CZ2101 Example Class 1

Hybrid Sort
(Merge-Insertion Sort)

Sahithya, Saori, Joel and Yi Hao

Table of Content

Pseudocode

```
void hybridSort(Array A[], int first, int last, int S)
{
       if((last - first) > S)
      {
            int mid = (first + last)/2 ;
            hybridSort(A, first, mid, S);
            hybridSort(A, mid + 1, last, S);
            merge(A, first, last);
      }
       else
      {
            insertionSort(A, first, last);
      }
}
```

Pseudocode

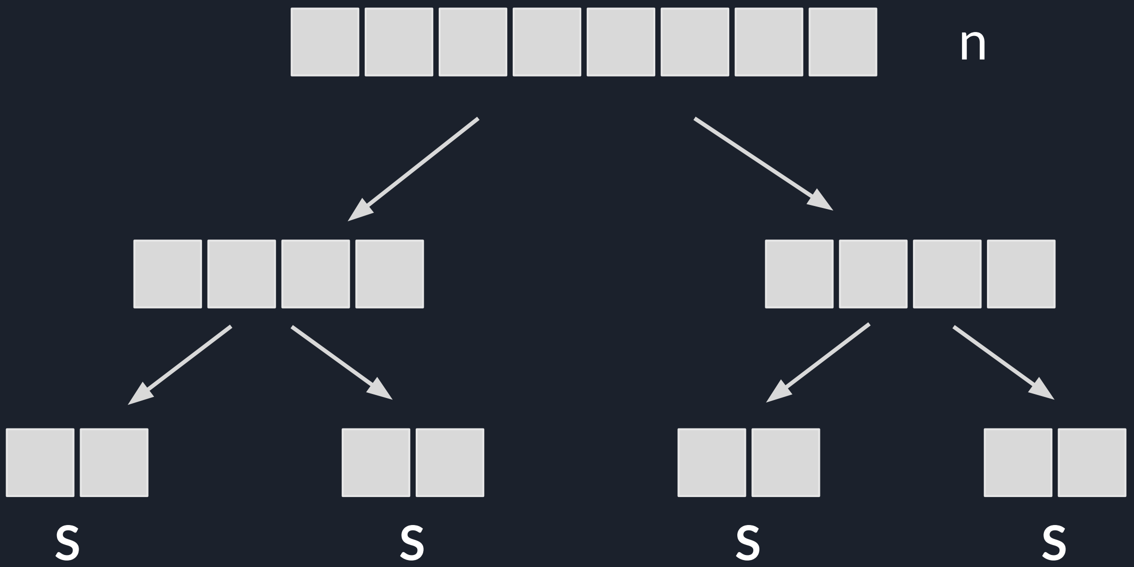
```
void insertionSort(Array A[], int low, int high)
{
      for(int i = low + 1; i<= high; i++)
     {
          for(int j = i; j > 0 ; j--)
         {
              if( A[j] < A[j-1] )
                 swap(A[j], A[j - 1];
              else break;
         {
     {

}
```

Pseudocode

```
void Merge(int Array A[], int n, int m)
{
      int mid = (m + n)/2 ;int a = n; int b = mid + 1;
      while(a <= mid && b <= m){
          if(A[a] < A[b])
             a++;
          else if(A[a] > A[b]){
             int temp = A[b];
             for(int i = b; i>a; i--)
                 A[i] = A[i-1];
             A[a] = temp;
             a++; mid++;
          }
          else {
             for(i = b; i > a;i--)
                 A[i] = A[i-1};
             A[a+1] = A[a];
             a += 2; mid++;
          }
      {
}
```

HybridSort



$\frac{n}{s}$ subarrays

Time complexity (Hybrid Sort)

	Best	Average	Worst
Merge Sort	$O(n \log(n/S))$	$O(n \log(n/S))$	$O(n \log(n/S))$
Insertion Sort	$\Theta(\mathbf{S})$	$\Theta(\mathbf{S}^2)$	$\Theta(\mathbf{S}^2)$
Hybrid Sort	$n \log(n/S) + (n/S)\cdot(S) = \Theta(n \log(n/S) + n)$	$n \log(n/S) + (n/S)\cdot(S^2) = \Theta(n \log(n/S) + nS)$	$n \log(n/S) + (n/S)\cdot(S^2) = \Theta(n \log(n/S) + nS)$

Cases for HybridSort

Best	Average	Worst
Sorted	Random	Reverse
1 2 3 4 5	3 4 1 5 2	5 4 3 2 1

Determining Optimal S value

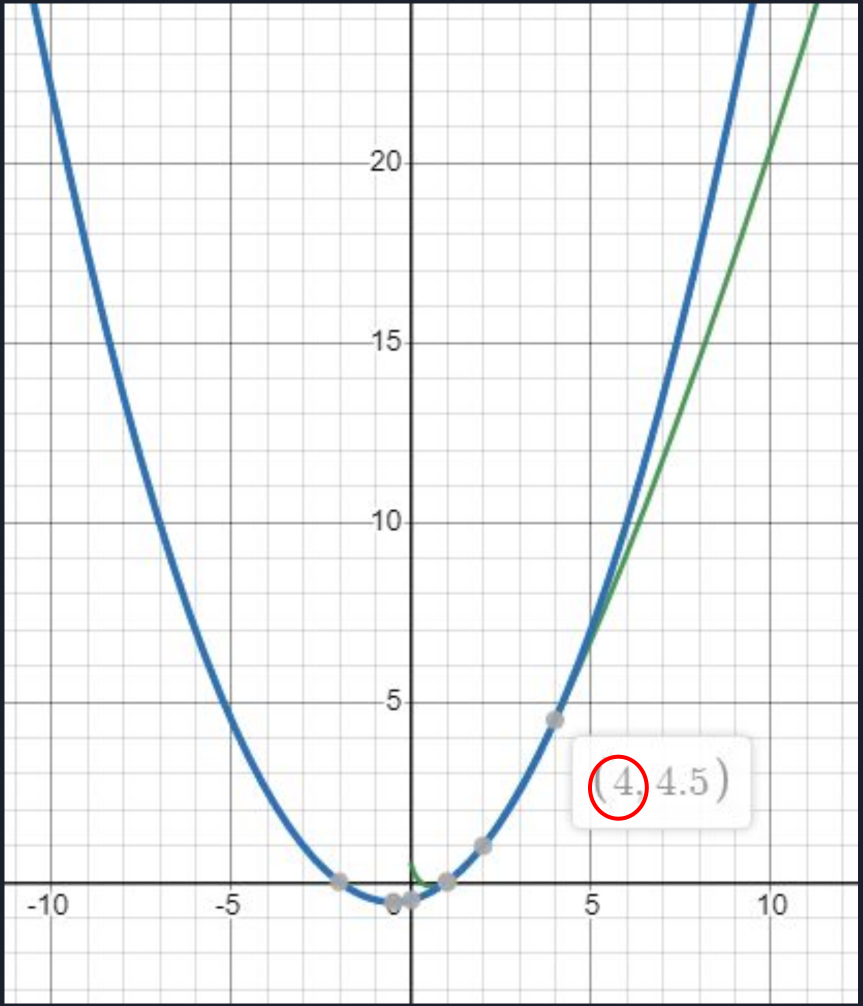
1) Theoretical S
2) Empirical S (CPU Time vs S)
3) Empirical S (Key Comparisons vs S)

