|  |  |  |  |
| --- | --- | --- | --- |
|  | losowe | posortowane | Posortowane odwrotnie |
| QS | 0:04:24.017861 | 0:04:34.407152 | 0:04:33.046857 |
| HeapSort | 0:00:00.499538 | 0:00:00.494479 | 0:00:00.513222 |
| BubbleSort | 0:12:37.054901 | 0:12:34.167597 | 0:12:46.607777 |
| InsertionSort | 0:04:21.891009 | 0:00:00.012550 | 0:08:52.833820 |

Liczba danych = 100000

Wygląda na to, iż czasowo najlepiej wypadł HeapSort co prawda nie tak dobrze jak InsertionSort w przypadku posortowanych danych ale nie często sortujemy posortowane dane 😊

QuickSort szkoda wspominać głównie przez Stack Overflow i recursion limit   
resource.setrlimit(resource.RLIMIT\_STACK, (resource.RLIM\_INFINITY, resource.RLIM\_INFINITY))

sys.setrecursionlimit(1500000)

te dwie linijki rozwiązują problem lecz jest to uciążliwe (konieczność używania linuxa).

|  |  |  |  |
| --- | --- | --- | --- |
|  | losowe | posortowane | Posortowane odwrotnie |
| QS | 0:16:38.228504 | 0:16:38.574928 | Jeszcze Liczy |
| HeapSort | 0:00:01.124391 | 0:00:01.130291 | 0:00:01.085495 |
| BubbleSort | 0:54:55.594106 | 0:54:50.540830 | Jeszcze Liczy |
| InsertionSort | 0:18:33.834546 | 0:00:00.028456 | 0:38:09.811364 |

Liczba danych = 200000