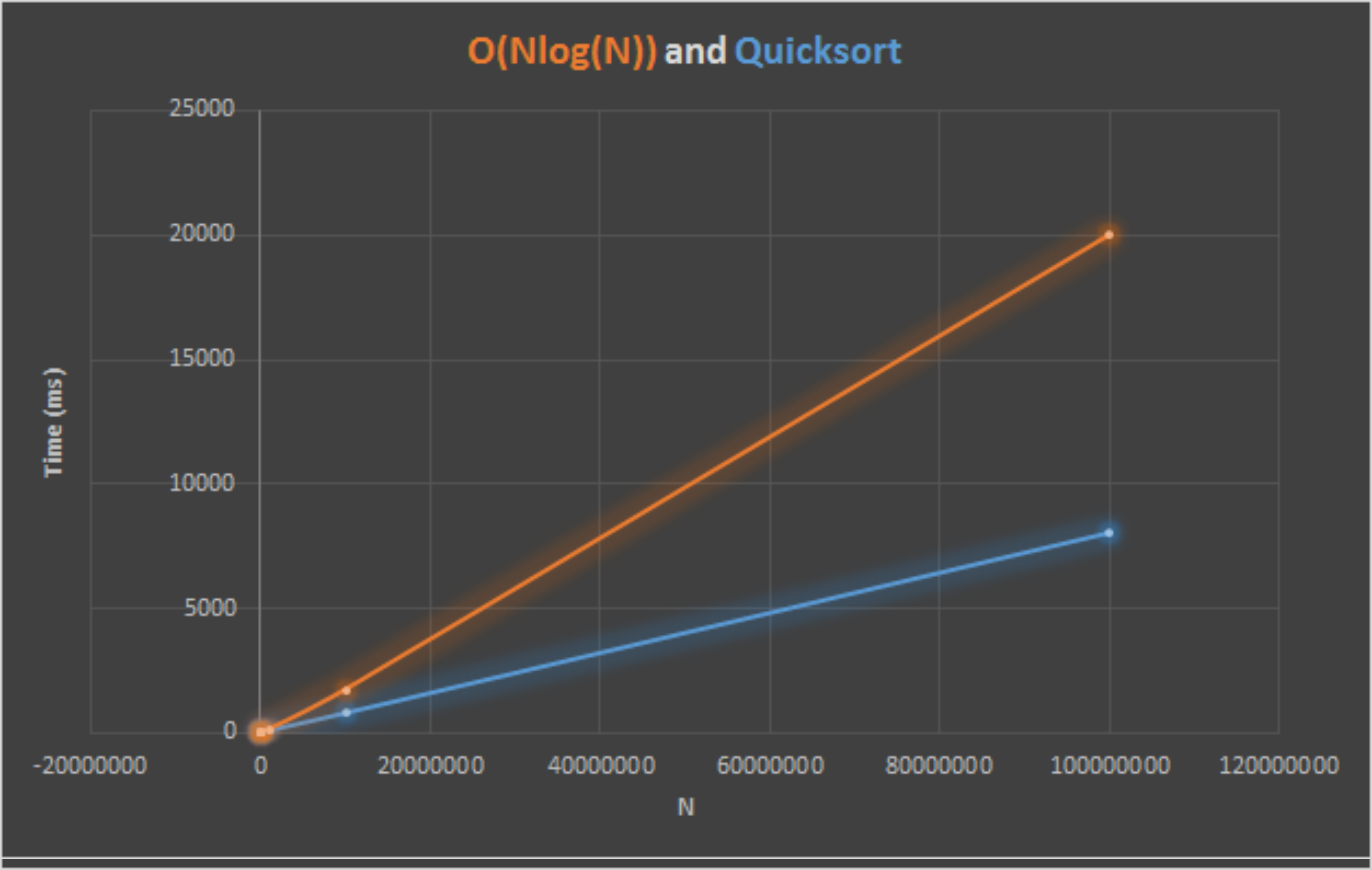
RESULTS - NOT-SORTED ARRAY VS. SORTED ARRAY (MACBOOK PRO)

