A1.

Bucket Sort:

Bucket Sort works by dividing the given array to several lists (called bucket), and then sorting it (generally by insertion sort or by inserting in the sorted order while inserting in list).

Best Case: When elements of array are distributed uniformly in all the buckets. The complexity in this case is O(n).

Worst Case: When all elements of array move into one bucket. It then becomes the worst case of insertion sort. Time complexity in this case is O(n2).

Average Case: The average case complexity is O(n).

Heap Sort:

The number of comparisons made actually can depend on the order in which the values are given. The fact that the best and worst case are each O(n log n) only means that asymptotically there's no difference between the two, though they can differ by a constant factor.

Best case: O(n log(n))

Worst case: O(n log(n))

Average case: O(n log(n))

Insertion Sort:

Best case: When elements inserted in sorted order. Number of comparisons in this case is equal to 1 less than the number of elements inserted.
Time complexity: O(n)

Worst case: When the list is reverse sorted. In this case, each element of the list is shifted with each insertion.
Time complexity: O(n^2)

Average case: If we take input in random order then on average, we expect that each element is less than half the elements to its left. In this case, on average,our algorithm will slide k/2 of numbers. The running time would be half of the worst case running time. But in asymptotic notation, would be O(n^2).

Merge Sort:

This is also not input order sensitive , as in any order we break the input list till the smallest unit. Complexity for Combining O(n), for all dividing steps O(log(n)).

Best case: Occurs when list is already sorted. In this case, number of shifts is minimum.
Time complexity: O(n log(n))

Worst case: Time complexity: O(n log(n))

Average case: Time complexity: O(n log(n))

Quick Sort:

Best Case: Even partition can be generated by taking median of numbers. Although the complexity remains the same
Time complexity : O(n log(n))

Worst Case: Bad Partition - when one subproblem always have 0 elements and the other has n-1.
Time complexity: O(n^2)

Average Case: Time complexity : O(n log(n))
The worst case does not often happen here.

Radix Sort:

Complexity for counting sort comes out to be O(n+k), here as the number of passes increases , complexity is multiplied with number of passes.

Complexity O(n * no. of passes)

Best Case: Time complexity: O(n*k)

Worst case: Time complexity: O(n*k)
Average case: Time complexity: O(n*k)

Selection Sort:
The selection sort searches for the smallest/greatest element in the array and swaps it with the first element. Similarly, it does for the second, third elements and so on. It is not much sensitive as for every input (in any order) , to find minimum from the unsorted list, we require O(n) comparisons.
Best case: O(n^2)
Worst case: O(n^2)
Average case: O(n^2)

A 2.
Machine Specifications-
OS System: Ubuntu 15.04, 64 bit
Processor: Intel® Core™ i5-4210U CPU @ 1.70GHz × 4

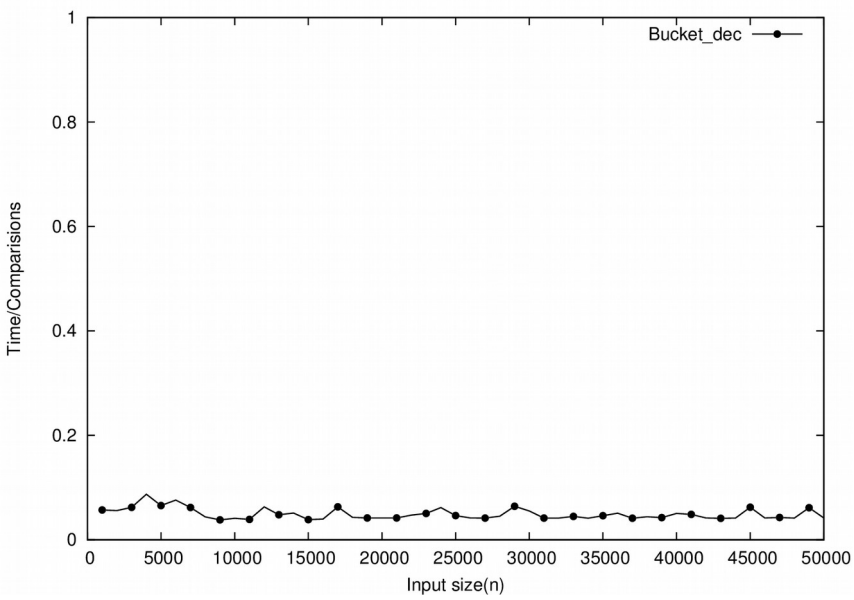
Timing Mechanism-
The timing is calculated using The C library function clock_t clock(void) which returns the number of clock ticks elapsed since the program was launched. To get the number of seconds used by the CPU, we divided by CLOCKS_PER_SEC.

Number of times each experiment was repeated- 3

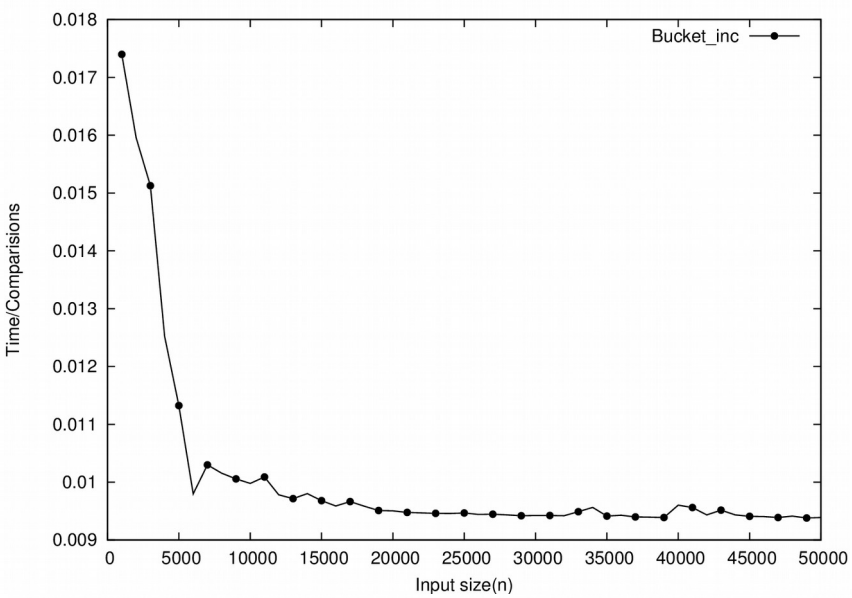
A 6)