Theoretical S value

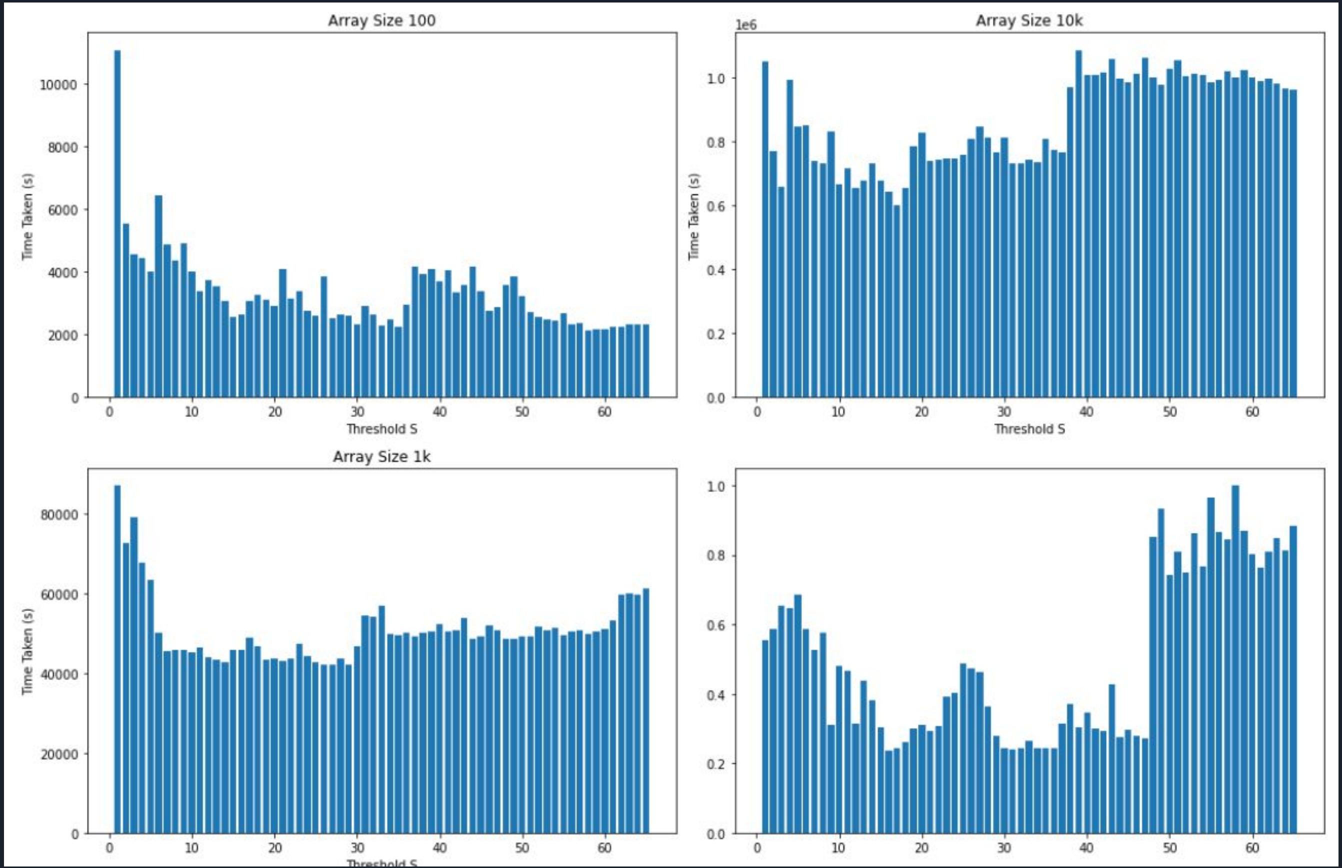
Average Time Complexity (n = Array Size)	
Insertion sort	**Merge sort**
$W(n) = \frac{1}{2}(\frac{n(n+1)}{2} - 1)$	$W(n) = \frac{3}{4}nlog_2(n) - \frac{1}{2}n + \frac{1}{2}$

Insertion sort
$W(n) = \frac{1}{2}(\frac{n(n+1)}{2} - 1)$
Merge sort
$W(n) = \frac{3}{4}nlog_2(n) - \frac{1}{2}n + \frac{1}{2}$

