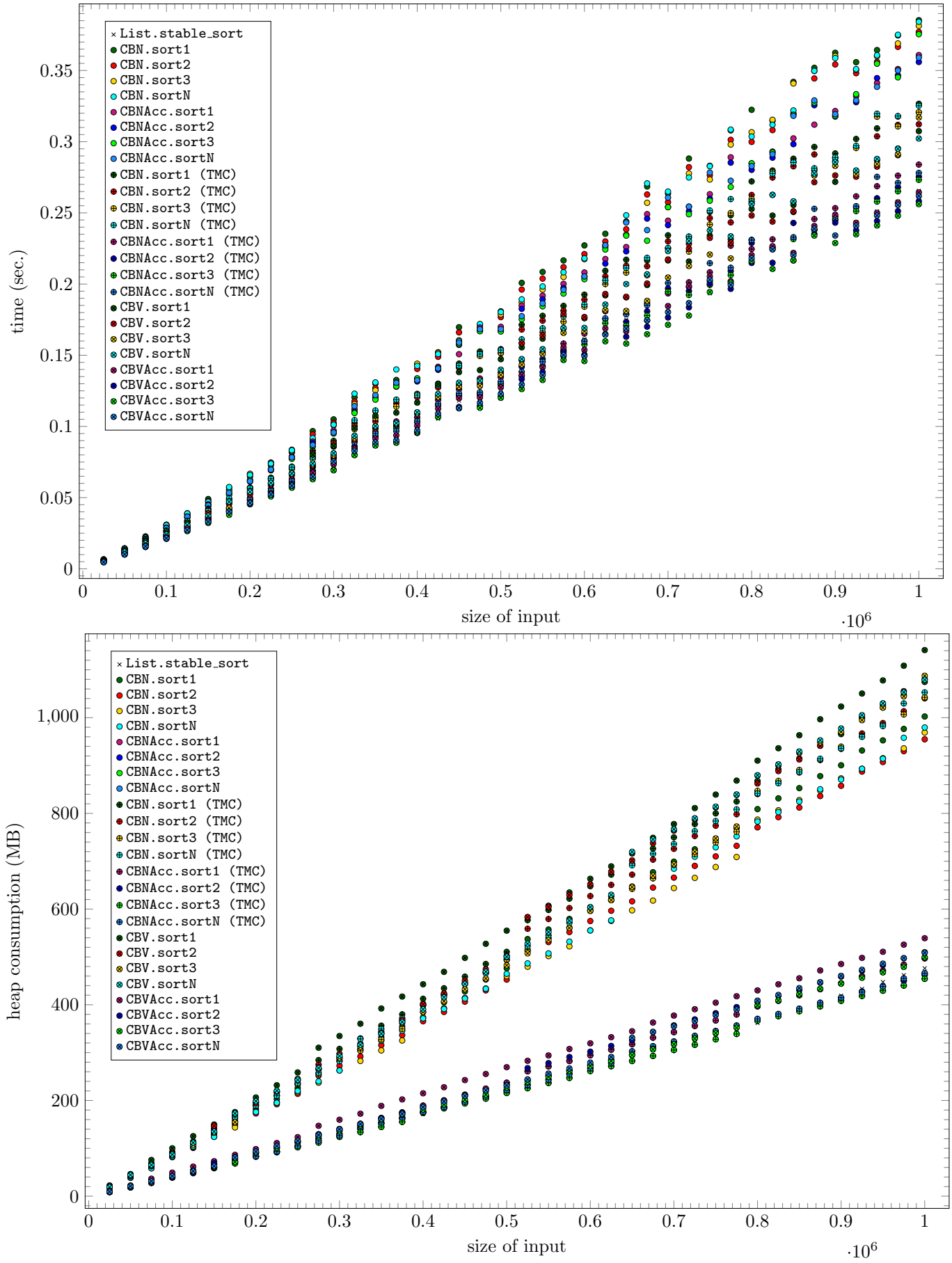

Figure 1: Benchmark result of *sort* `xs` in OCaml, where *sort* is `List.stable_sort compare` or an extracted sorting function applied to `(<=)`, and `xs` is a list of random natural numbers of type `int`.

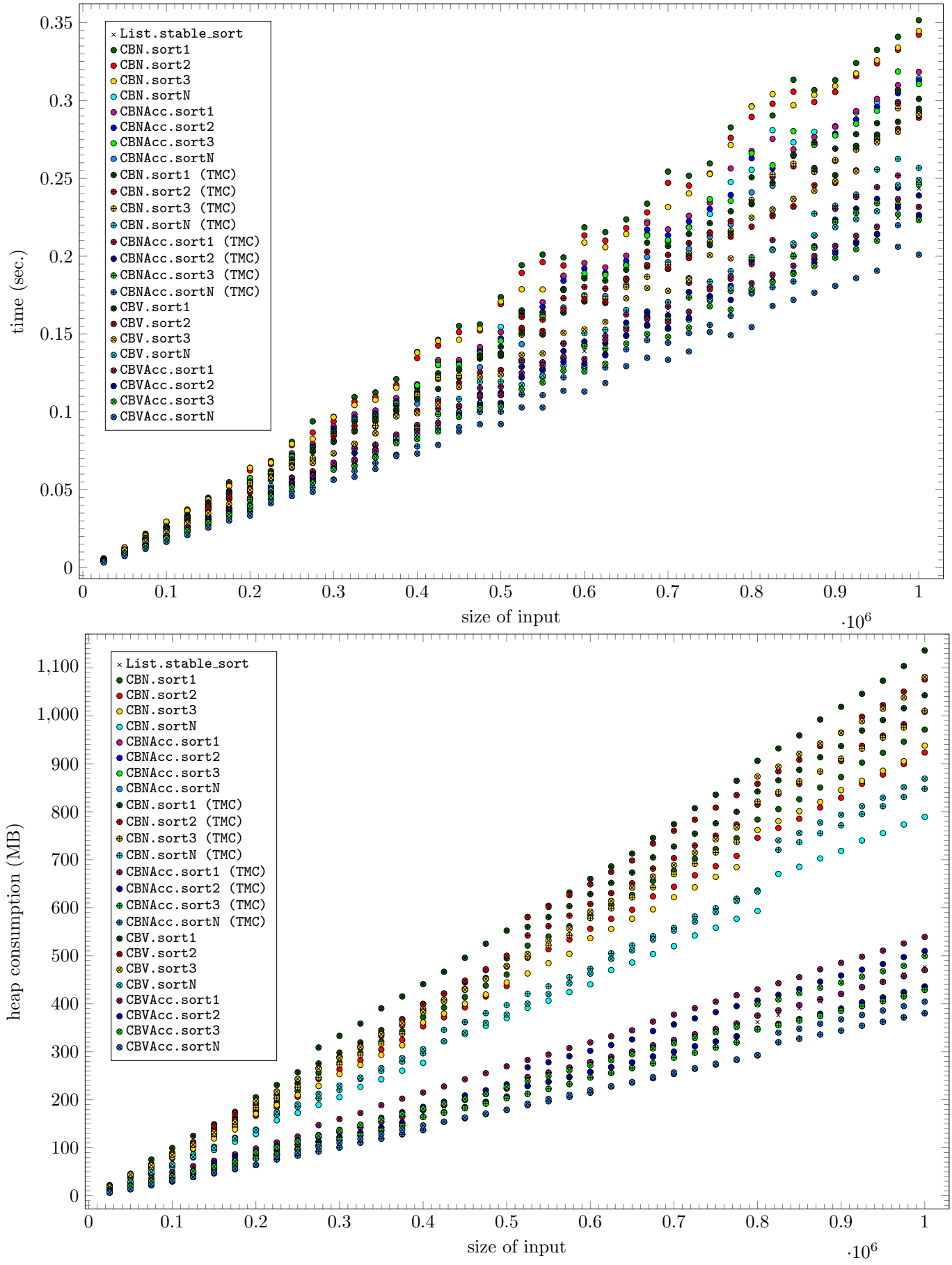

Figure 2: Benchmark result of *sort* `xs` in OCaml, where *sort* is `List.stable_sort compare` or an extracted sorting function applied to `(<=)`, and `xs` is a list of random natural numbers of type `int` but its every block of length 50 is sorted in ascending order.