- quicksort and sort perform better
  - quicksort - choosing pivot in the middle - prevents worst-case behaviour - $O(n^2)$ - on already sorted arrays
- radix sort slightly worse

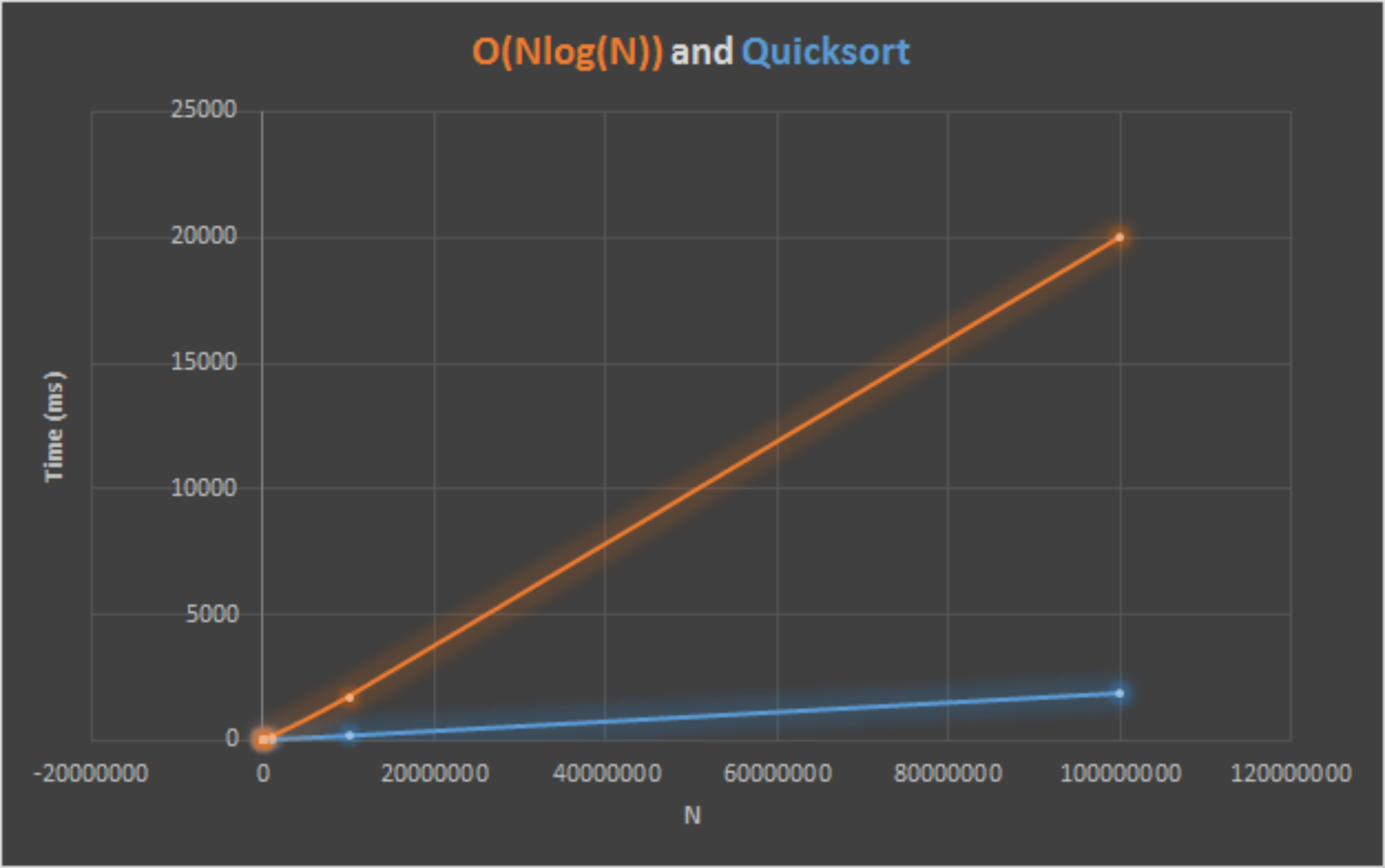
RESULTS - NOT-SORTED ARRAY VS. SORTED ARRAY