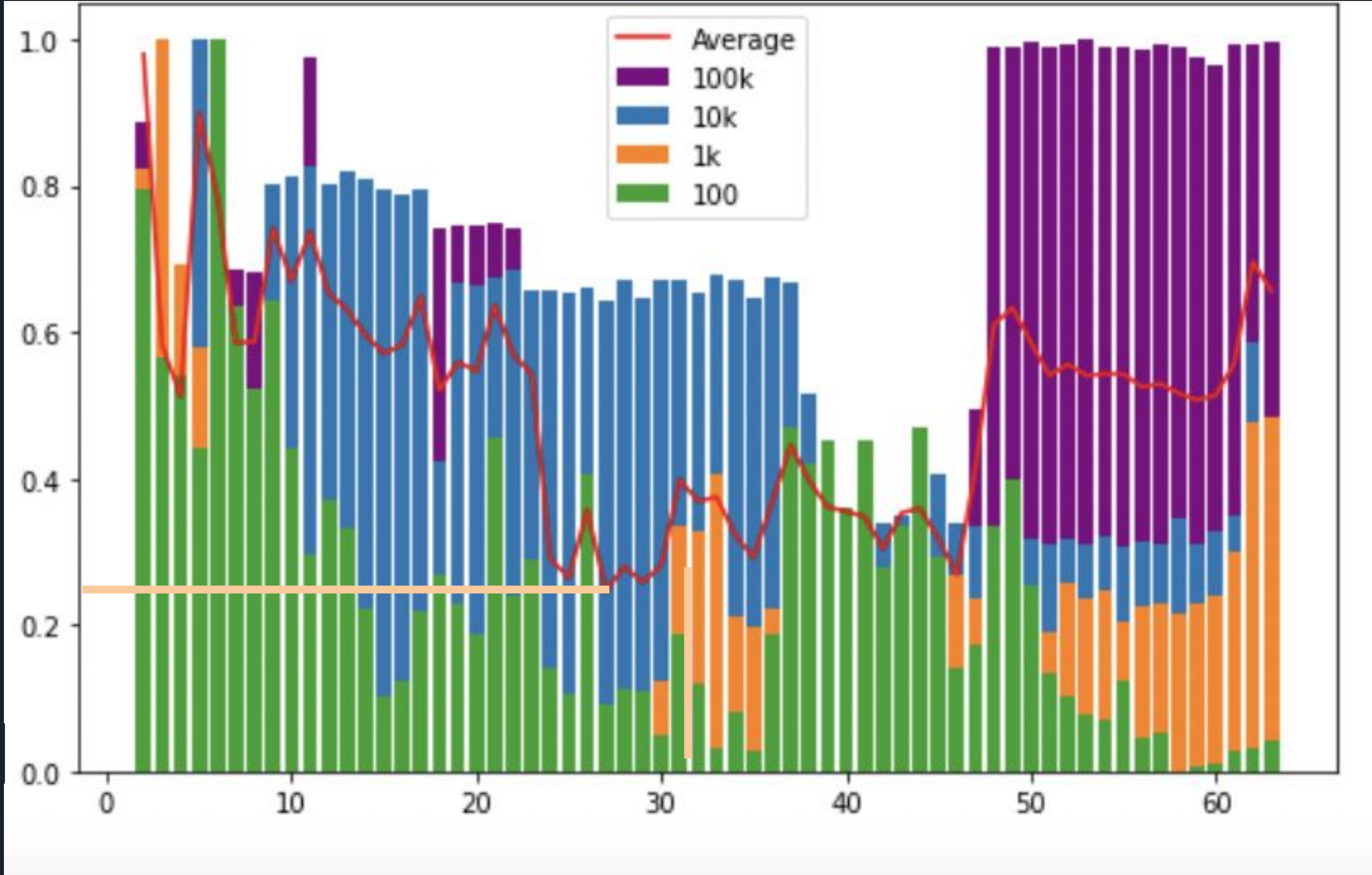
S = 4



Empirical S value (CPU Time)

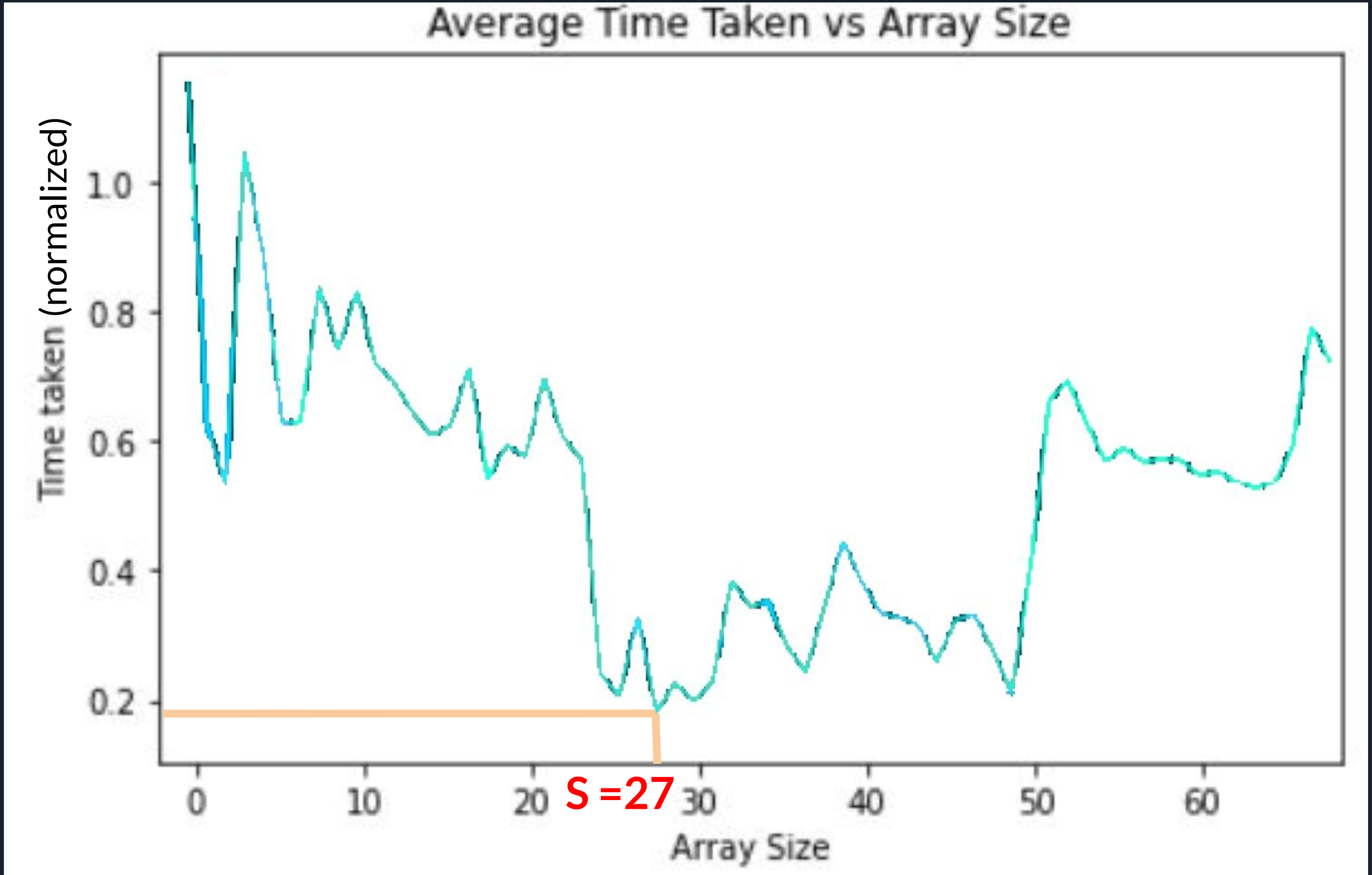


Empirical S value (CPU Time)



