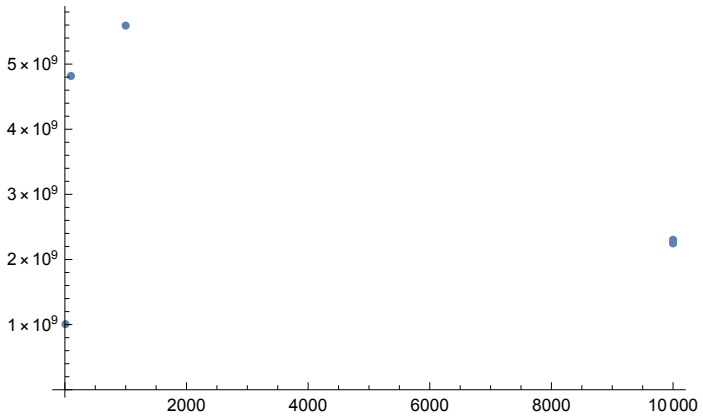
In[390]:= ## SortedUnboundedArray :

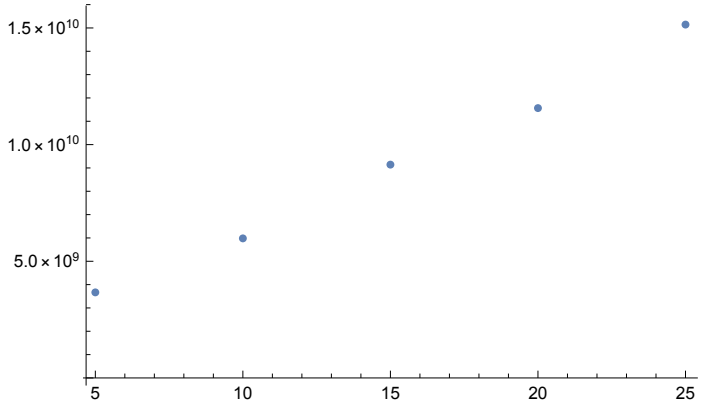
In[386]:=
```
ListPlot[{{10, 1 006 958 535}, {100, 4 818 248 115},
  {1000, 5 590 895 759}, {10 000, 2 302 959 101}, {10 000, 2 247 386 509}}]
```

Out[386]=



In[392]:=
```
ListPlot[{{5, 3 667 020 905}, {10, 5 981 381 522},
  {15, 9 142 299 154}, {20, 11 564 978 300}, {25, 15 144 269 721}}]
```

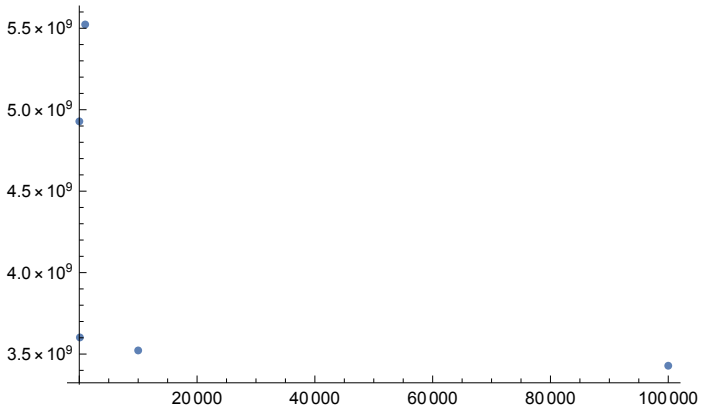
Out[392]=



LinkedList :

