Project 1

Integration of Merge sort & Insertion Sort

Group 3: Potala, Kiefer, Yi Heng

(a)(ii) Algorithm Implementation - Insertion Sort

```
def insertionSort(array):
    comparisons = 0
    for i in range(1, len(array)):
        key = array[i]
        j = i - 1

        while j >= 0 and array[j] > key:
            comparisons += 1
            array[j + 1] = array[j]
            j -= 1

        array[j + 1] = key

    return comparisons
```

Instantiate key from second element of array

for loop, instantiate j as element before i, range(0, len(array) - 1)

While loop to sort the element with the the left side of the list
E.g 2, 4, 5, 3 -> 2, 4, 3, 5 -> 2, 3, 4, 5

Insert key being compared into its correct position in the current sorted list (left side)

(a)(ii) Algorithm Implementation - Merge Sort (Divide)

```python
def mergeSort(array, comparisons=0):
    if len(array) <= 1:
        return array, comparisons

    mid = len(array) // 2
    left, comparisons_left = mergeSort(array[:mid], comparisons)
    right, comparisons_right = mergeSort(array[mid:], comparisons)

    result, comparisons = merge(left, right, comparisons_left + comparisons_right)

    return result, comparisons
```

Divide complete :
Array cannot be split anymore

Divide and Conquer :
Split left and right into subarrays

Conquer :
Sort left and right subarrays into one array

(a)(ii) Algorithm Implementation - Merge Sort (Conquer)

```python
def merge(left, right, comparisons):
    result = []
    i, j = 0, 0

    while i < len(left) and j < len(right):
        comparisons += 1
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result += left[i:]
    result += right[j:]

    return result, comparisons
```

Create result array, which is the sorted combination of both left and right

Start from left of both subarrays, once one array is exhausted, end loop.

Compare elements from both subarray
Insert smaller element into sorted array and move the index forward for comparison of the rest

Since one array is exhausted, the remaining must be all greater, so insert the rest into result array

(a)(ii) Algorithm Implementation - Hybrid sort

```
def hybridSort(arr, S,  comparisons=0):
      if len(arr) <= S: #if rch limit size, use insertion
        insertionCom = insertionSort(arr)
        return arr, comparisons+insertionCom

      mid = len(arr) // 2
      left, comparisons_left = hybridSort(arr[:mid], S, comparisons,)
      right, comparisons_right = hybridSort(arr[mid:], S, comparisons)

      result, comparisons = merge(left, right, comparisons_left + comparisons_right)

      return result, comparisons
```