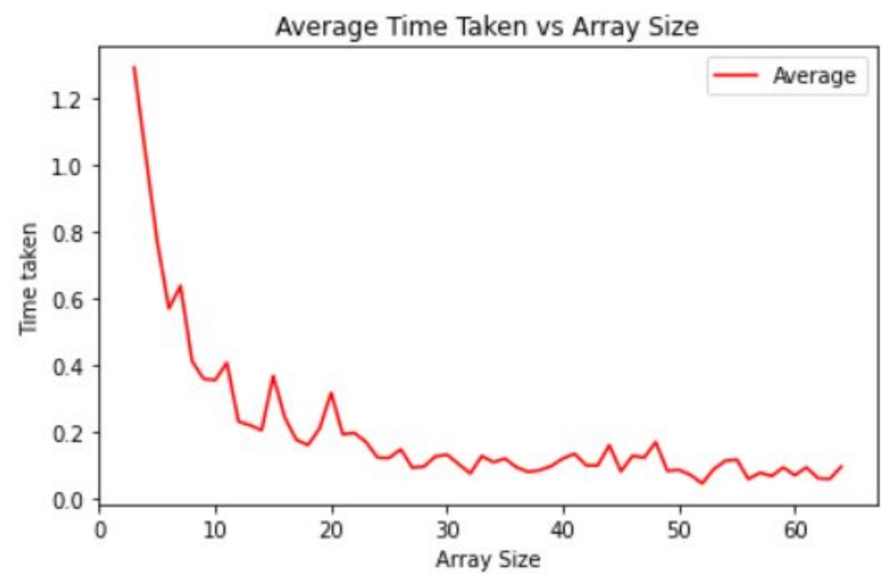
S =27

Empirical S value (CPU Time)

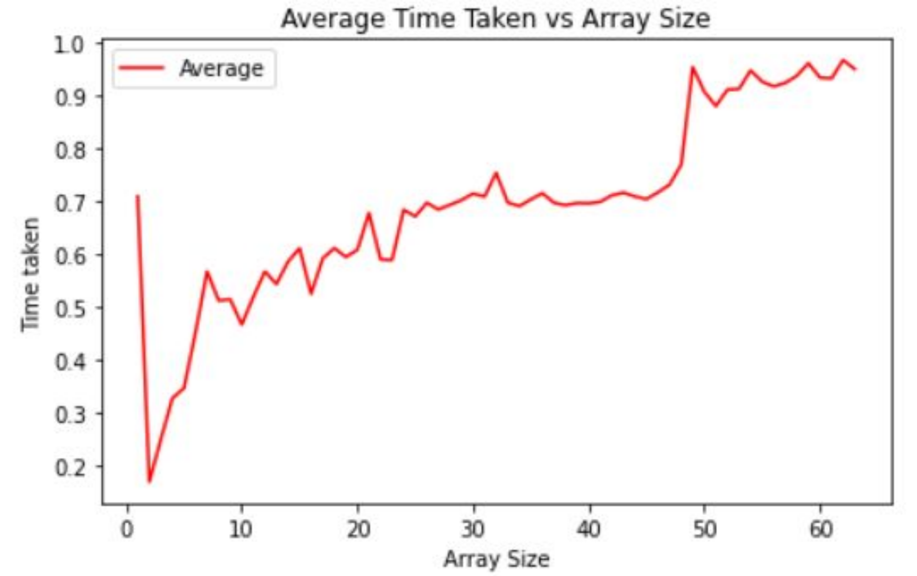


Empirical S value (CPU Time)