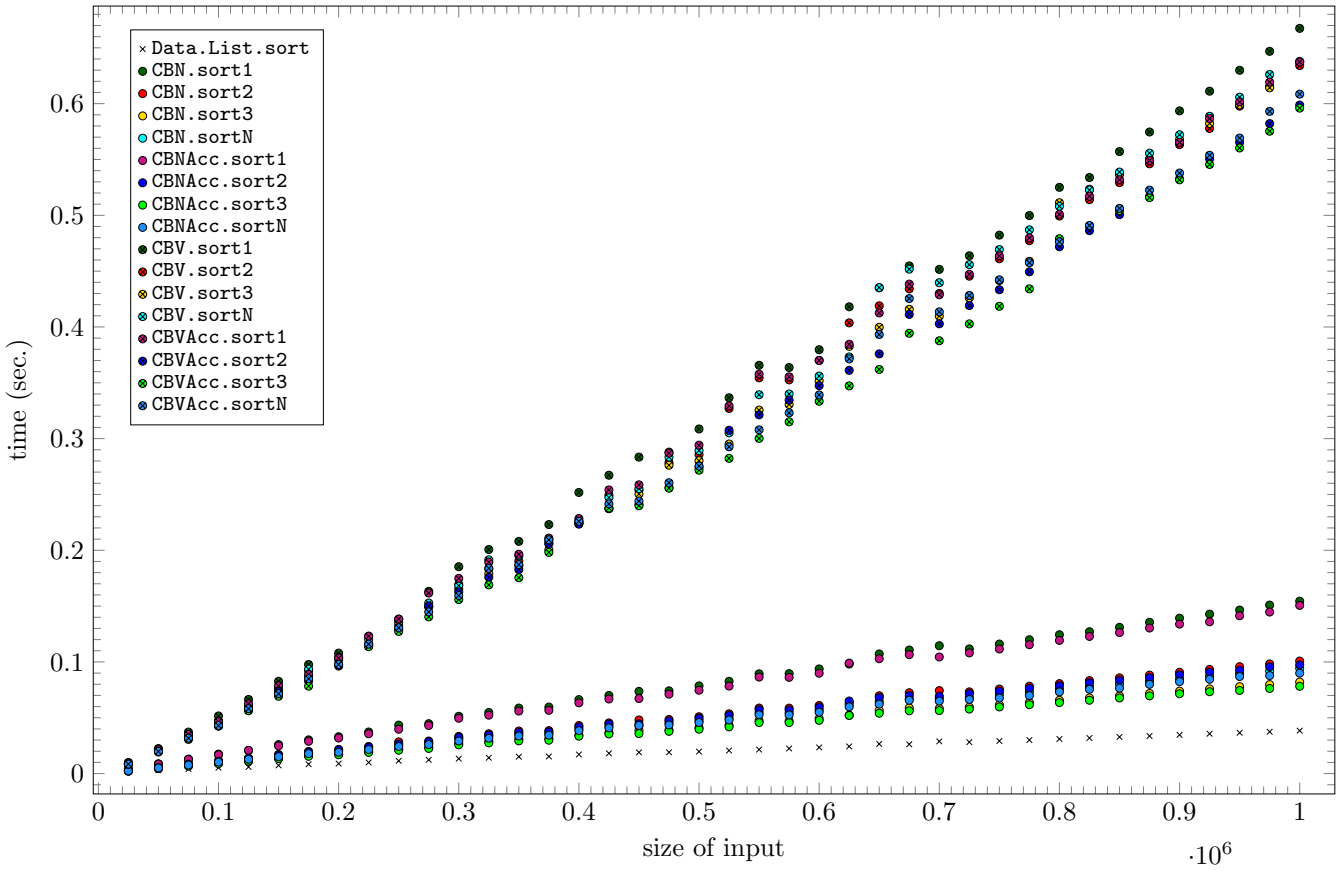

Figure 3: Benchmark result of `sorted (take 1000 (sort xs))` in Haskell, where *sort* is `Data.List.sort` or an extracted sorting function applied to `(<=)`, and `xs` is a list of random natural numbers of type `Int`.

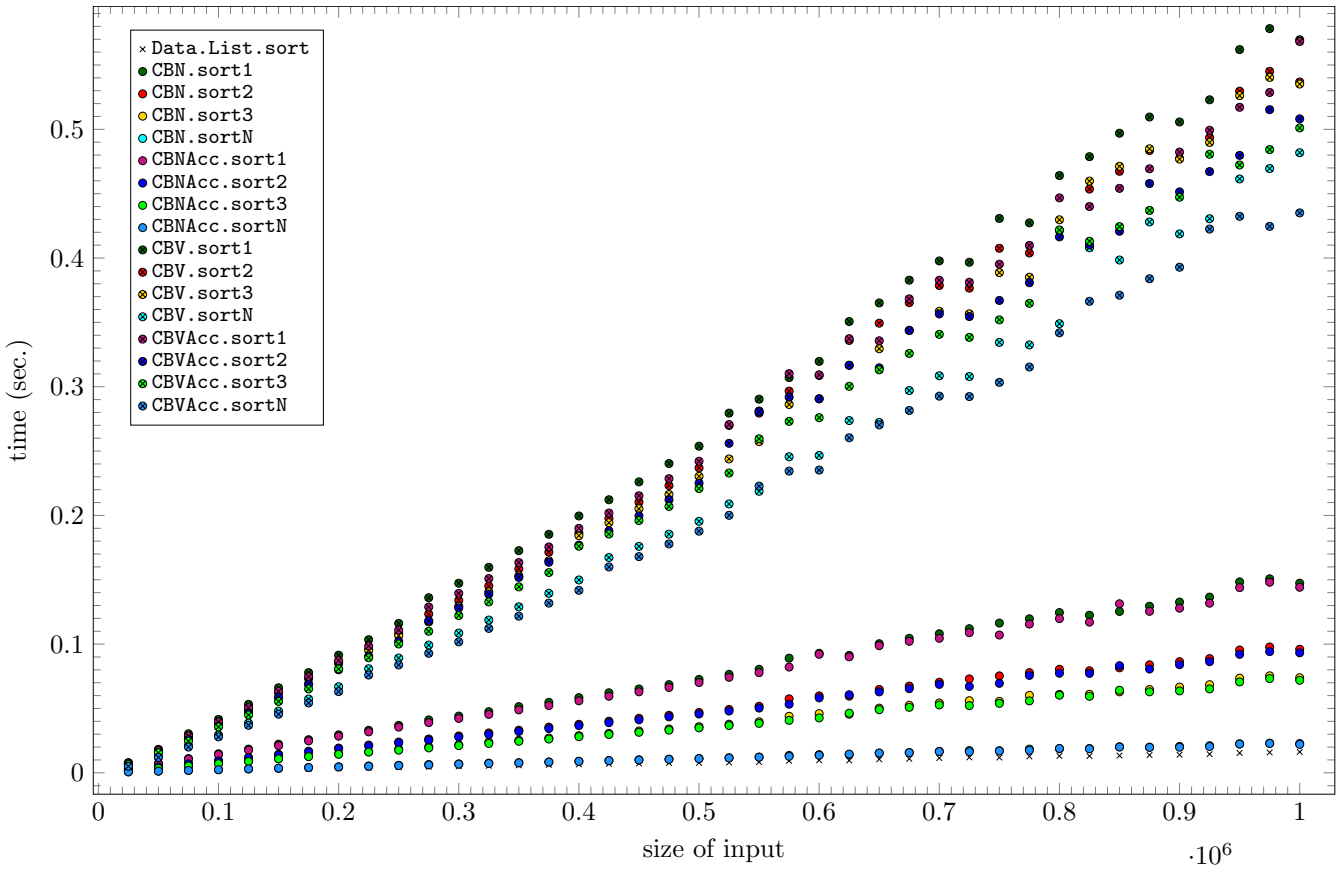

Figure 4: Benchmark result of `sorted (take 1000 (sort xs))` in Haskell, where *sort* is `Data.List.sort` or an extracted sorting function applied to `(<=)`, and `xs` is a list of random natural numbers of type `Int` but its every block of length 50 is sorted in ascending order.