Bucket Sort:

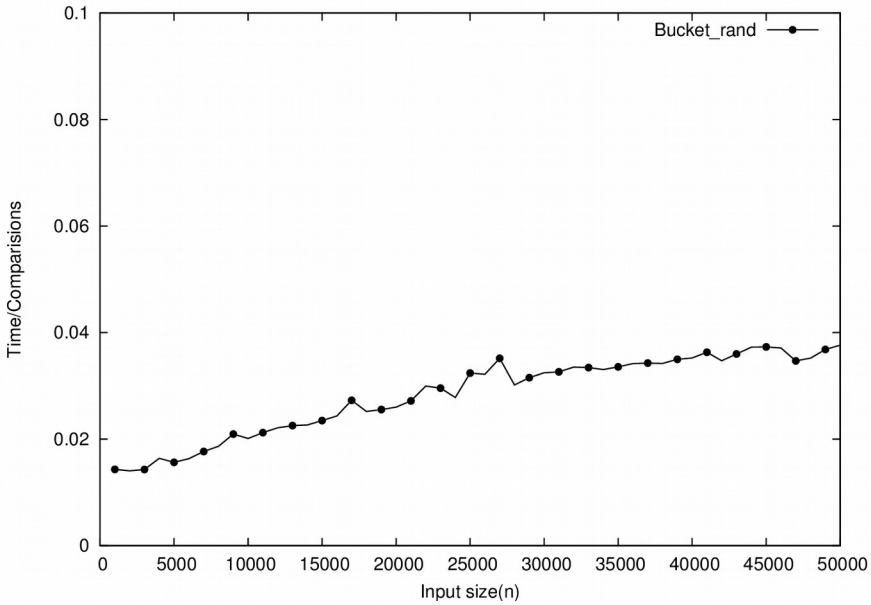
Decreasing values



Increasing values

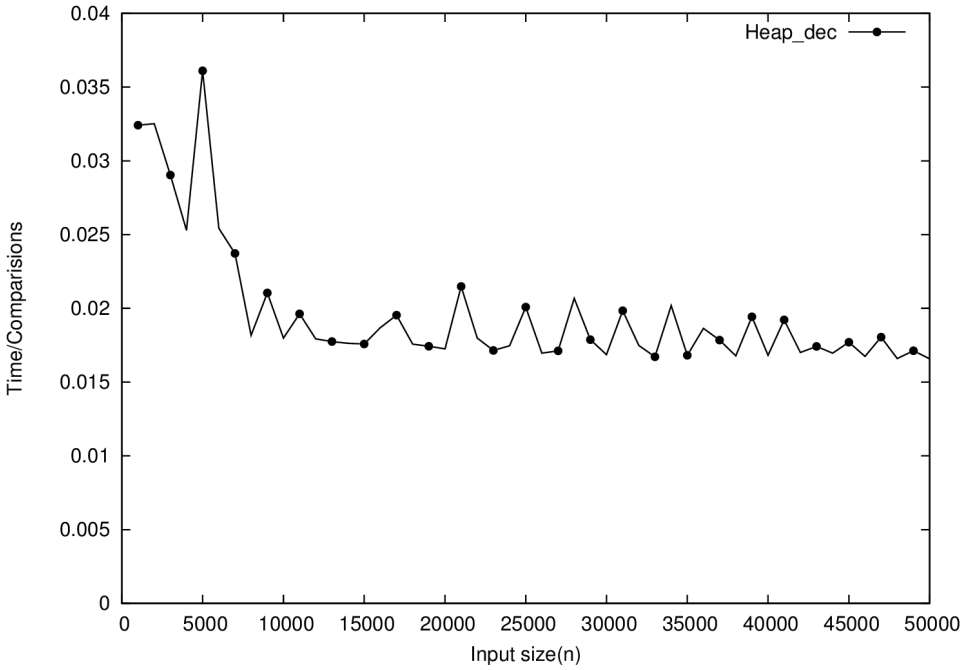


Random values

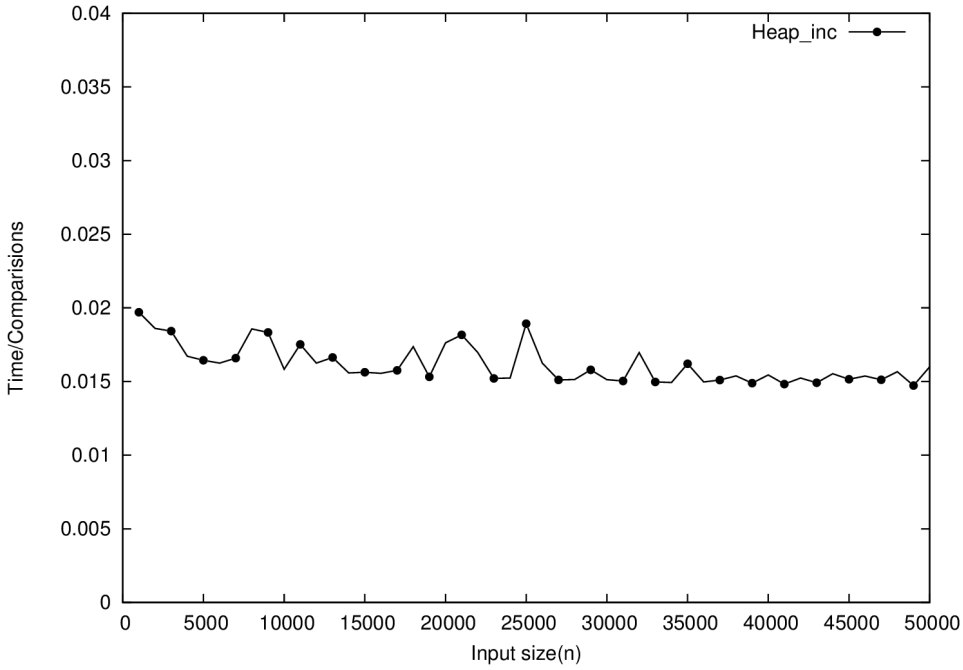


Heap Sort