- quicksort - release mode - using at() function, no warnings

QUICKSORT - NOT-SORTED - RELEASE MODE - USING AT() FUNCTION, NO WARNINGS, MACBOOK PRO



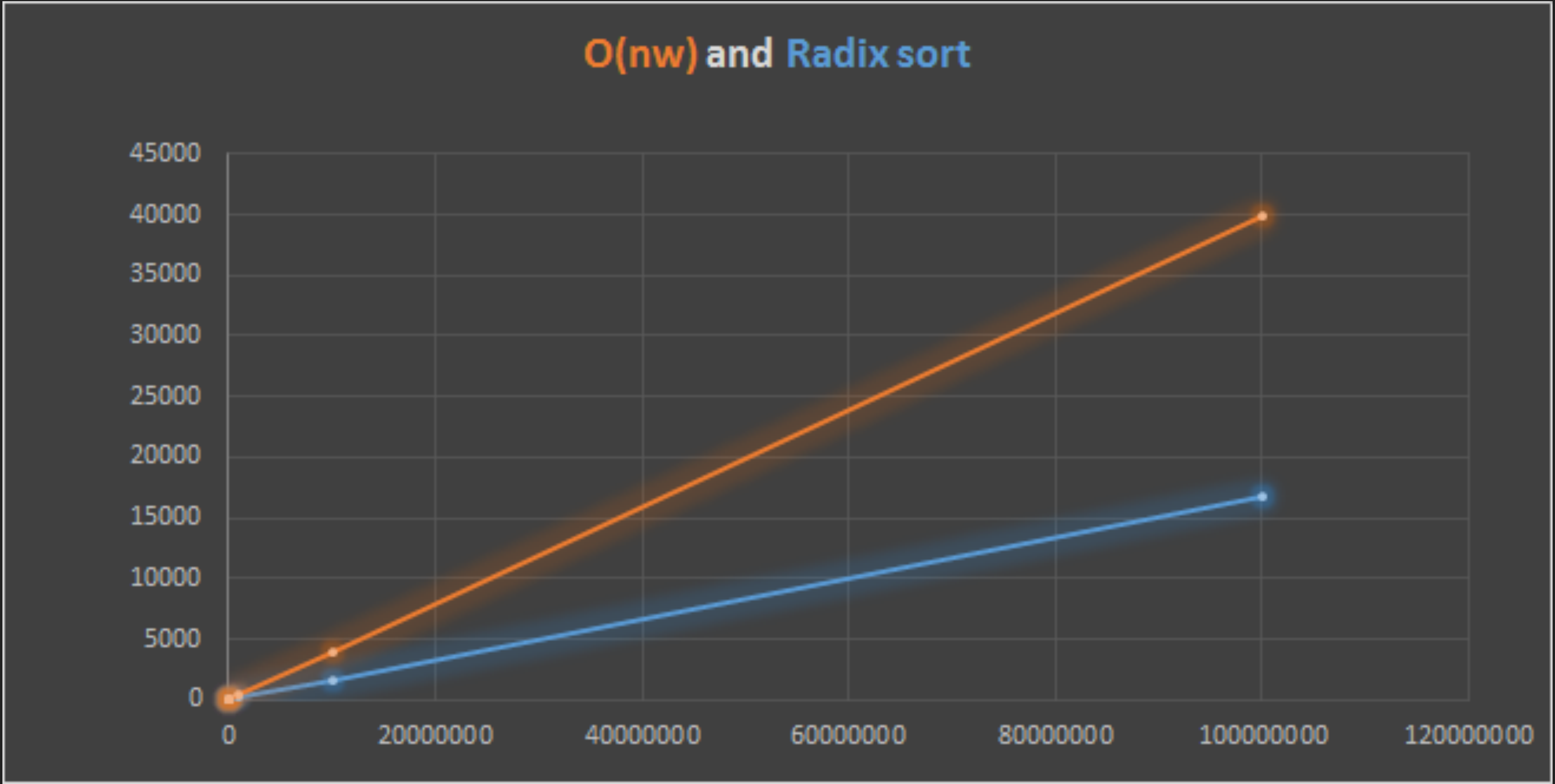
QUICKSORT – SORTED – RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, MACBOOK PRO