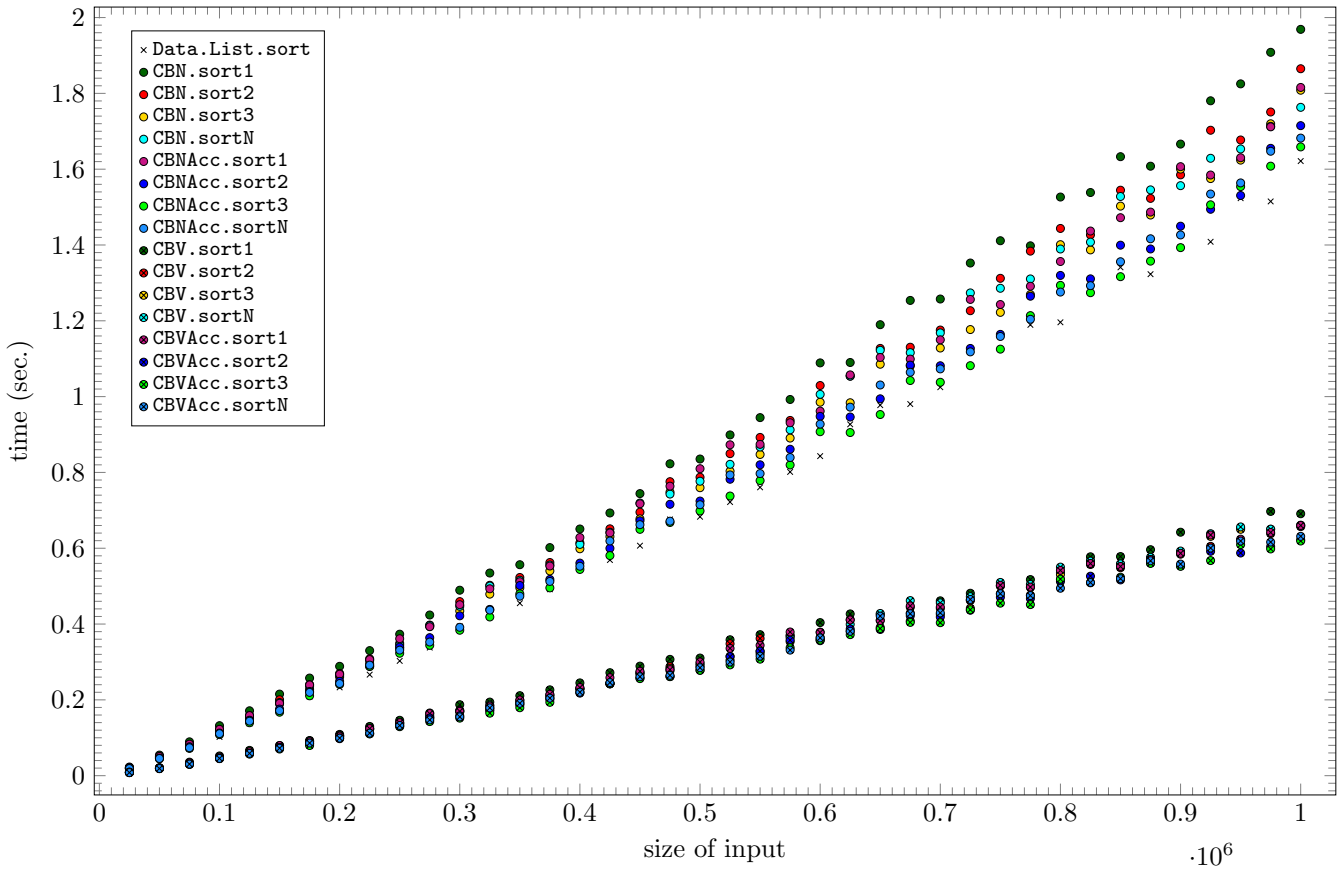

Figure 5: Benchmark result of `sorted (sort xs)` in Haskell, where *sort* is `Data.List.sort` or an extracted sorting function applied to `(<=)`, and `xs` is a list of random natural numbers of type `Int`.

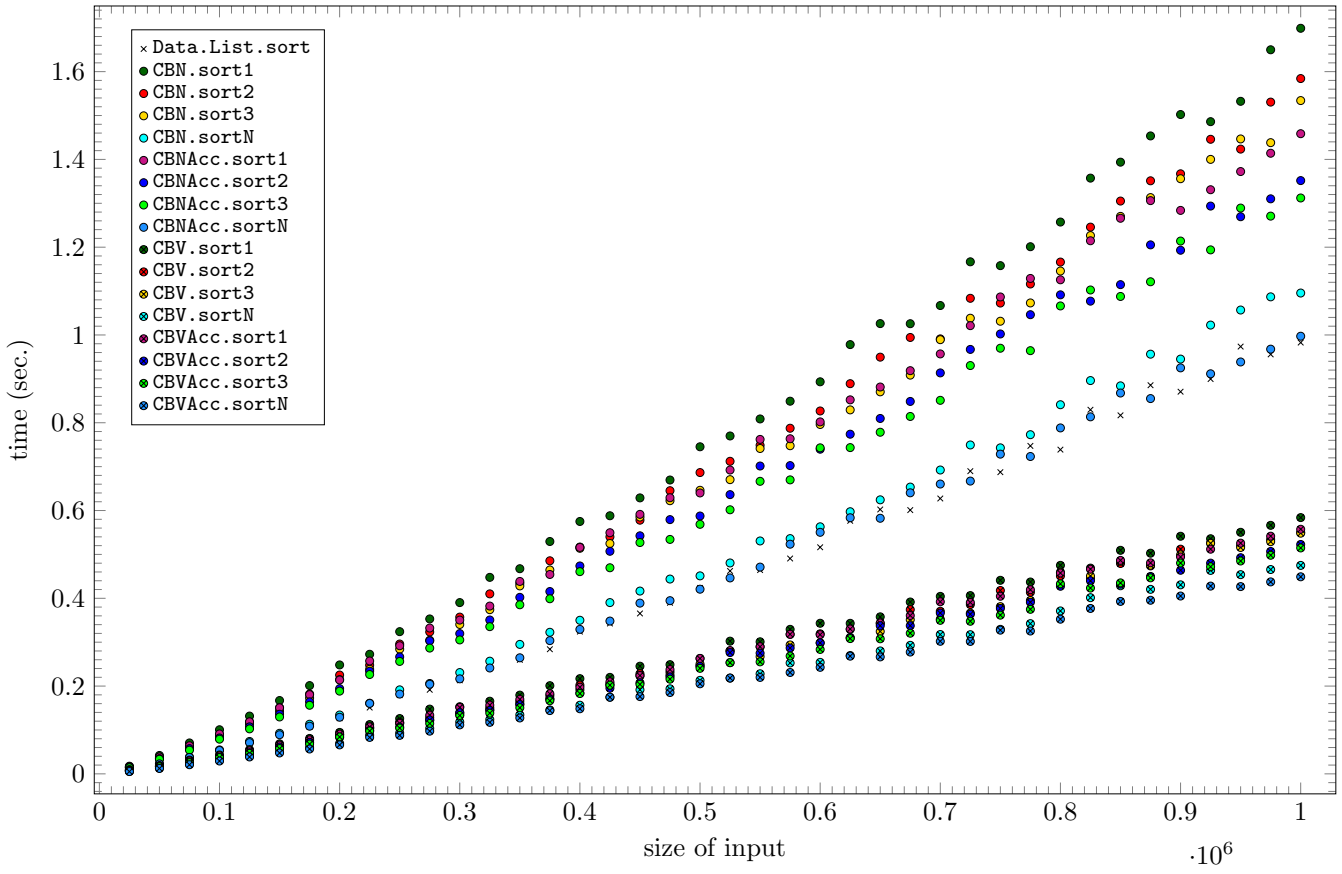

Figure 6: Benchmark result of `sorted (sort xs)` in Haskell, where *sort* is `Data.List.sort` or an extracted sorting function applied to `(<=)`, and `xs` is a list of random natural numbers of type `Int` but its every block of length 50 is sorted in ascending order.