Subarray size hits limit ‘S’, switch to insertion sort

If subarray size > S, continue splitting into smaller subarrays

(b)Generate input data

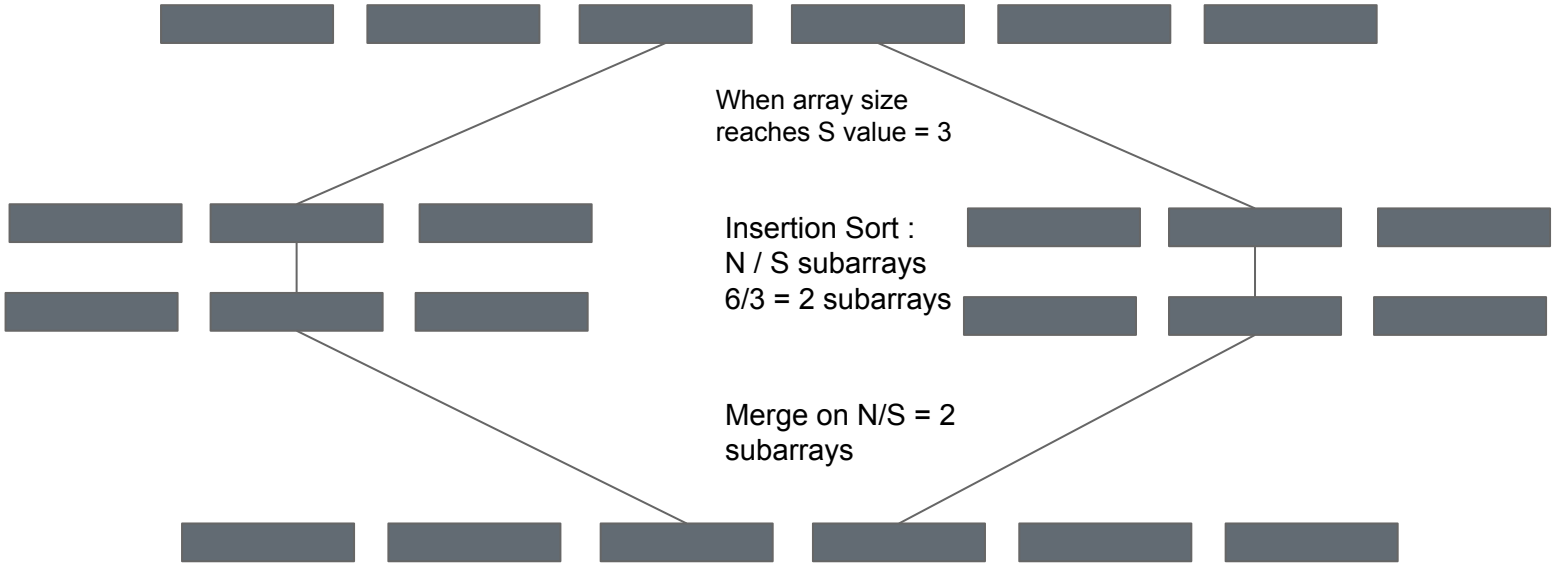
```
def main():
    timeplt = {}
    # varying n, S fixed
    datasizes = [100, 1000, 10000, 100000, 1000000, 10000000]
    # datasizes = [5, 50, 100, 200]
    s_values = list(range(10, 110, 20))
    # s_values = [30]
    s_value = list(range(0,))
    total_N = {}
    execution_times = []
    for s_value in s_values:  # Make graphs of KC vs N with different S values
      comparisons_N = []
      for datasize in datasizes:
        dataset = [random.randint(1, datasize) for _ in range(datasize)]
        #compare with optimal S
        start_time = time.time()
        comparisons = run_sort(dataset, s_value)
        end_time = time.time()
        execution_time = end_time - start_time
```

Use random.randint to generate random integers

(c)(i) Theoretical Analysis : Time Complexity of Merge and Insertion Sort

	Merge Sort	Insertion Sort
Best Case	O(nlog(n))	O(n)
Average Case	O(nlog(n))	$O(n^2)$
Worst Case	O(nlog(n))	$O(n^2)$

(c)(i) Theoretical Analysis : Time complexity of Hybrid Sort



(c)(i) Theoretical Analysis : Time complexity of Hybrid Sort

To derive HybridSort,

Case(Insertion, where n = N/S + Merge, where n = N/S)

Best Case : O(N/S(S) + Nlog(N/S)) = O(N + Nlog(N/S))

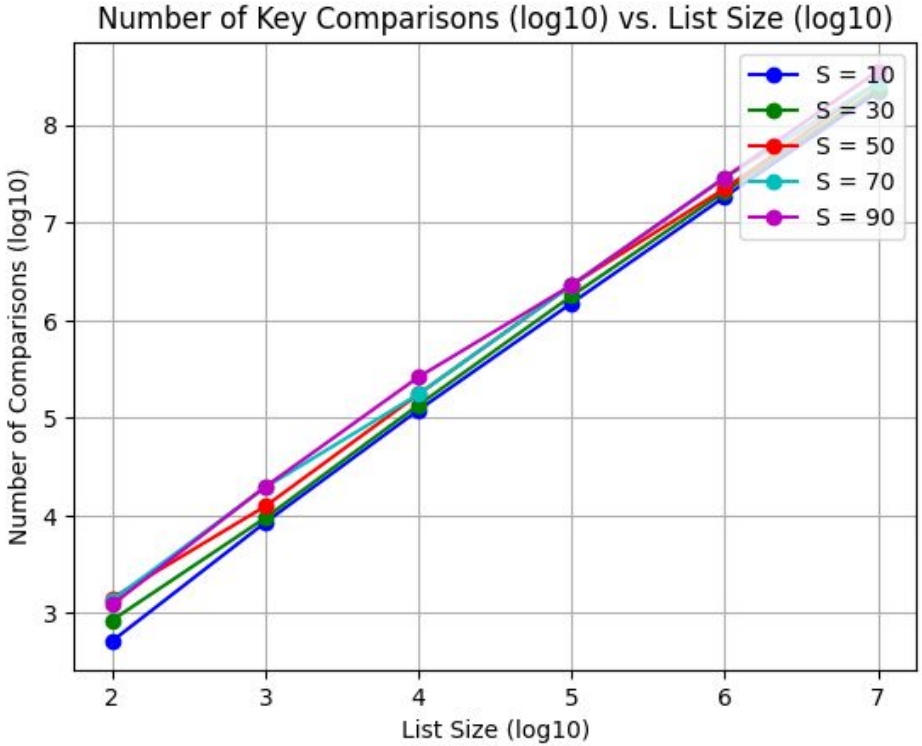
Average Case : O(N/S(S^2) + Nlog(N/S)) = O(NS + Nlog(N/S))

