Activity 1. Bubble Algorithm

n	T ordered	T reverse	T random
10000	331	1532	1028
2*10000	1306	5988	4129
2**2*10000	5251	24914	17469
2**3*10000	21250	OoT	OoT
2**4*10000	OoT	OoT	OoT

Bubble Sort has a worst-case and average-case time complexity of $O(n^2)$, which is evident in the exponential growth of execution times as the input size doubles. For **ordered arrays**, Bubble Sort performs slightly better (O(n)) because it can detect early that no swaps are needed. However, for reverse-ordered arrays, it performs the worst, requiring the maximum number of swaps and comparisons. In random arrays, the performance lies between the two extremes, as the number of swaps depends on the initial arrangement.

Activity 2. Selection algorithm

n	T ordered	T reverse	T random
10000	306	325	327
2*10000	1222	1281	1292
2**2*10000	5106	4903	5202
2**3*10000	20784	20954	20318
2**4*10000	OoT	OoT	OoT

Selection sort has a consistent time complexity of O(n^2) across all cases, as evidenced by the times increasing quadratically with input size. For smaller inputs, the algorithm completes in a reasonable time, but as the input size grows, it results in "Out of Time"

(OoT) errors, highlighting its inefficiency for large numbers. The ordered, reversed, and random cases perform similarly, as it always performs the same number of comparisons and swaps.

Activity 3. Insertion algorithm

n	Tordered	Treversed	Trandom
10000	LoR	151	149
2*10000	LoR	573	569
2**2*10000	LoR	2331	2315
2**3*10000	LoR	9766	9281
2**4*10000	LoR	37825	36957
2**5*10000	LoR	OoT	OoT
2**6*10000	LoR	OoT	OoT
2**7*10000	LoR	OoT	OoT
2**8*10000	LoR	OoT	OoT
2**9*10000	91	OoT	OoT
2**10*10000	184	OoT	OoT
2**11*10000	361	OoT	OoT
2**12*10000	722	OoT	OoT
2**13*10000	1477	OoT	OoT

For smaller input sizes, the algorithm performs efficiently in all cases, but as the input size grows exponentially, the ordered and reversed cases show a sharp increase in time complexity, eventually leading to "Out of Time" errors for larger inputs. The algorithm struggles with nearly sorted or reverse-sorted data, likely due to its quadratic time complexity O(n^2) in such cases. Interestingly, the random case performs significantly better, completing in constant time for larger inputs, which implies the algorithm handles randomness more efficiently, possibly due to fewer required comparisons or swaps. This highlights the algorithm's sensitivity to input order and its inefficiency for worst-case scenarios.

Activity 4. Quicksort algorithm

n	Tordered	Treversed	Trandom
250000	LoR	97	94
2*25000	64	190	190
2**2*25000	123	404	402
2**3*25000	256	869	870
2**4*25000	525	1879	1884
2**5*25000	1084	4233	4255
2**6*25000	2231	10327	103332

Quicksort performs efficiently in the random case, as expected, with times increasing linearly or near-linearly, reflecting its average-case complexity of O(nlogn). However, in the ordered and reversed cases, the times grow significantly faster, particularly for larger inputs, indicating a degradation to its worst-case complexity of O(n^2) This occurs because quicksort's performance heavily depends on the choice of pivot; in ordered or reversed data, poor pivot selection leads to unbalanced partitions.

Activity 5. QuicksortInsertion algorithm

n	T random
QuickSort	
QuickSort+Insertion(k=5)	11699
QuickSort+Insertion(k=10)	11215
QuickSort+Insertion(k=20)	11685
QuickSort+Insertion(k=30)	10521

QuickSort+Insertion(k=50)	12031
QuickSort+Insertion(k=100)	9957
QuickSort+Insertion(k=200)	8314
QuickSort+Insertion(k=500)	12379
QuickSort+Insertion(k=1000)	22440

The results obtained for the combined QuickSort and Insertion Sort algorithm on a random vector of size 16,000,000 do not align with the expected behavior. Ideally, for small values of k (e.g., k = 5 or k = 10), the algorithm should perform efficiently, as Insertion Sort is well-suited for small subarrays. However, the execution times show inconsistencies, with some larger k values (e.g., k = 100 or k = 200) performing better than smaller ones, which is unexpected. This suggests a potential flaw in the implementation, such as incorrect pivot selection in QuickSort or improper handling of subarray sizes.