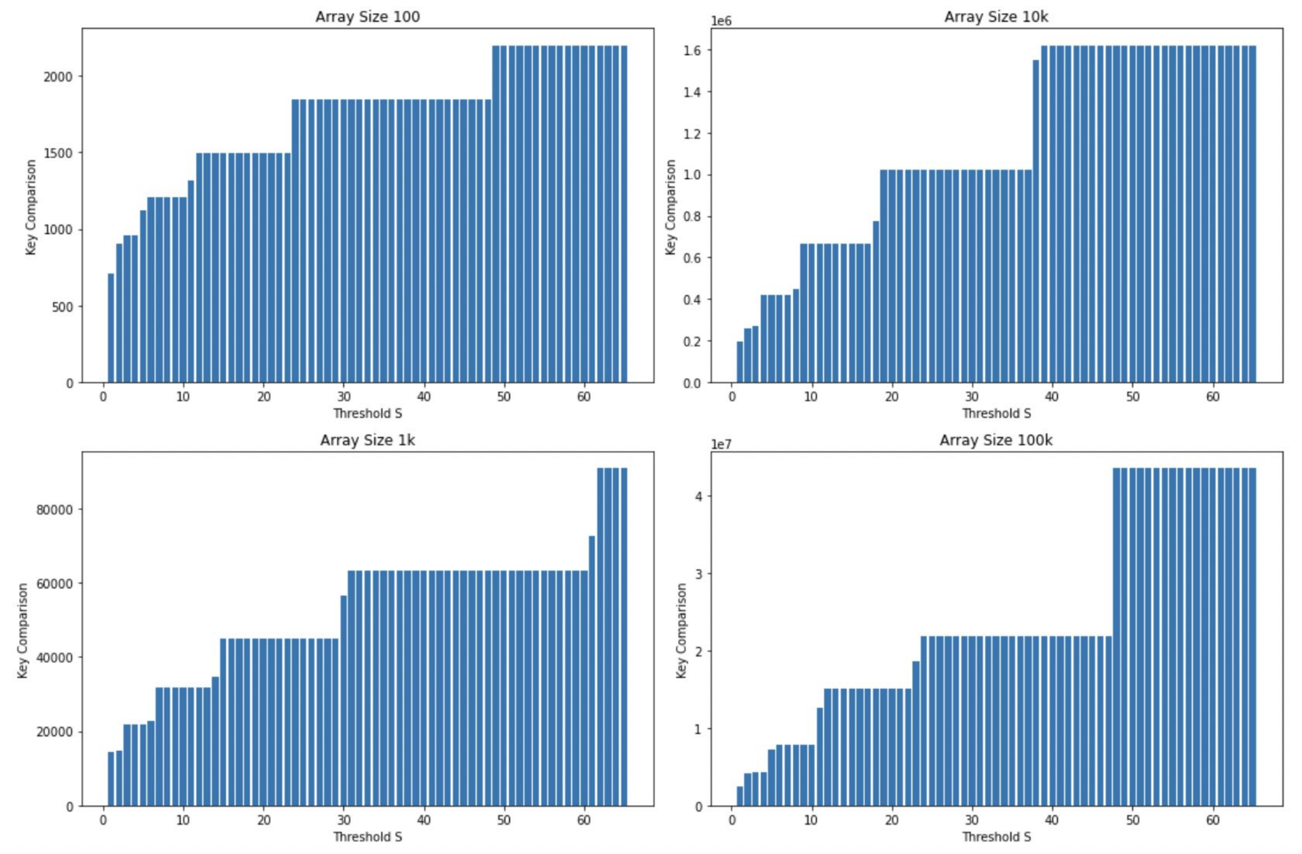
Sorted



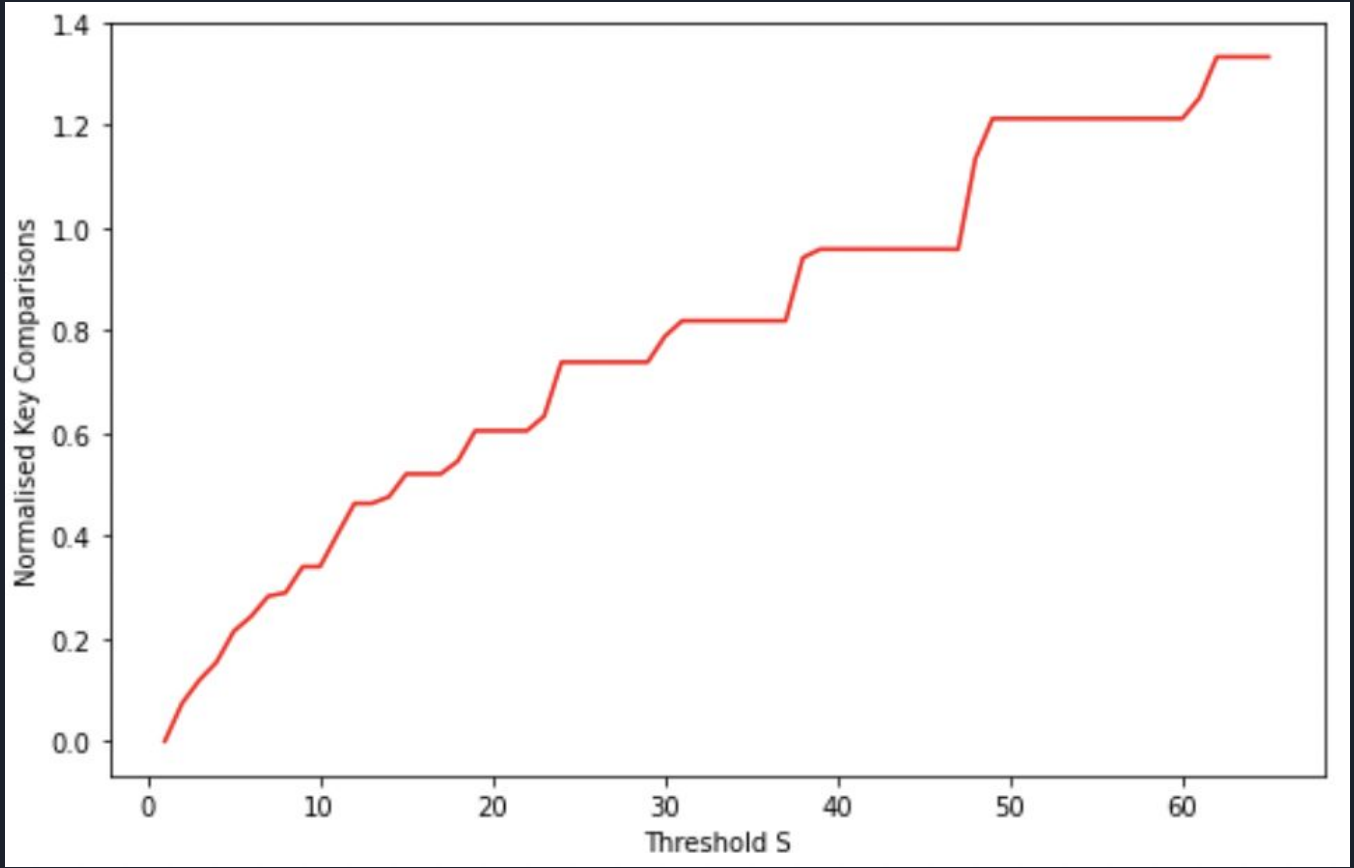
Reverse



Empirical S value (Key Comparisons)

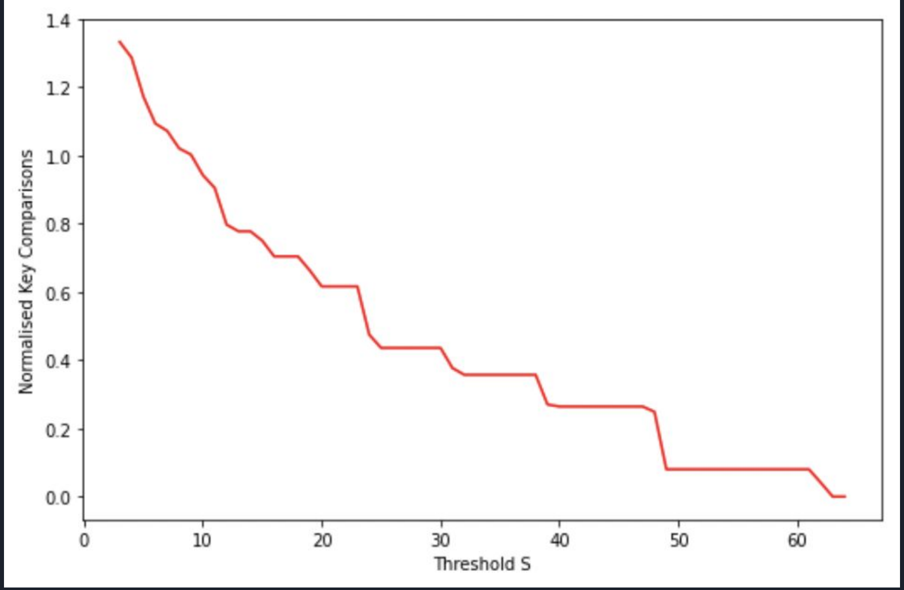


Empirical S value (Key Comparisons)