Worst Case : O(N/S(S^2) + Nlog(N/S)) = O(NS + Nlog(N/S))

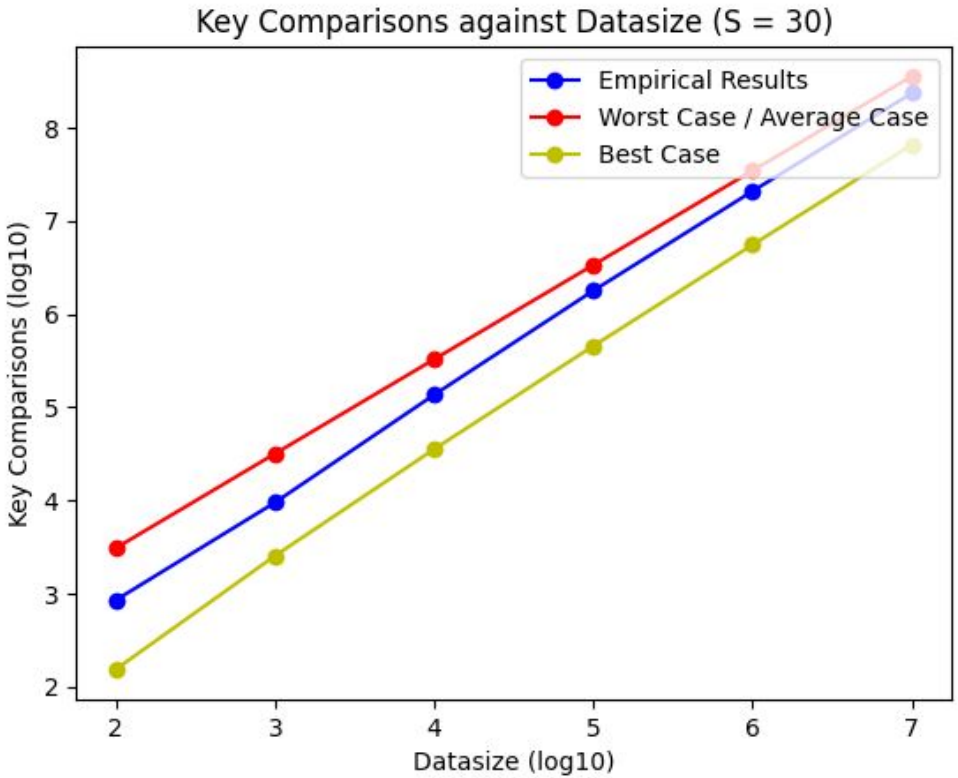
(c)(i) Graph of Key Comparisons against List Size