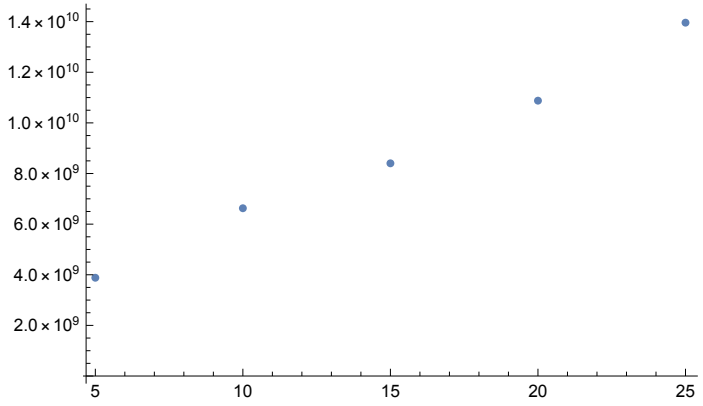
In[390]:=
```
ListPlot[{{10, 4 928 451 545}, {100, 3 602 079 732},
  {1000, 5 523 214 338}, {10 000, 3 522 235 431}, {100 000, 3 428 403 334}}]
```

Out[390]=



In[393]:=
```
ListPlot[{{5, 3 881 985 289}, {10, 6 628 003 488},
  {15, 8 402 615 806}, {20, 10 877 785 670}, {25, 13 959 249 322}}]
```

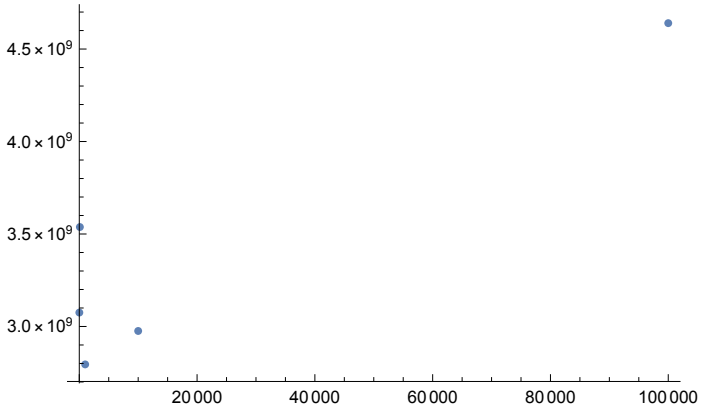
Out[393]=



HashTable:

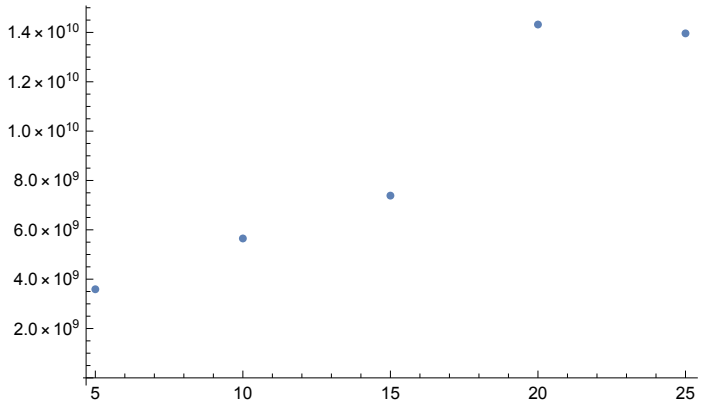
In[388]:=
```
ListPlot[{{10, 3 075 245 424}, {100, 3 537 646 111},
  {1000, 2 794 980 675}, {10 000, 2 975 720 925}, {100 000, 4 639 983 374}}]
```

Out[388]=



In[394]:=
```
ListPlot[{{5, 3 590 928 745}, {10, 5 649 700 029},
  {15, 7 385 997 627}, {20, 14 320 238 592}, {25, 13 961 385 016}}]
```

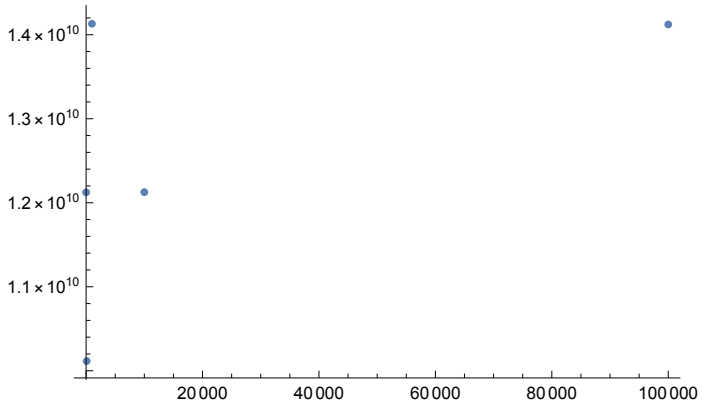
Out[394]=



Priority Queue :