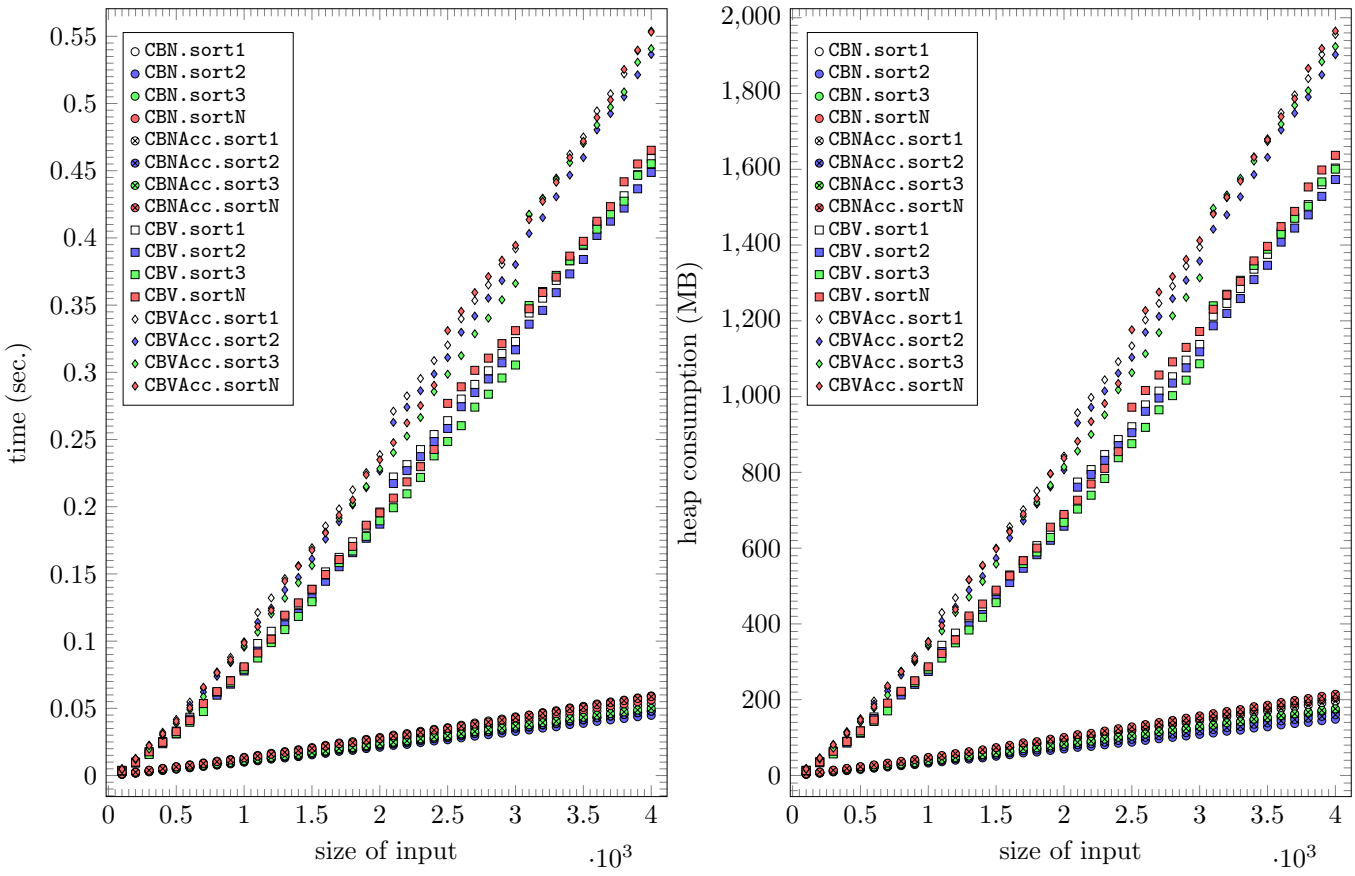

Figure 7: Benchmark result of `sorted N.leb (take 10 (sort N.leb xs))` with `lazy`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N`.

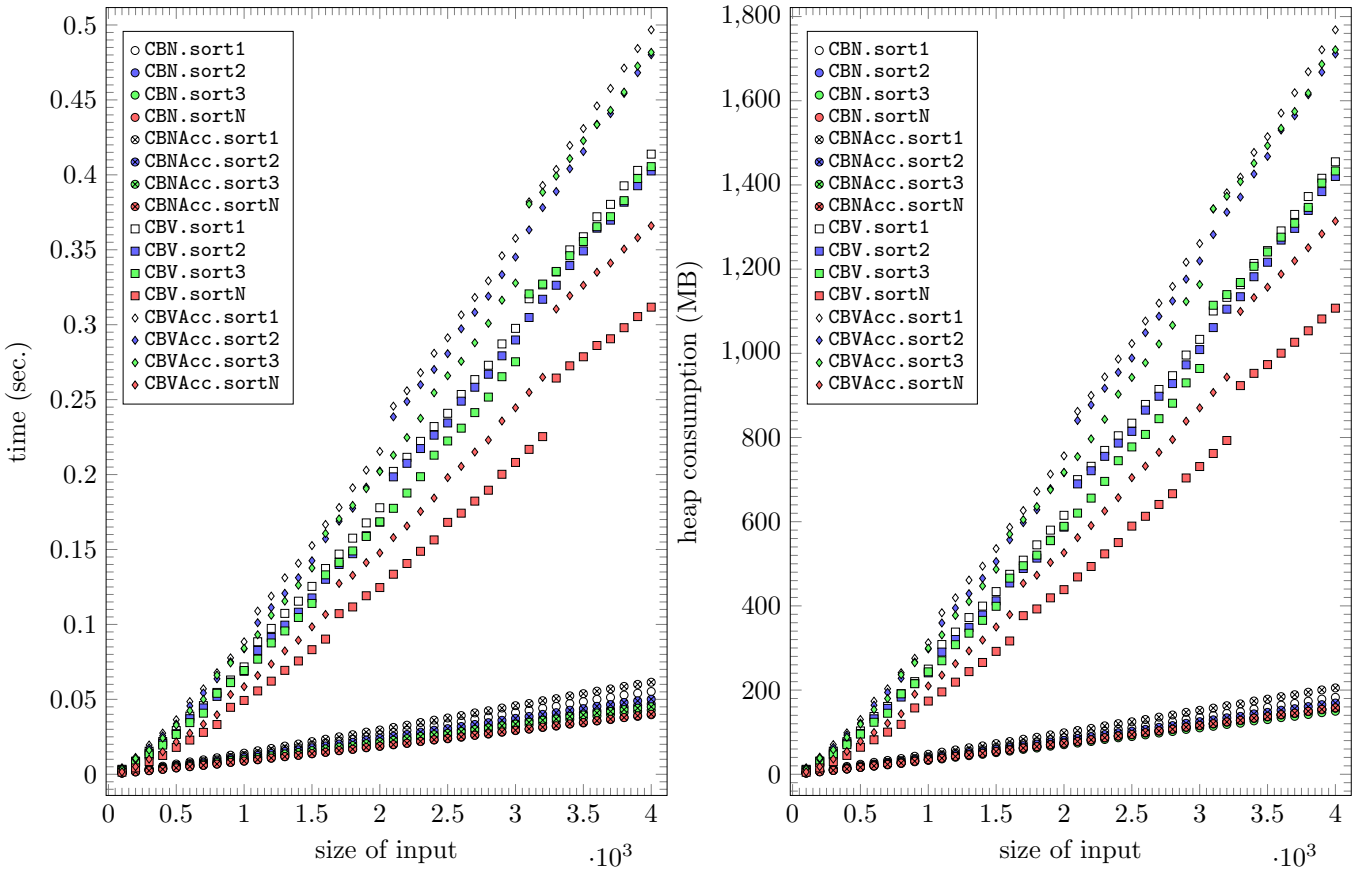

Figure 8: Benchmark result of `sorted N.leb (take 10 (sort N.leb xs))` with `lazy`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N` but its every block of length 50 is sorted in ascending order.