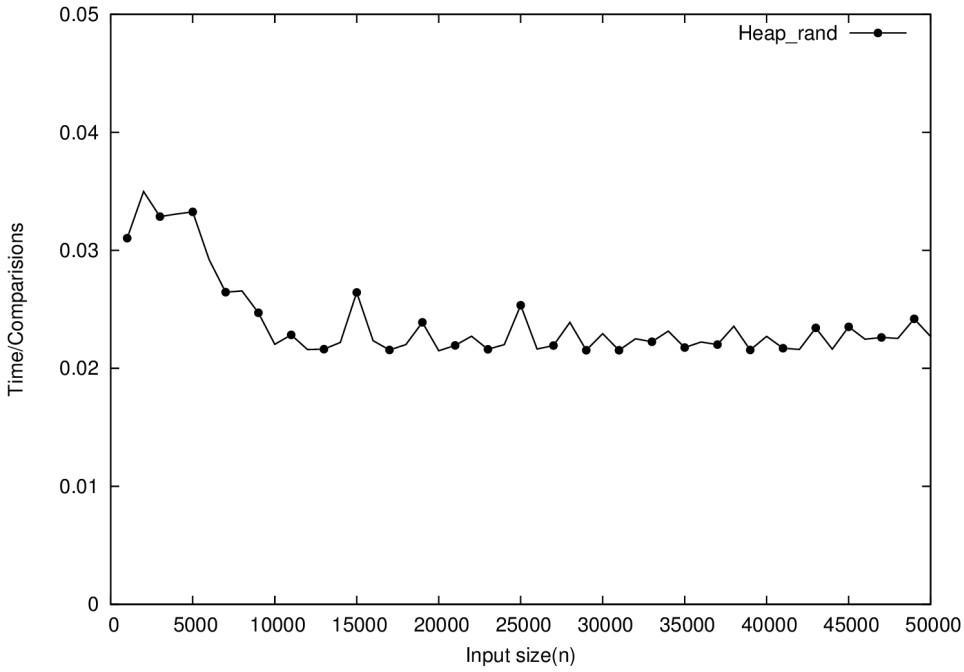
Decreasing values



Increasing values

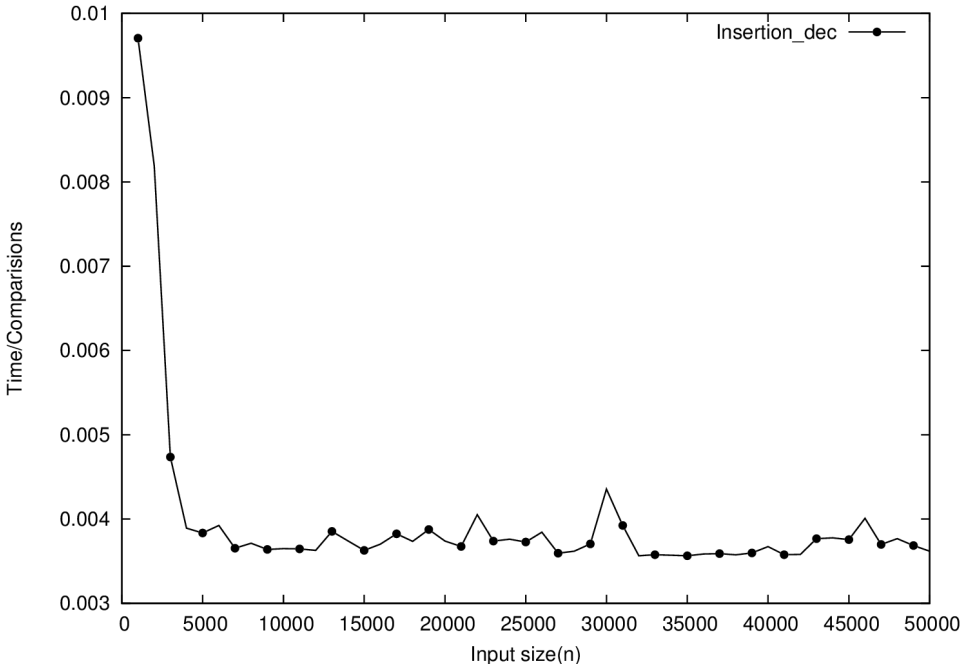


Random values

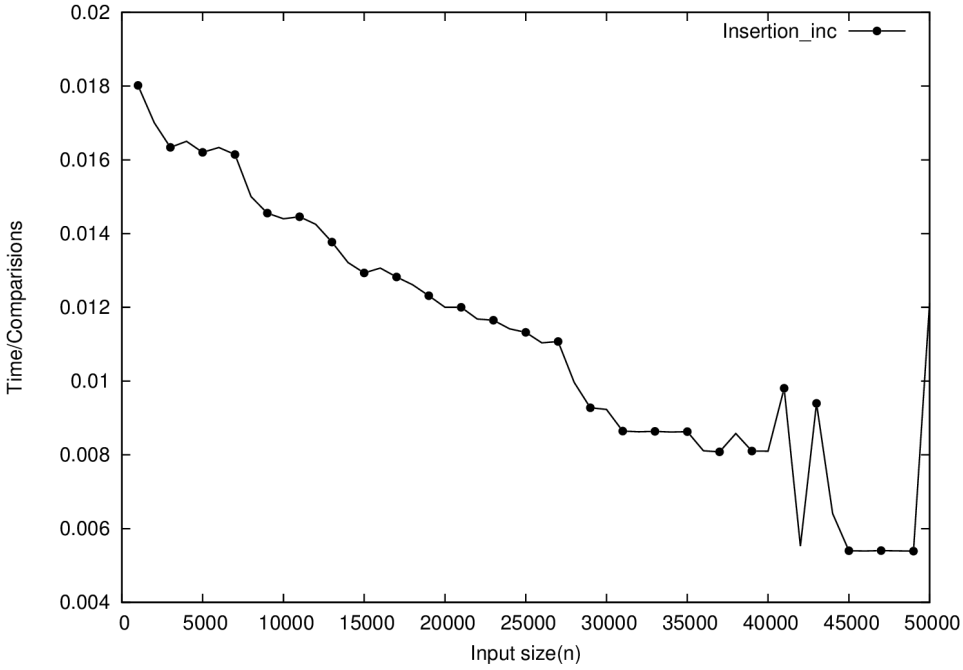


Insertion Sort

