All three methods are in-place sorting algorithms. The algorithms have different time complexity for the best, average, and worst-case:

Time complexity	Heapsort	Quicksort	Insertionsort
Best case	$O(nlog(n))$	$\Theta(nlog(n))$	$\Theta(n)$
Average case	$O(nlog(n))$	$\Theta(nlog(n))$	$\Theta(n^2)$
Worst case	$O(nlog(n))$	$\Theta(n^2)$	$\Theta(n^2)$

We used rangen.py to try the algorithms with different amounts of integers and measured the running time. The results are shown in the tables below.

Running time for 1000 random integers

Heapsort	Quicksort	Insertionsort
0,00546 s.	0,00389 s.	0,0733 s.

Running time for 10 000 random integers

Heapsort	Quicksort	Insertionsort
0,0817	0,0382 s.	8,04 s.

Running time for 30 000 random integers

Heapsort	Quicksort	Insertionsort
0,269 s.	0,128 s.	98,328 s.

As it shows in the tables above, Insertionsort is slower than Quicksort and Heapsort and it becomes slower when the length of the array grows larger.

Insertionsort:
This method works better when the number of elements is small and when the input array are almost sorted.

Heapsort:
This one works fine when we don't need it to be super quick. It's slower than quicksort but it has a worst-case that is guaranteed O(nlog(n)) as for quicksort it's n^2. So Heapsort could be useful in situations when we want a guaranteed O(nlog(n)) performance.

Quicksort:
Quicksort method is very fast and is useful men the running time does not matter (if the worst-case may happen).