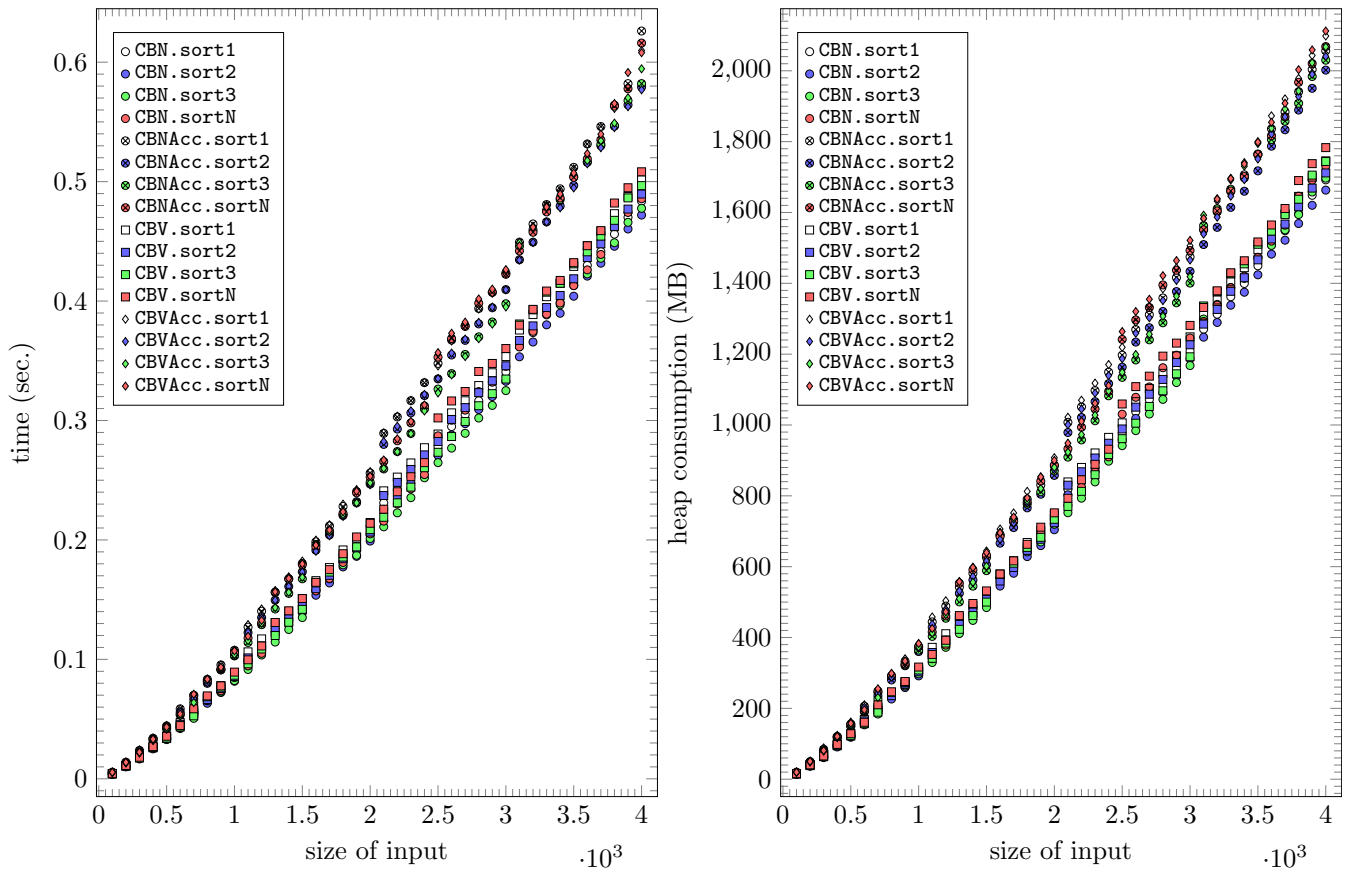

Figure 9: Benchmark result of `sorted N.leb (sort N.leb xs)` with `lazy`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N`.

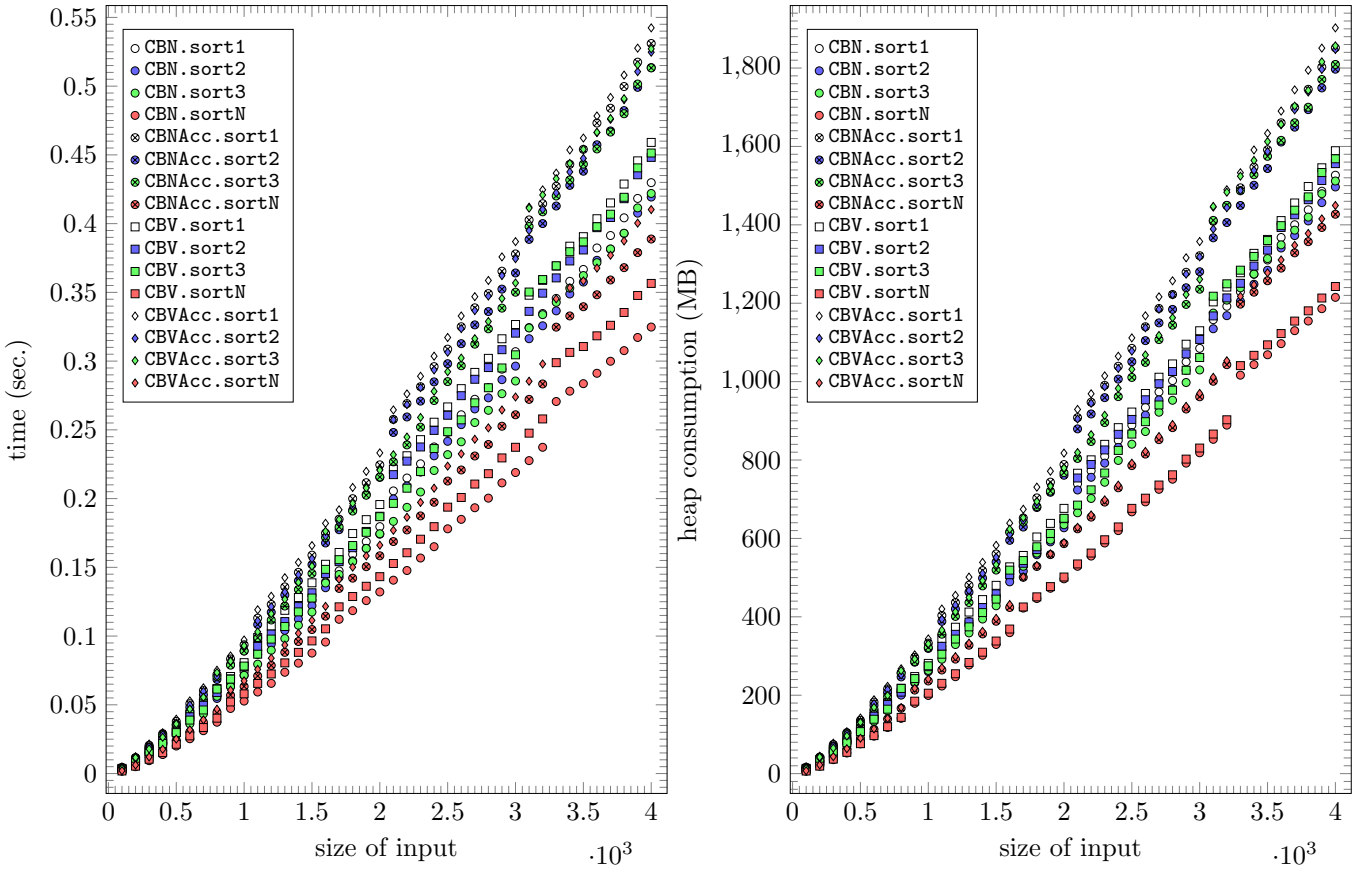

Figure 10: Benchmark result of `sorted N.leb (sort N.leb xs)` with `lazy`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N` but its every block of length 50 is sorted in ascending order.