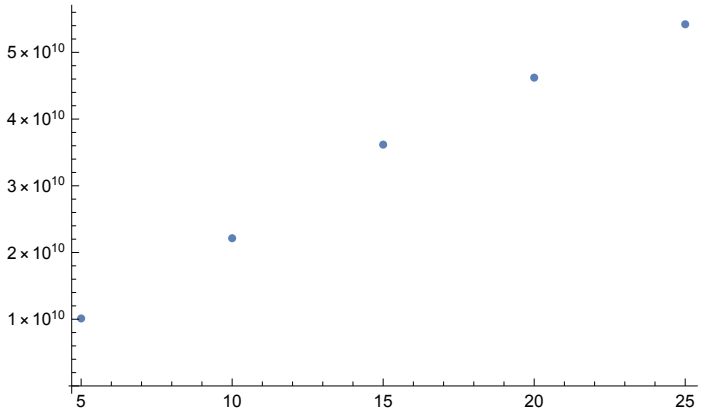
In[389]:=
```
ListPlot[{{10, 12 124 038 820}, {100, 10 115 433 998},
  {1000, 14 130 490 921}, {10 000, 12 125 959 697}, {100 000, 14 122 071 850}}]
```

Out[389]=



In[395]:=
```
ListPlot[{{5, 10 117 597 414}, {10, 22 141 954 699},
  {15, 36 172 105 322}, {20, 46 210 691 863}, {25, 54 211 481 069}}]
```

Out[395]=



First graphs

Observation of the first graphs：
The total number we picked is smaller than the capacity, which means the generator does not need to sleep when the data structure is full since it can not be full.
From the above graphs, we can see that
1) For sorted unbounded array, as we increase the capacity, the runtime first increases but when the size is large enough, it will start to decrease.
2) For linked list, as we increase the capacity, the runtime first increases sharply, then it will start to decrease and remain roughly the same value.
3) For hash table, as we increase the capacity, the runtime first decreases, then it will start to increase.
4) For priority queue, as we increase the capacity, the runtime varies without clear pattern.

Comparison:
1) We can tell from the above observations that both sorted unbounded array and linked list increase first and then at some point, they will decrease.
The reason for sorted unbounded array is that each time we insert and delete, we sort this using insertion sort, for which the worst and average runtime is O(n^2) but the best case is O(n). The comparison inside insertion sort will decrease with more sorted element in the array but when the size is small, it will not cover the cost of reallocate.
The reason for linked list is that when we insert element, we will find the suitable place for the key and insert it so that deletion can be done by simple deleting the head of the linked list. In order to find the place, we need to at most traversal the list and compare them to the key that we want to insert. Therefore, it seems that the runtime should depends on the size of the linked list as the first three points, but there is the possibility that we can quickly find the place is the priority of them is high enough.

2) We can tell from the above observations that hash table decrease first and then at some point, they will increase.
The reason for hash table is that the insertion of it simply depends on the hash function, which basically takes constant time. But the deletion need to findMax, which first need to find first and then traversal the table and do the comparison. At first, there will be lot of null in the list with decrease the probability of findFirst faster. However, it is just the possibility.

3) We can tell from the above observations that priority queue does not have specific turning point since as we change the capacity to 10 times, the runtime does not even become two times. The sort method we use is heapsort which has relatively small runtime. Since the total requests we pick is not so large, the runtime might varies without clear pattern due to small height of the heap.

Second graphs

Observations & Comparison from the graphs:
We can clearly see when the total number of requests increases, the runtime for each data structure will increase. And it seems like there is a linearly correlation between the total number of requests and

the runtime.
When the number of requests increases, we need more time to generate and process. Also, more generated requests need more time to sort, delete, determine where to insert, etc. according to different data structures.