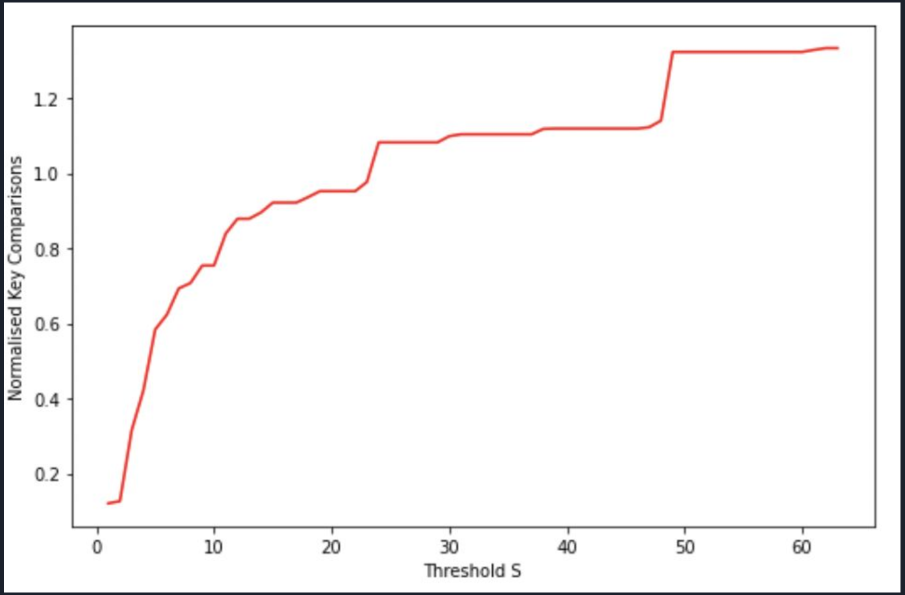


Empirical S value (Key Comparison)

Sorted



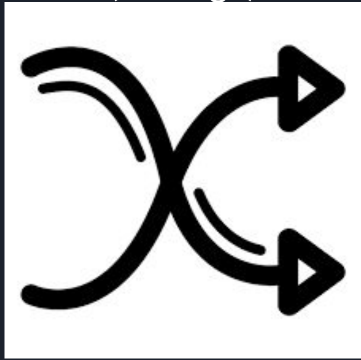
Reverse



Testing our S = 27

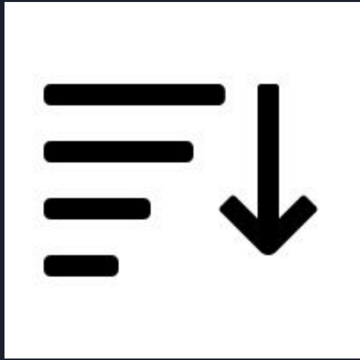
Array Types

Random
(average)



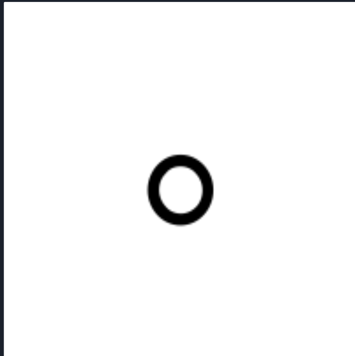
Sorted
(best)

Reverse
(worst)

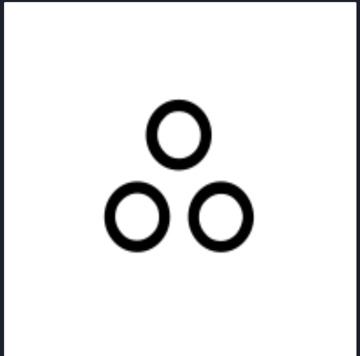


Array Sizes

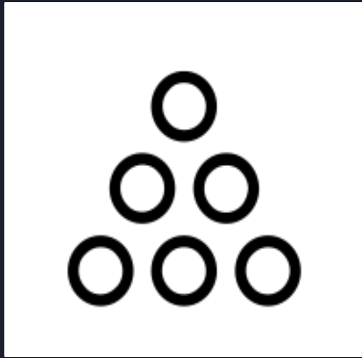
100



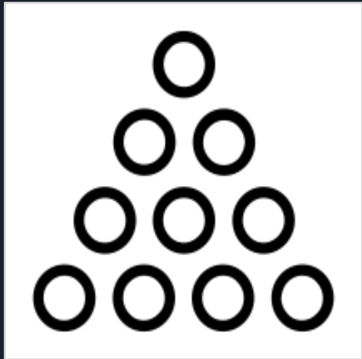
1,000



10,000



100,000

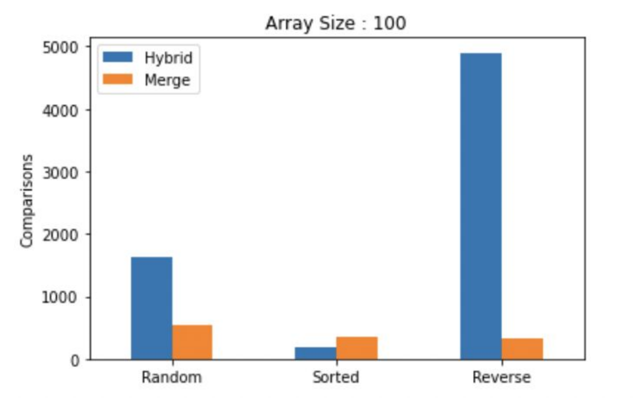


Hybrid Sort VS Merge Sort (Key Comparisons)

Hybrid Sort VS Merge Sort (Key Comparisons)