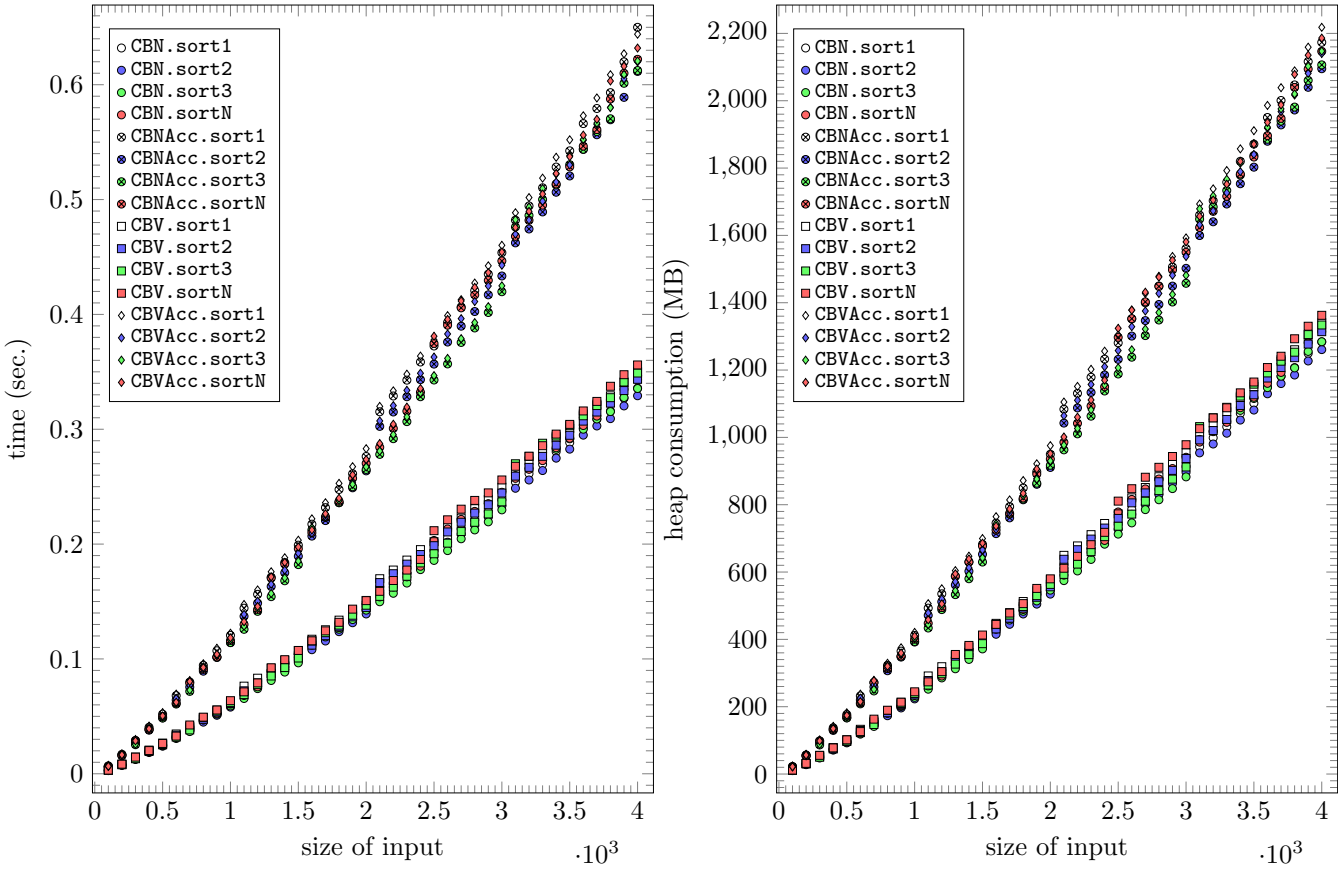

Figure 11: Benchmark result of `sorted N.leb (sort N.leb xs)` with `compute`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N`.

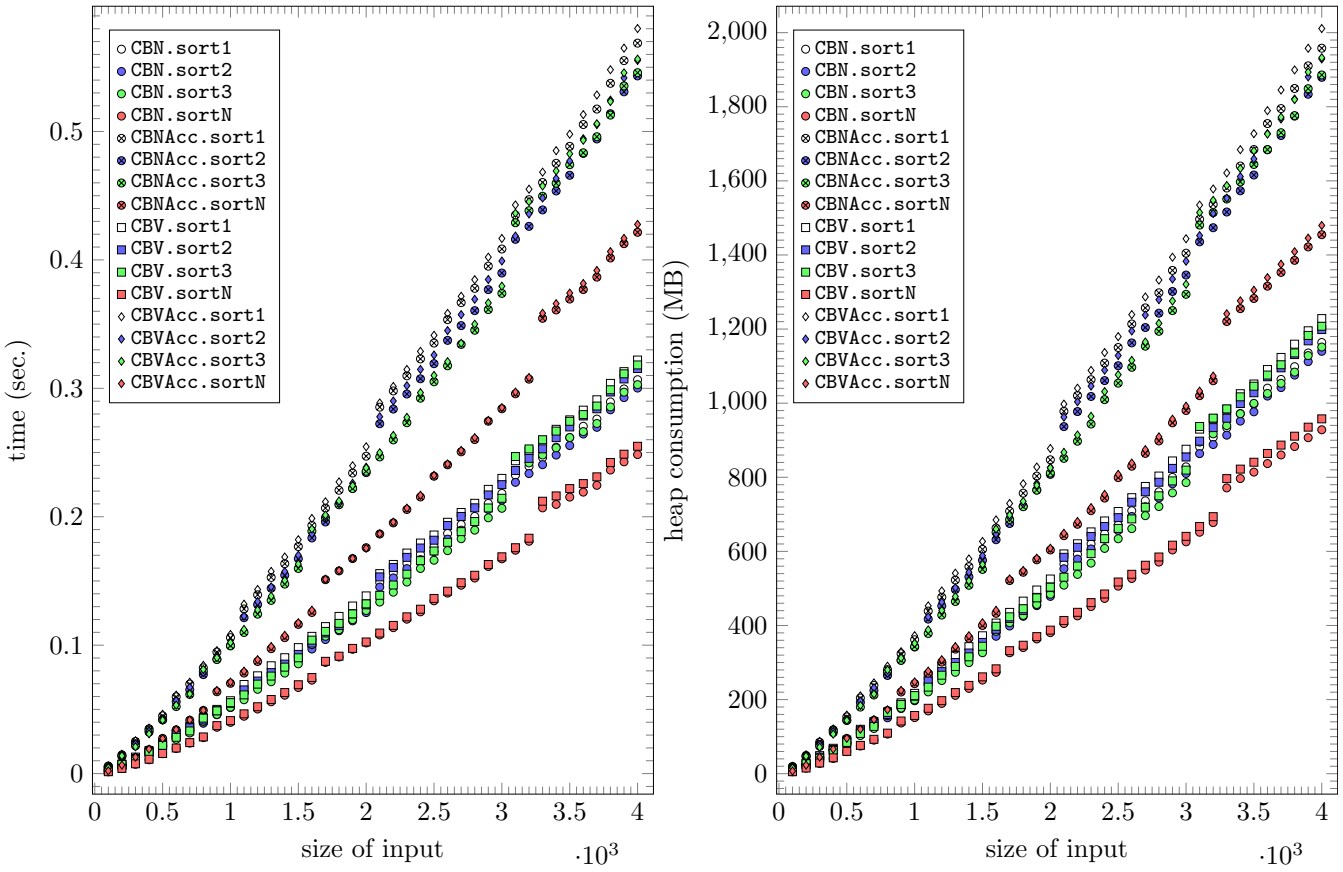

Figure 12: Benchmark result of `sorted N.leb (sort N.leb xs)` with `compute`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N` but its every block of length 50 is sorted in ascending order.