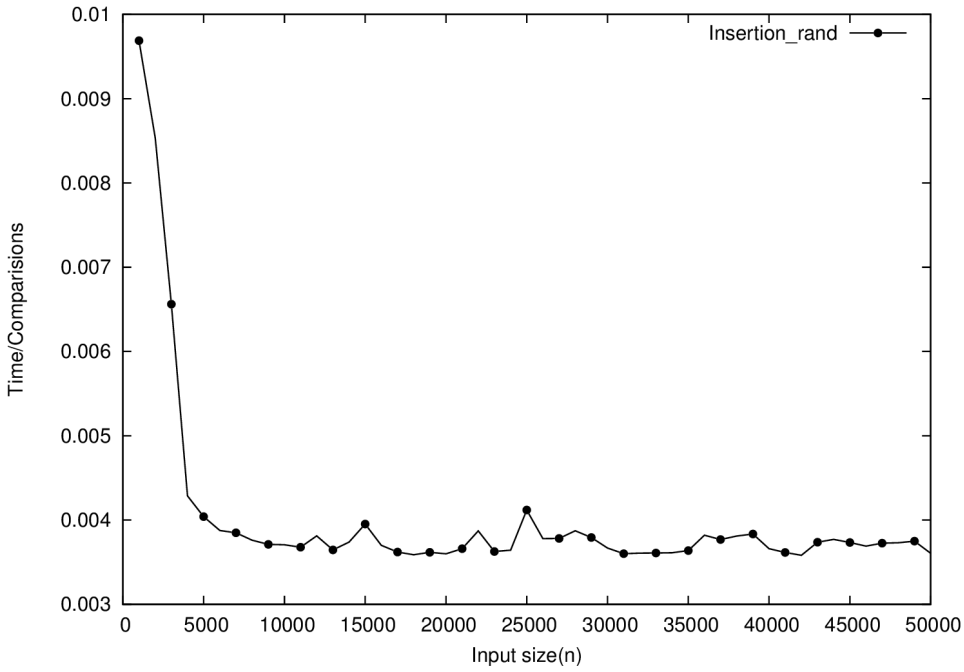
Decreasing values



Increasing values

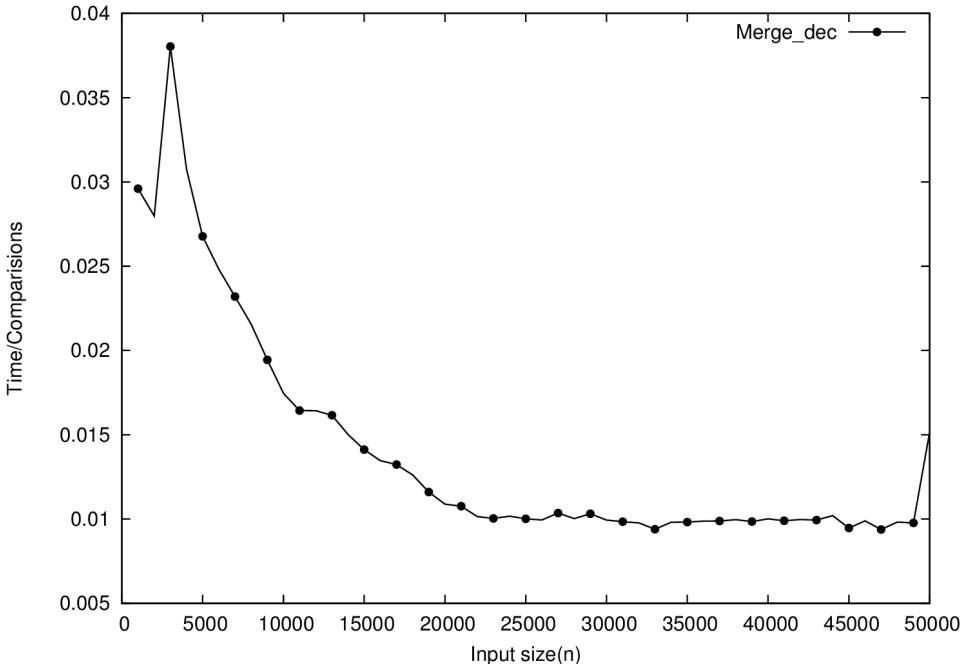


Random values

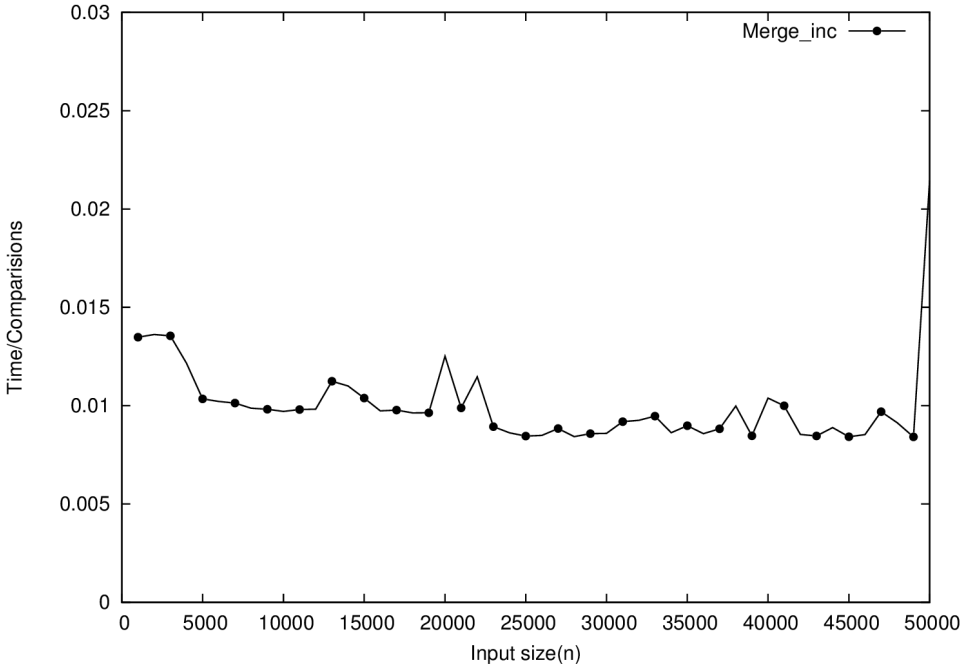


Merge Sort