Array size n = 100

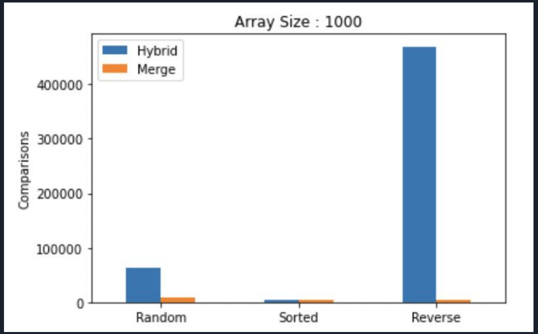
Array Type	Random	Sorted	Reverse
HybridSort	1,625	196	4,900
MergeSort	538	356	316



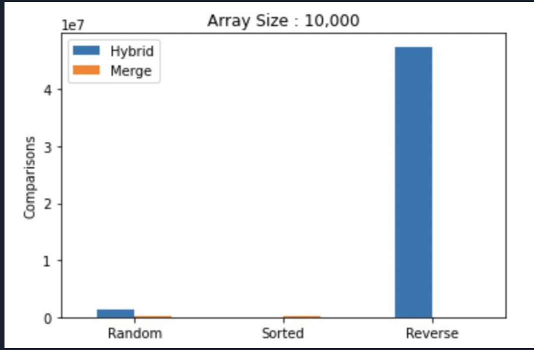
Hybrid Sort vs Merge Sort (Key Comparisons)

Array Size

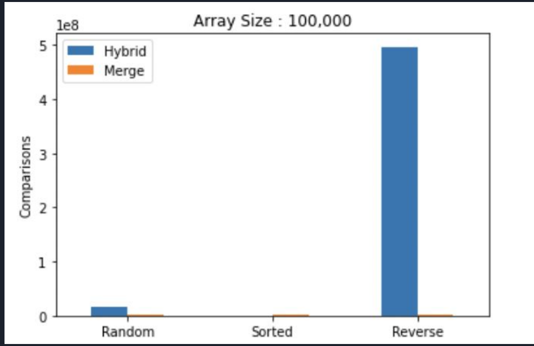
n=1,000



n=10,000



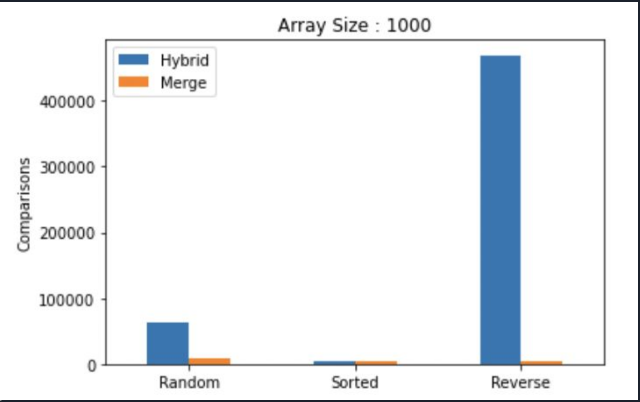
n=100,000



Hybrid Sort VS Merge Sort (Key Comparisons)

Array size n = 1,000

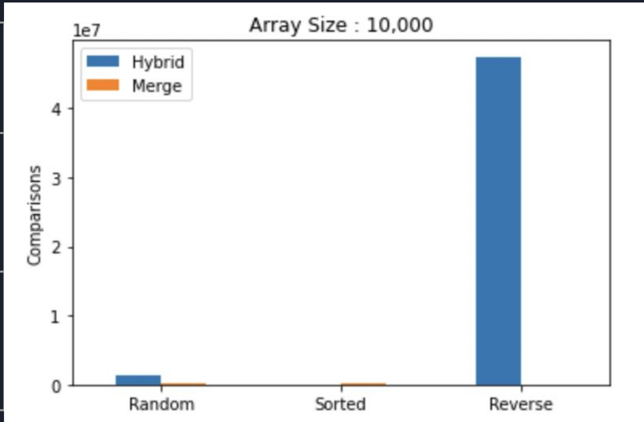
Array Type	Random	Sorted	Reverse
HybridSort	62,721	3,956	469,298
MergeSort	8,694	5,044	4,932



Comparison to MergeSort (Key Comparisons)

Array size n = 10,000

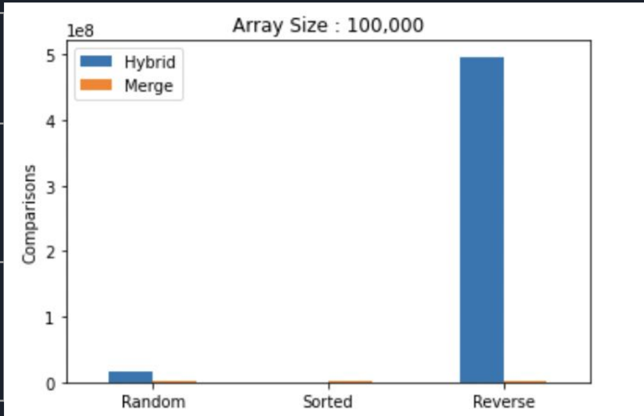
Array Type	Random	Sorted	Reverse
HybridSort	1,009,001	54,640	47,446,397
MergeSort	120,524	69,008	64,608



Hybrid Sort VS Merge Sort (Key Comparisons)

Array size n = 100,000

Array Type	Random	Sorted	Reverse
HybridSort	17,501,059	697,264	496,787,177
MergeSort	1,535,665	853,904	815,024

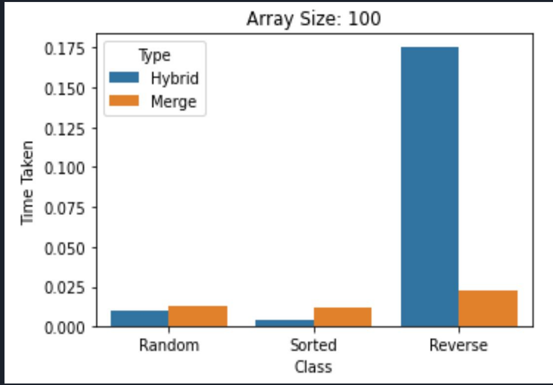


HybridSort VS MergeSort (CPU Time)

Hybrid Sort vs Merge Sort (CPU Time)