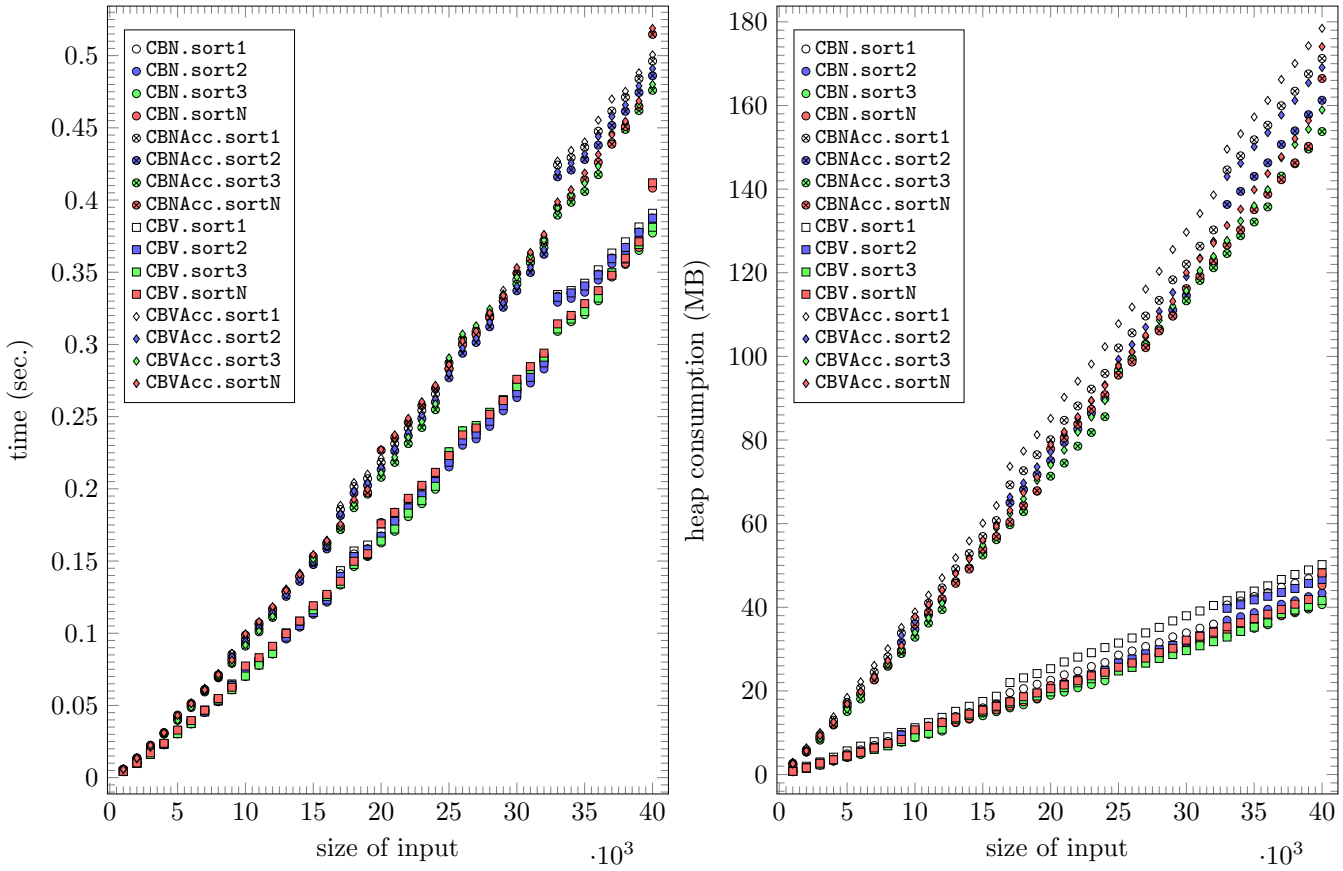

Figure 13: Benchmark result of `sorted N.leb (sort N.leb xs)` with `vm_compute`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N`.

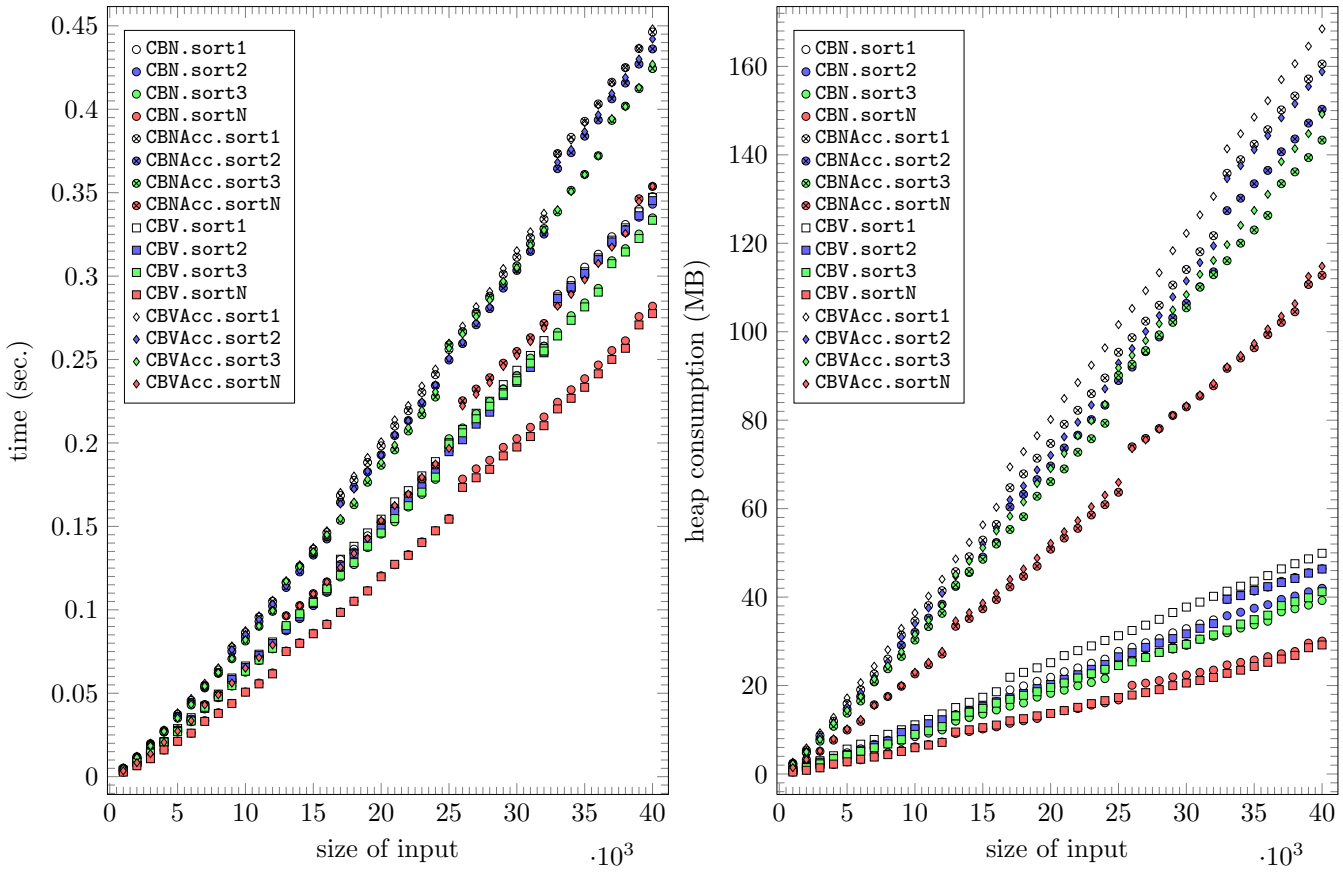

Figure 14: Benchmark result of `sorted N.leb (sort N.leb xs)` with `vm_compute`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N` but its every block of length 50 is sorted in ascending order.