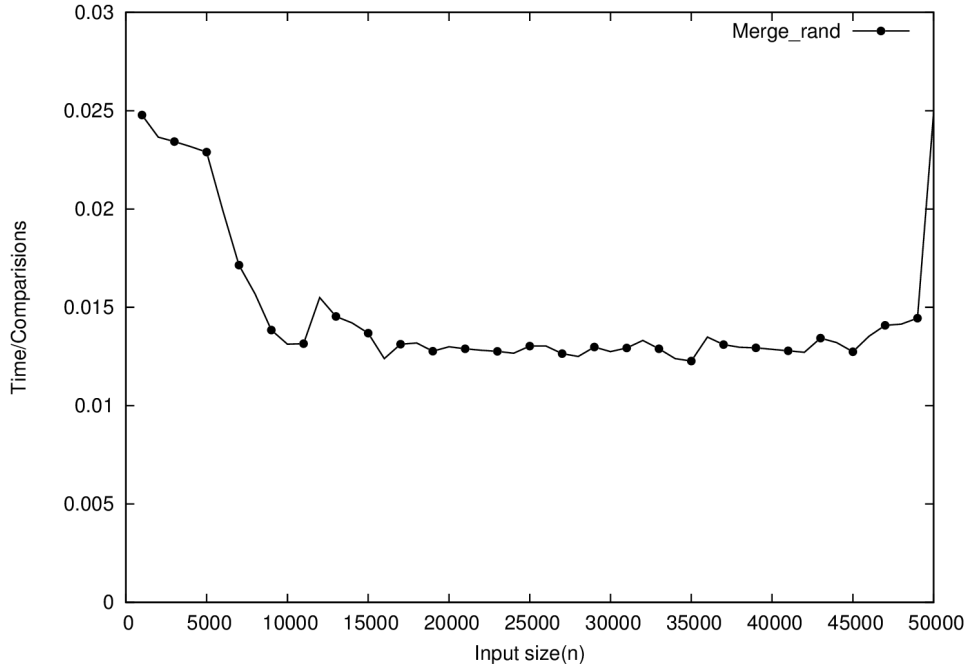
Decreasing values



Increasing values

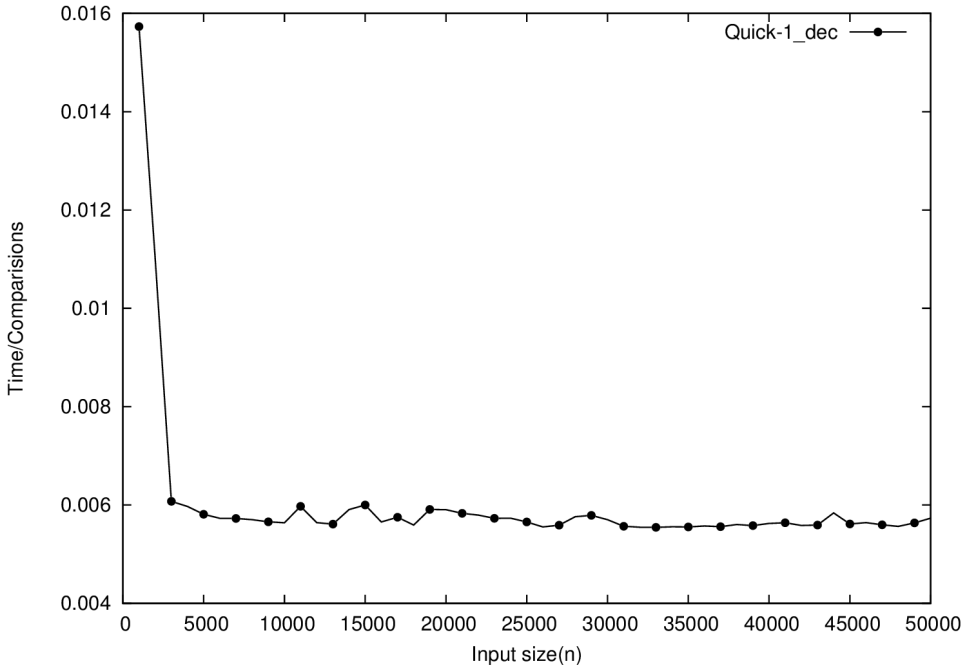


Random values

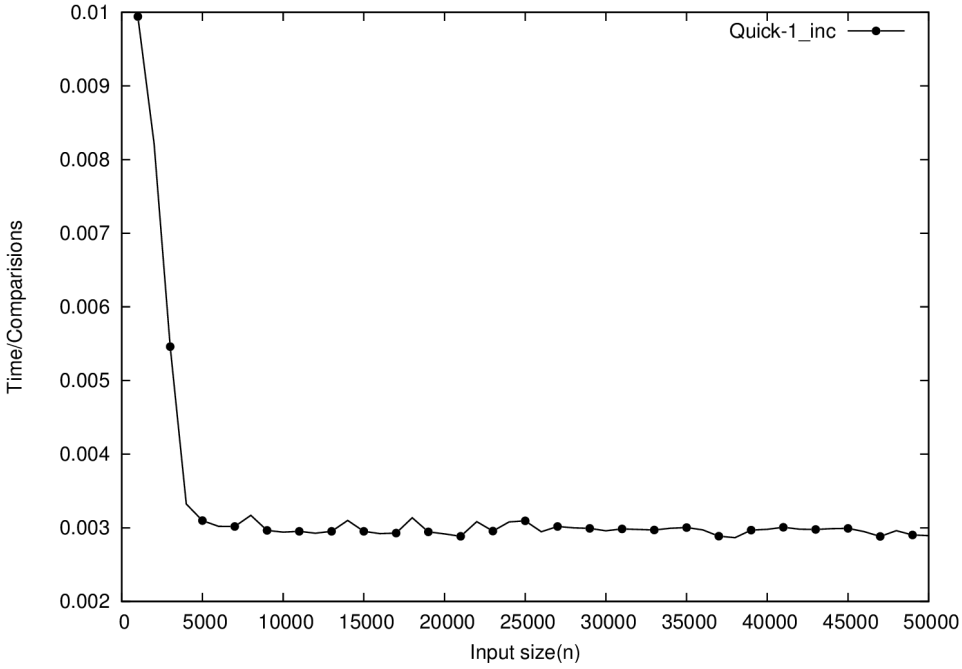


Quick Sort(pivot as first index element)