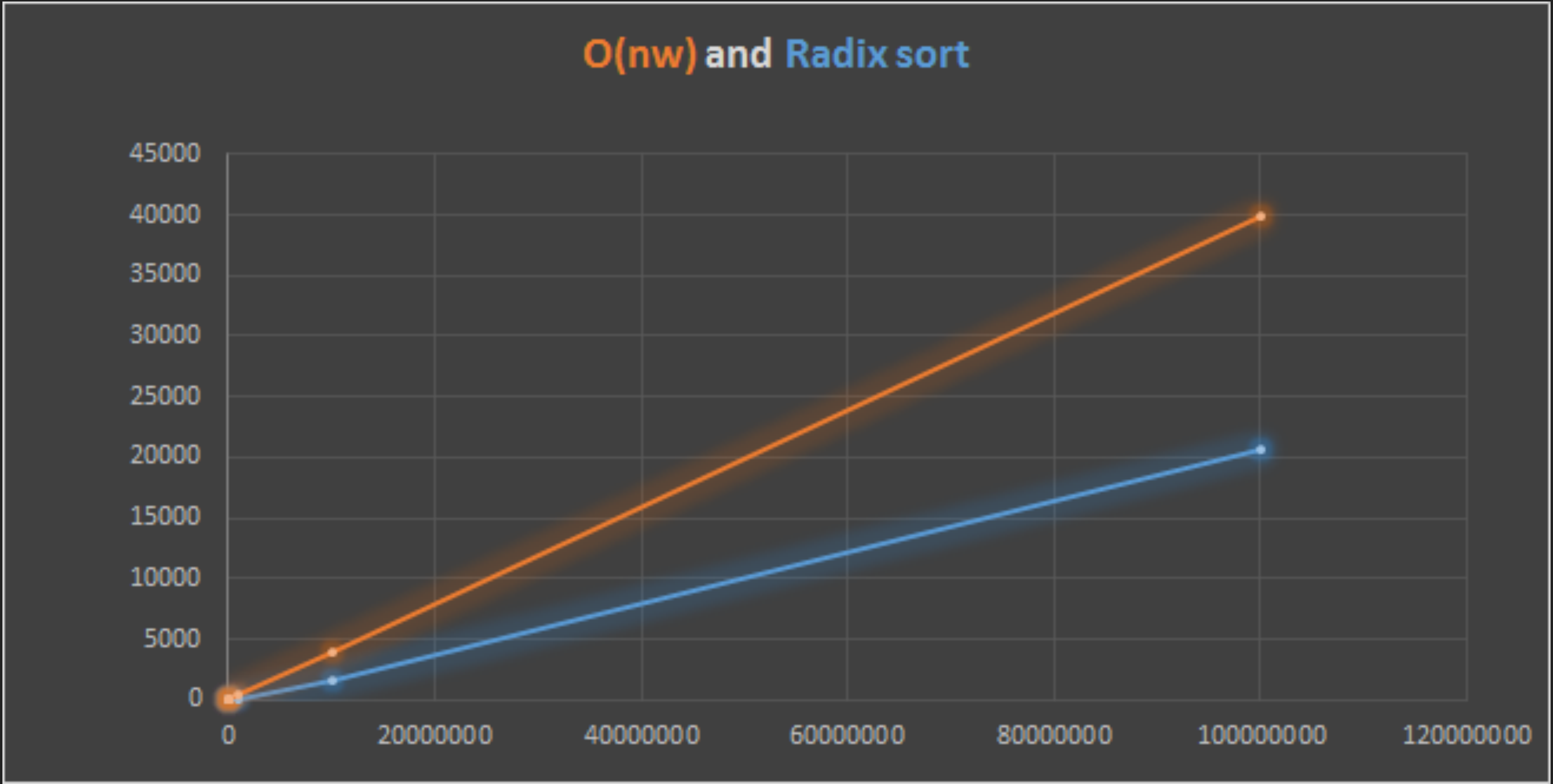
RESULTS - NOT-SORTED ARRAY VS. SORTED ARRAY

- radix sort - release mode - using at() function, no warnings

RADIX SORT- NOT-SORTED – RELEASE MODE – USING AT() FUNCTION, NO