Array size n = 100, time taken in ms

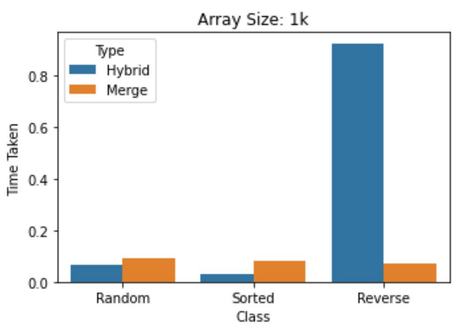
Array Type	Random	Sorted	Reverse
HybridSort	0.00982	0.00385	0.17491
MergeSort	0.01281	0.01182	0.02237



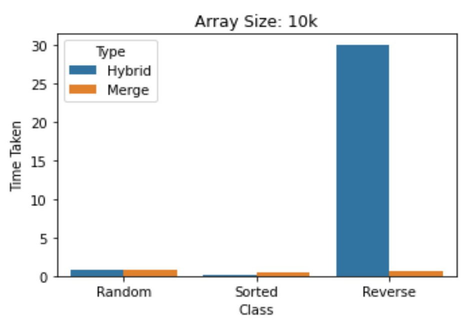
Hybrid Sort vs Merge Sort (CPU Time)

Array Size

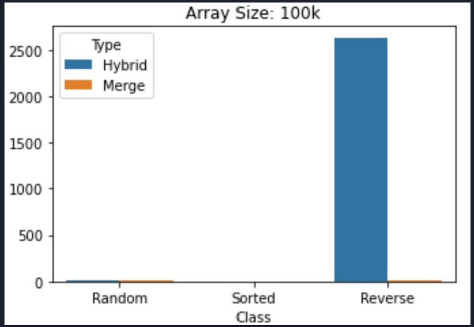
n=1,000



n=10,000



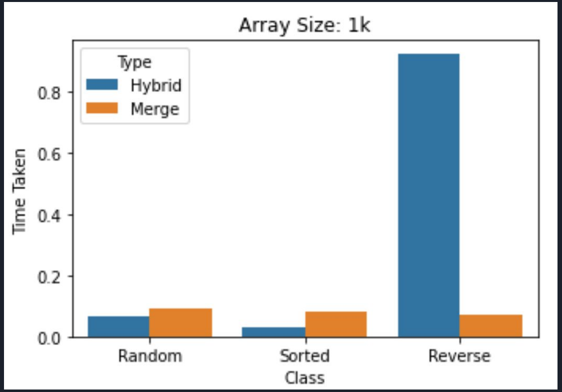
n=100,000



Comparison to MergeSort (CPU Time)

Array size n = 1,000

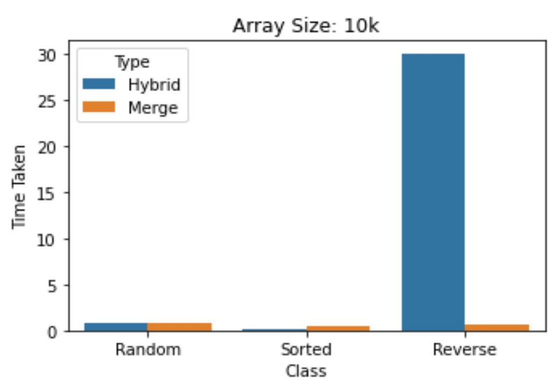
Array Type	Random	Sorted	Reverse
HybridSort	0.06570	0.02992	0.92315
MergeSort	0.09112	0.08288	0.07284



Comparison to MergeSort (CPU Time)

Array size n = 10,000

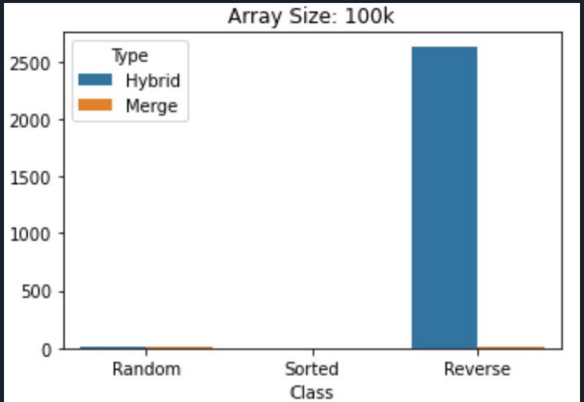
Array Type	Random	Sorted	Reverse
HybridSort	0.73201	0.18465	30.07350
MergeSort	0.84710	0.43182	0.62745



Comparison to MergeSort (CPU Time)

Array size n = 100,000

Array Type	Random	Sorted	Reverse
HybridSort	9.82008	1.56891	2532.11970
MergeSort	10.53474	3.29573	4.63260



CPU Time Percentage improvement

- Comparing the % change in Time taken of the Hybrid algorithm
- Performs better on Random and Sorted arrays, but much poorer on Reverse arrays (due to InsertionSort)

CPU Time % Change of HybridSort against MergeSort				
Array Size (n)	100	1000	10000	100000
Random	-23.4%	-27.9%	-13.6%	-10.7%
Sorted	-31.6%	-63.9%	-56.7%	-52.4%
Reverse	+680%	+1160%	+4900%	+52000%

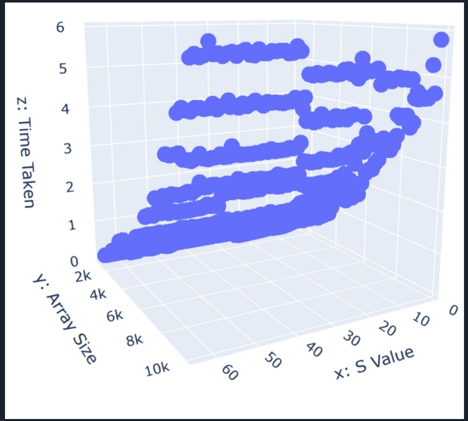
Comparison vs CPU Time

- On average case, Hybrid Merge Insertion sort outperforms Merge sort despite higher comparison
- Constant value differs between the algorithms complexity
- Derivative indicates that Hybrid Sort rate of change is not only dependent on n

	Hybrid Sort	Merge Sort
Time Complexity	a(nS + nlog(n/S)), a is a constant	b(nlogn), b is a constant
Derivative (wrt n)	S + log(n/S) + 1/ln2	log(n) + 1/ln2

Conclusion

- Space complexity O(n) can be achieved with internal sorting
- Empirical S differs from Theoretical S
- Our optimal empirical S range should vary from S = 16 to S = 32
- Algorithm can be further improved by introducing a Galloping Method



Thank You!

Q&A