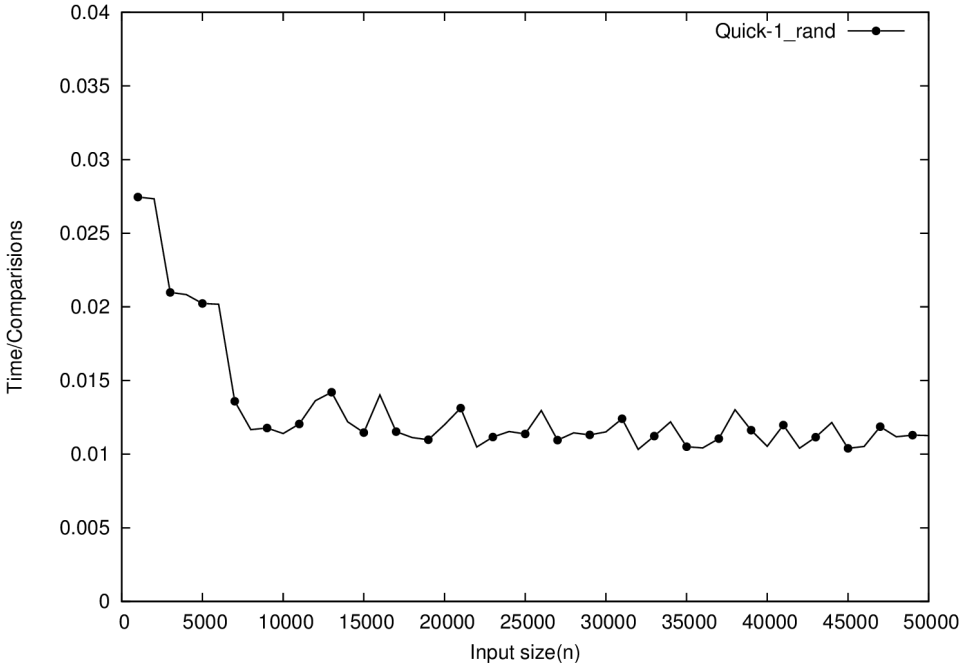
Decreasing values



Increasing values

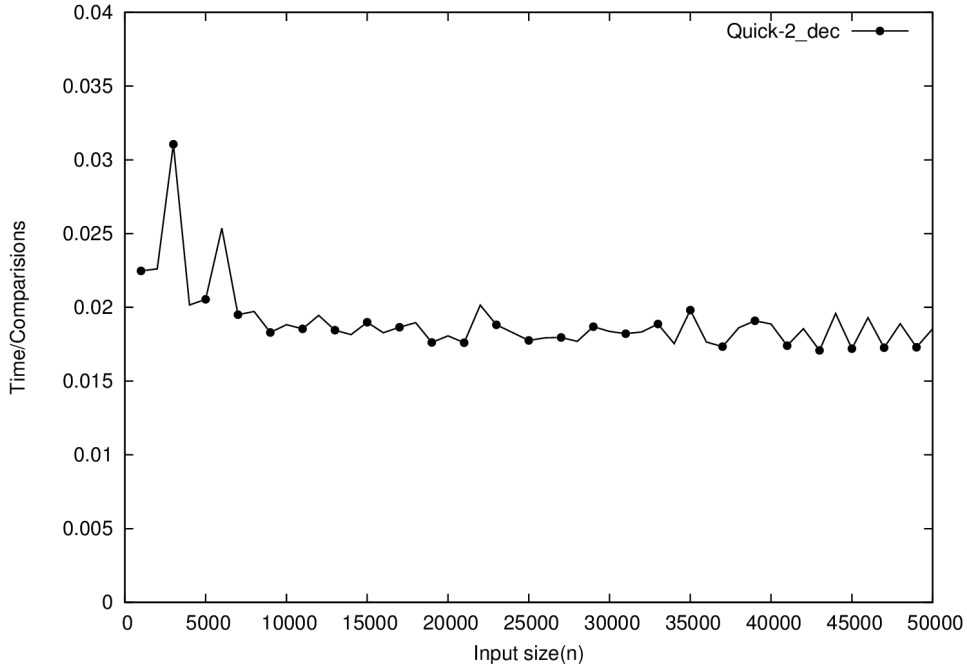


Random values

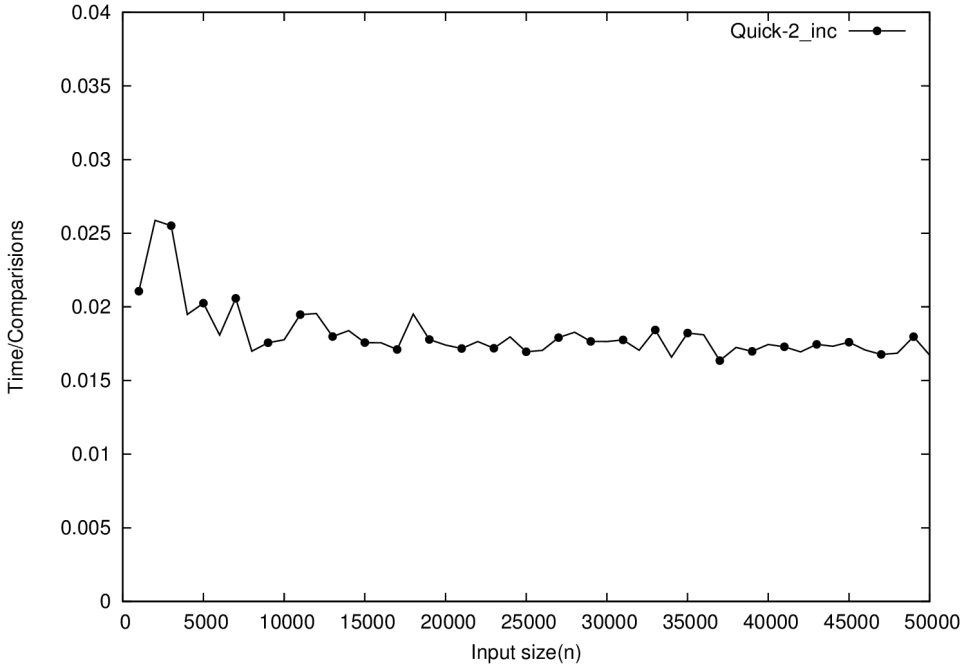


Quick Sort(random element as pivot)