WARNINGS, MACBOOK PRO



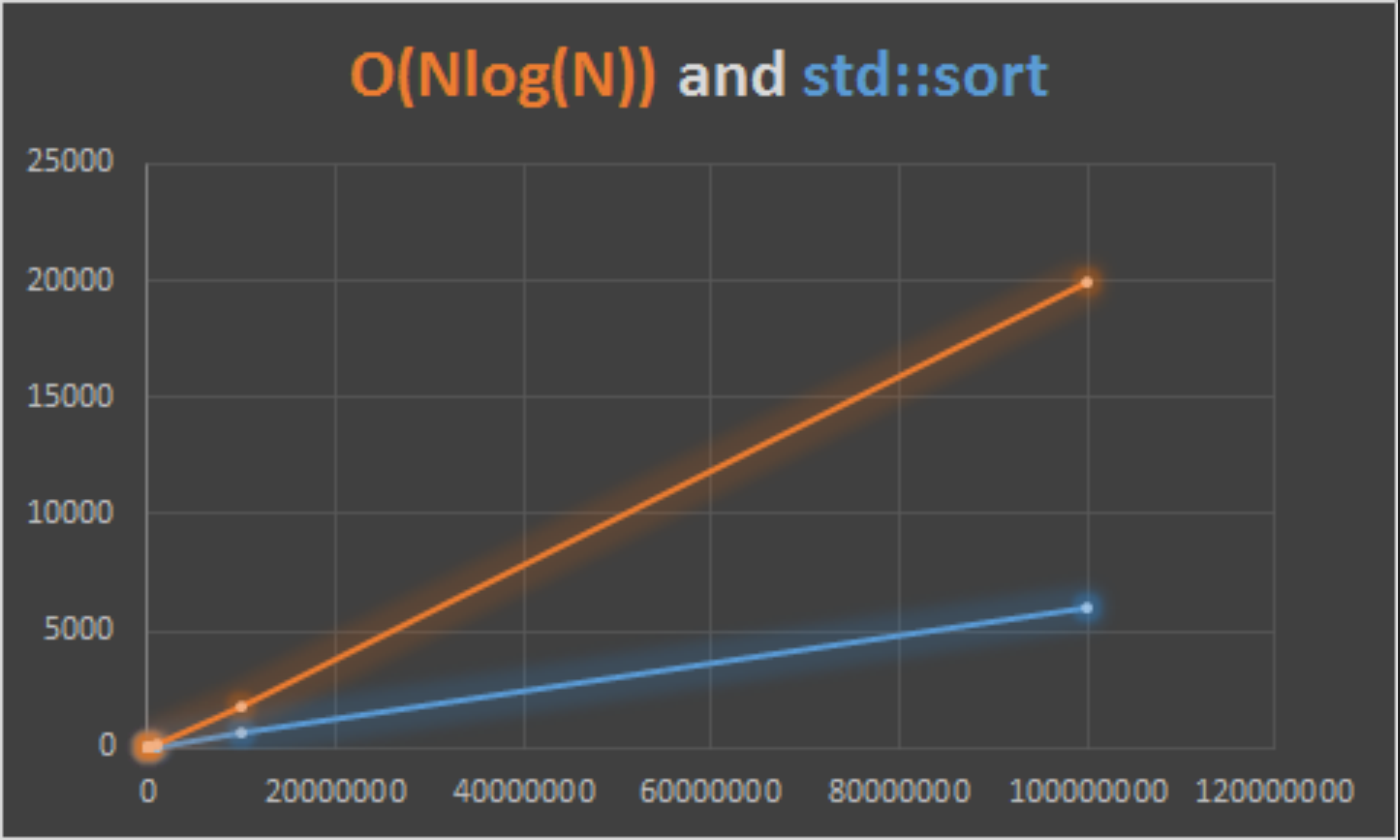
RADIX SORT- SORTED - RELEASE MODE - USING AT() FUNCTION, NO WARNINGS



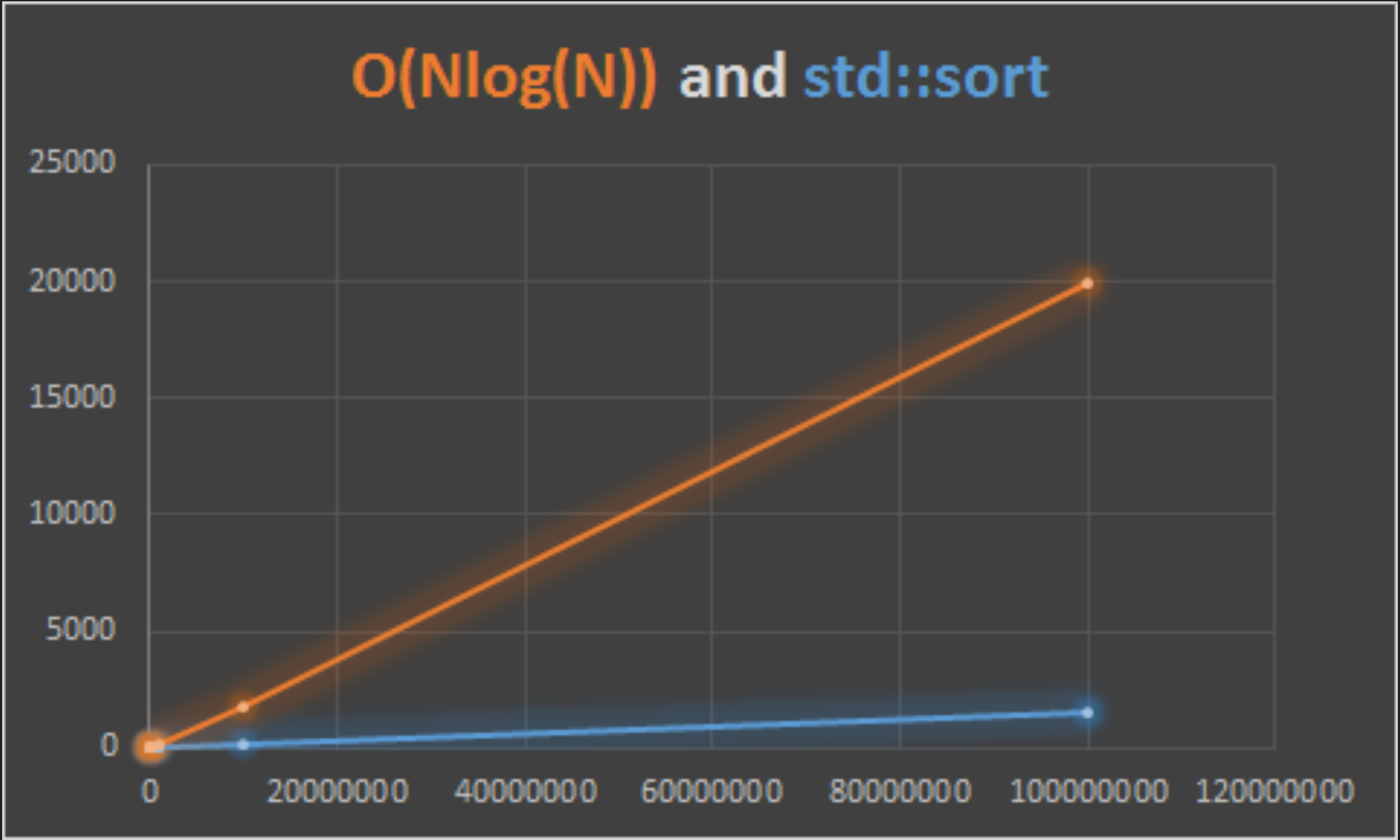
RESULTS - NOT-SORTED ARRAY VS. SORTED ARRAY

- std::sort - release mode - no warnings

STD::SORT - NOT-SORTED - RELEASE MODE - NO WARNINGS, MACBOOK PRO



STD::SORT - SORTED - RELEASE MODE - NO WARNINGS, MACBOOK PRO



THANK YOU!