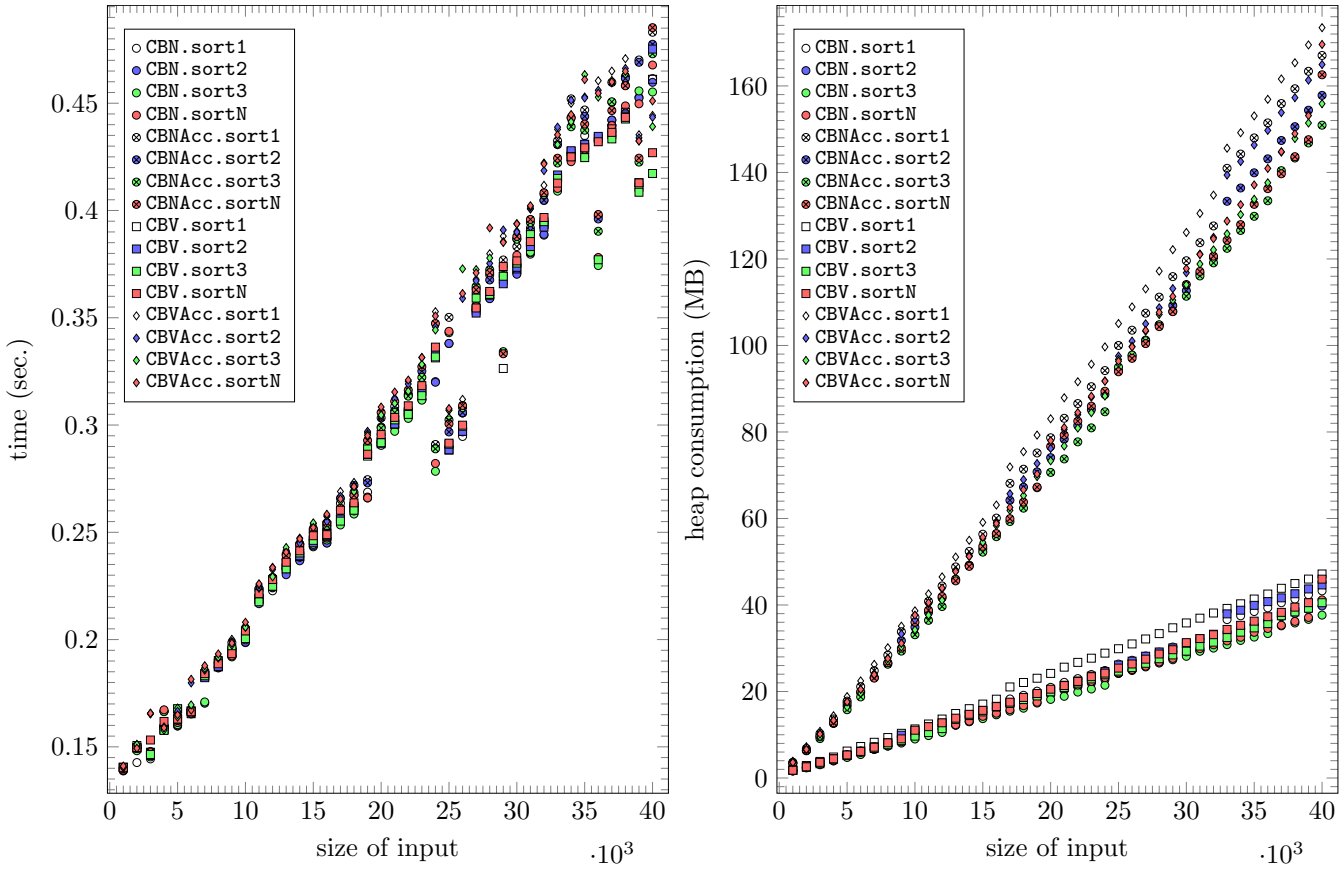

Figure 15: Benchmark result of `sorted N.leb (sort N.leb xs)` with `native_compute`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N`.

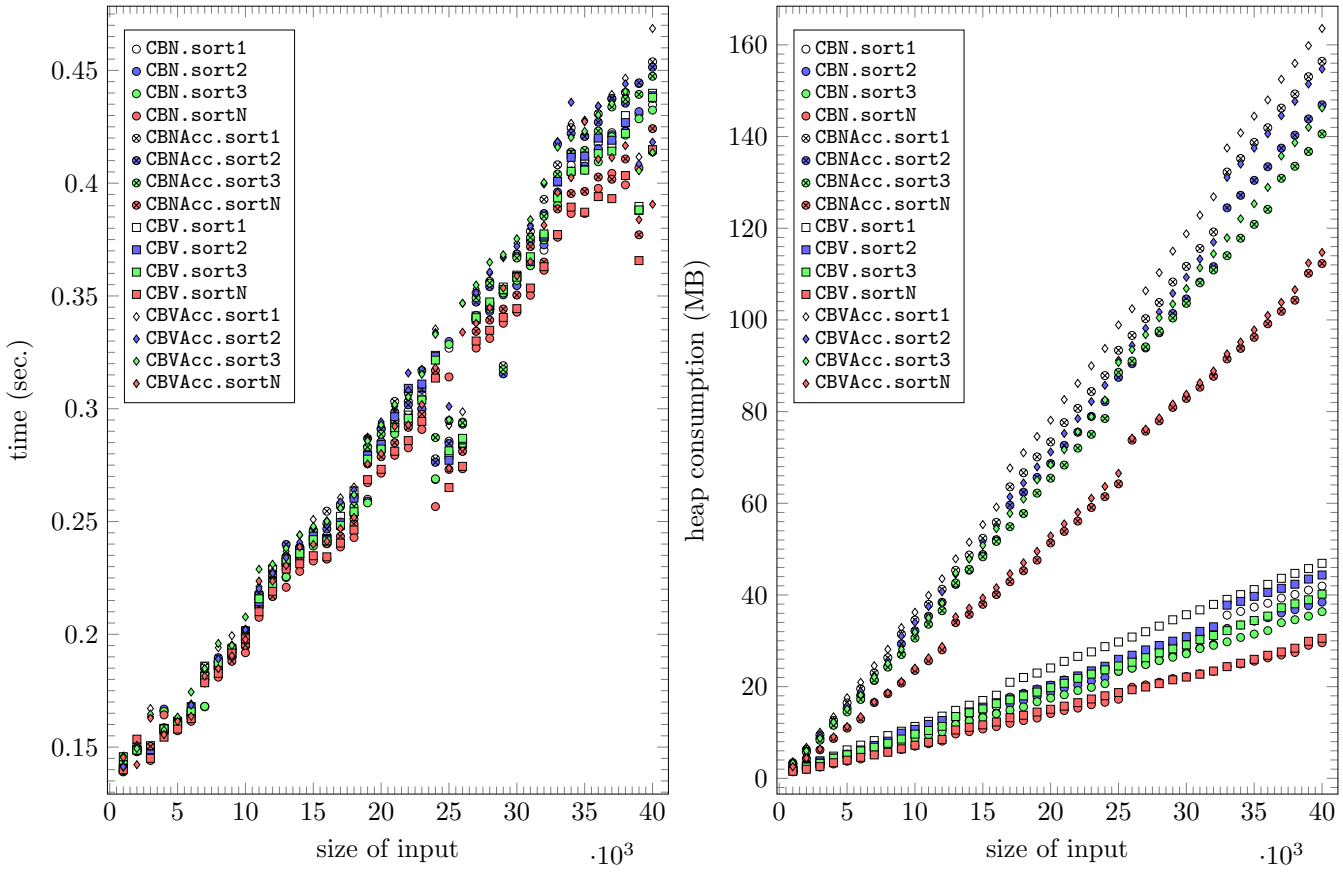

Figure 16: Benchmark result of `sorted N.leb (sort N.leb xs)` with `native_compute`, where *`sort`* is a sorting function, and `xs` is a list of random natural numbers of type `N` but its every block of length 50 is sorted in ascending order.