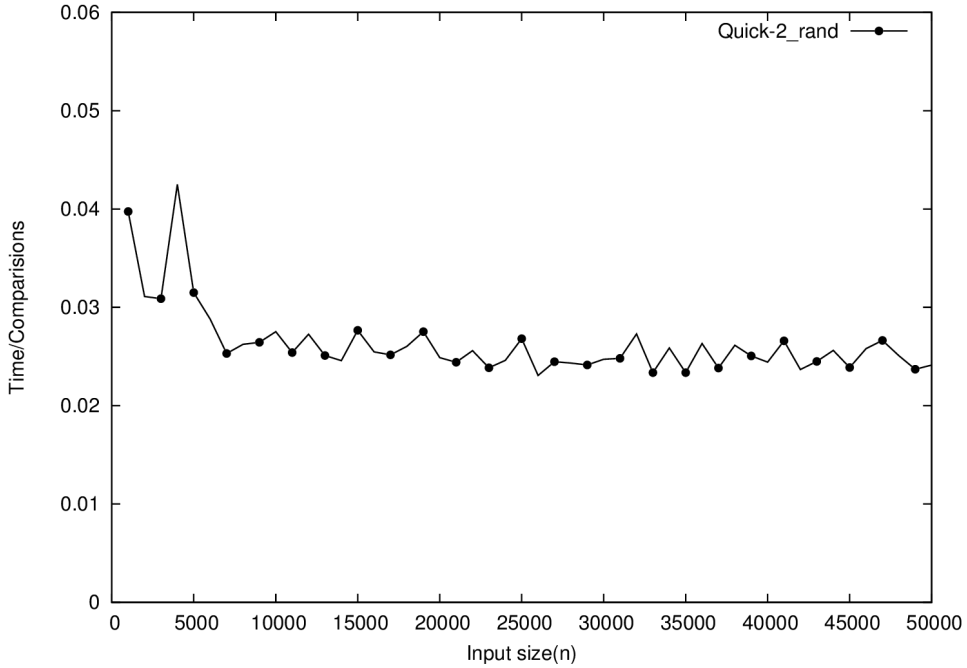
Decreasing values



Increasing values

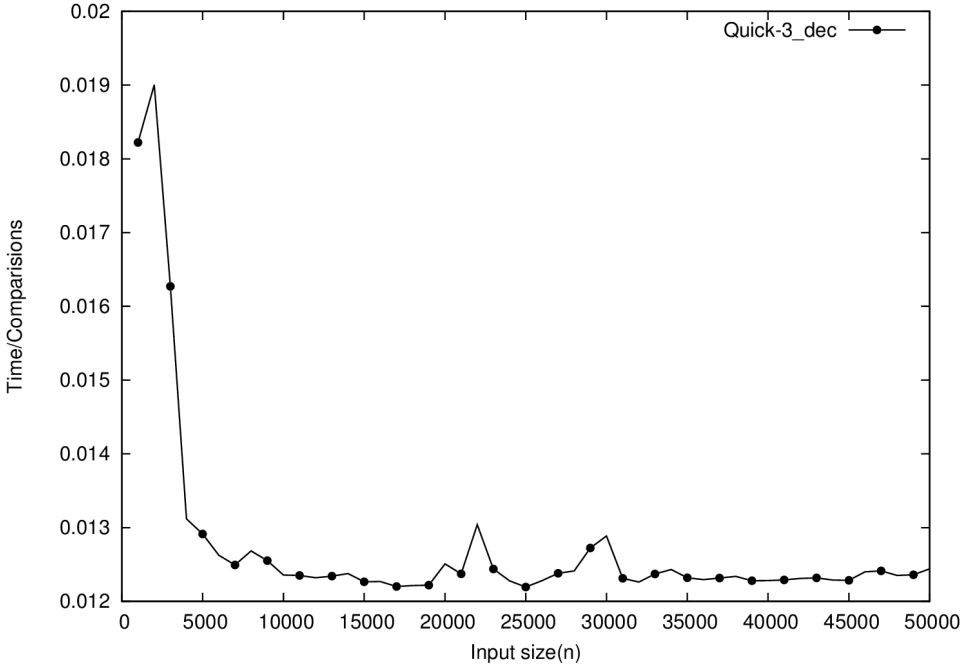


Random values

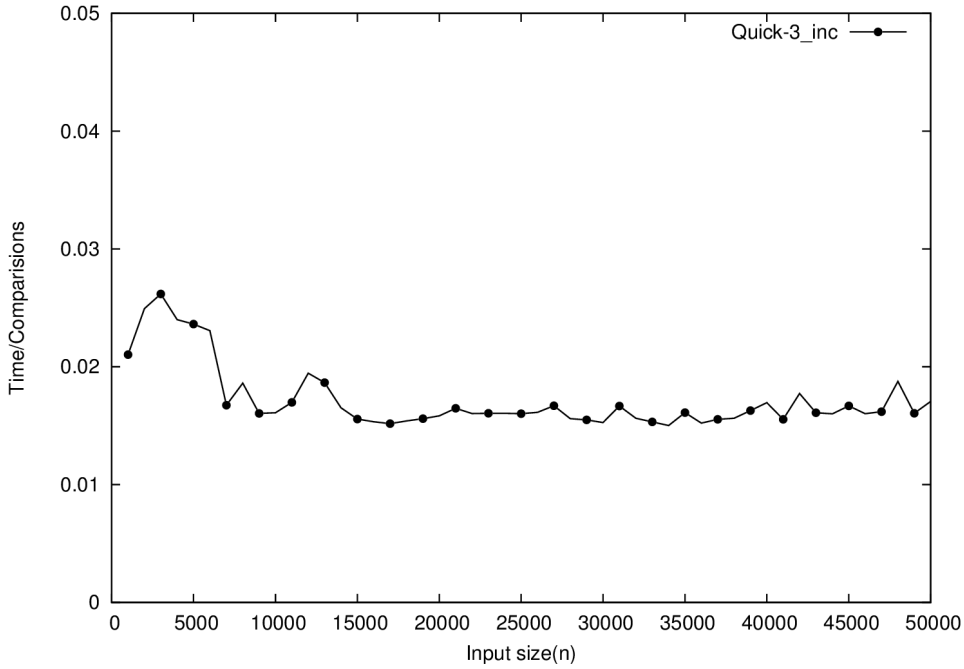


Quick Sort(median element as pivot)

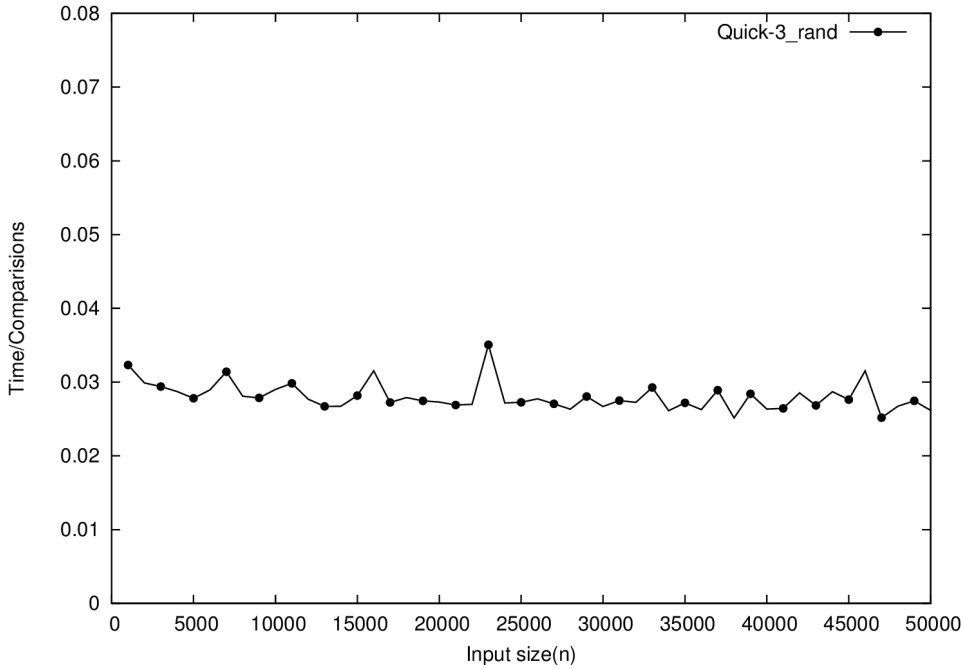
Decreasing values



Increasing values

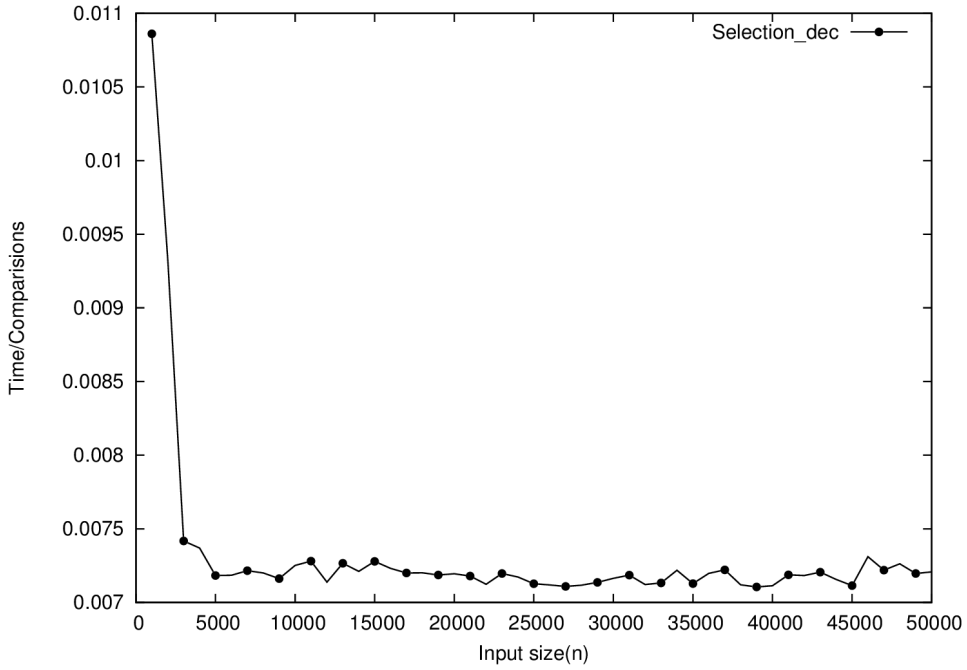


Random values

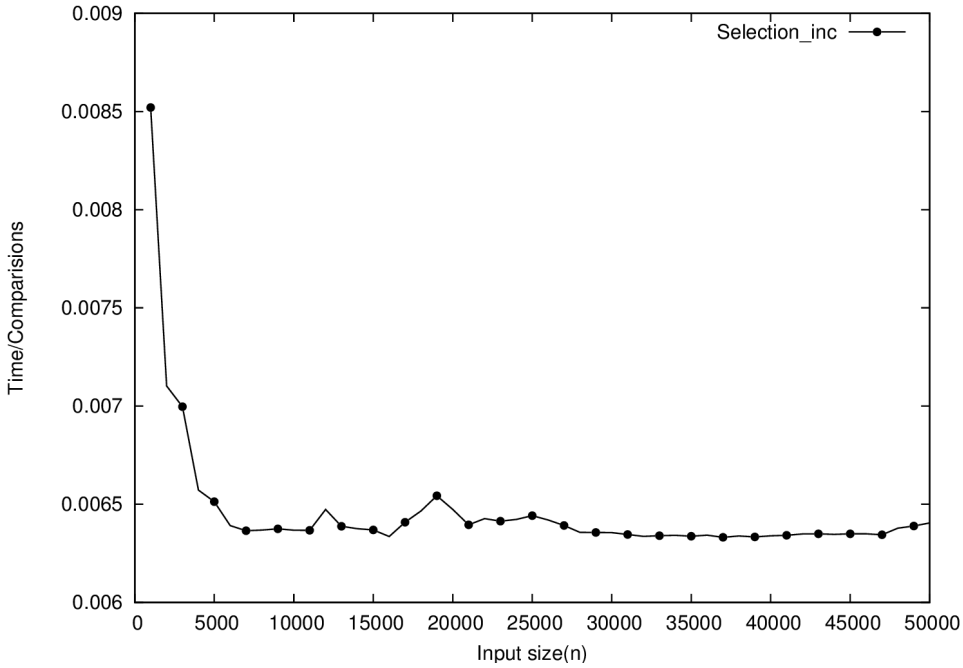


Selection Sort

Decreasing values



Increasing values



Random values