We plotted five graphs of Key comparisons against List Size

General positive correlation between key comparisons and list size



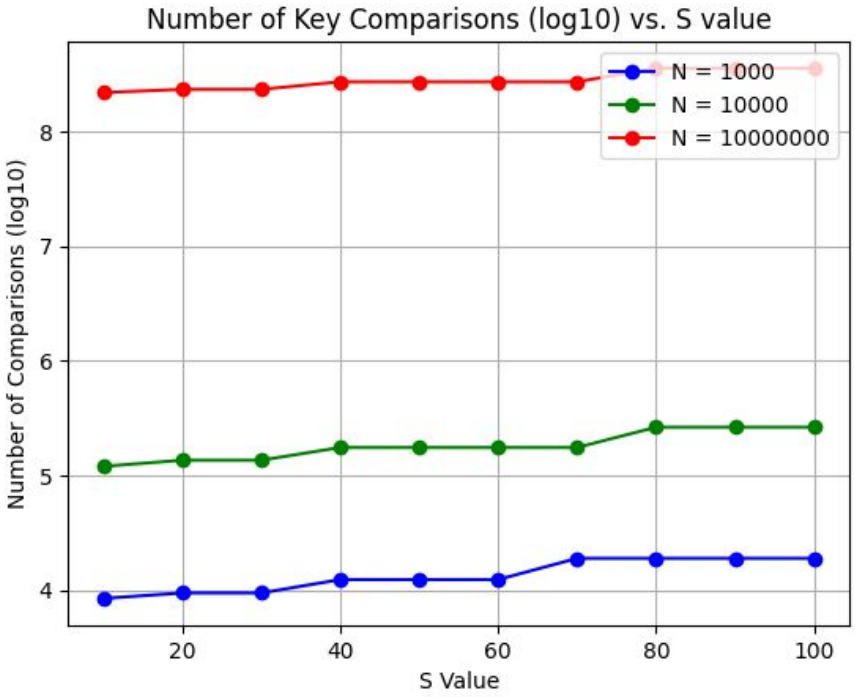
(c)(i) Comparing empirical with theoretical time complexity



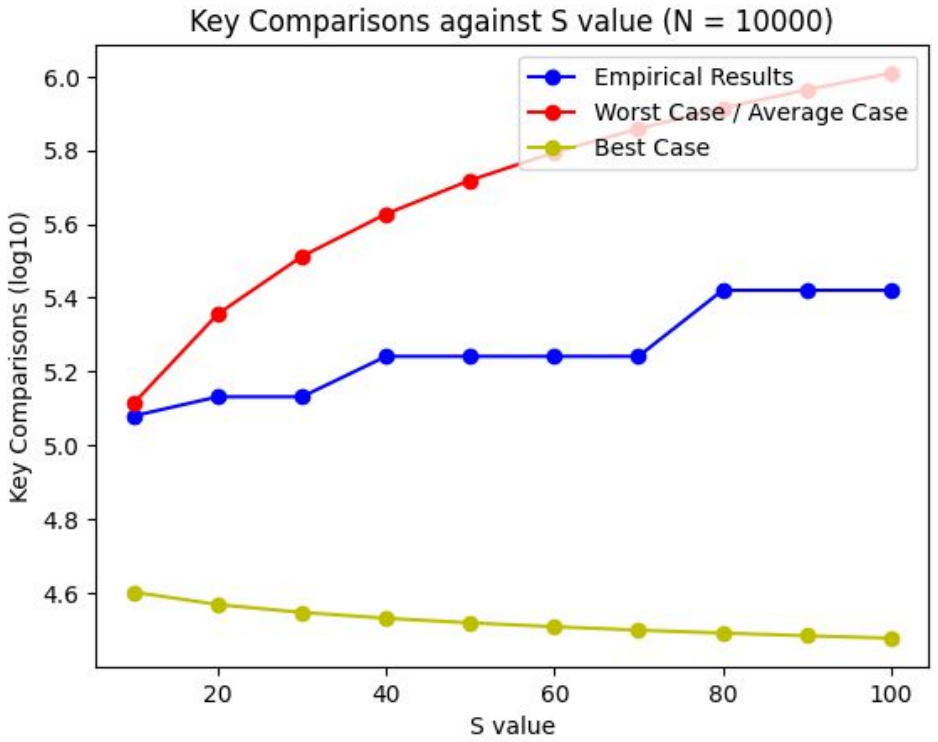
(c)(ii) Graph of Key Comparisons against S

We plotted three graphs of Key comparisons against S

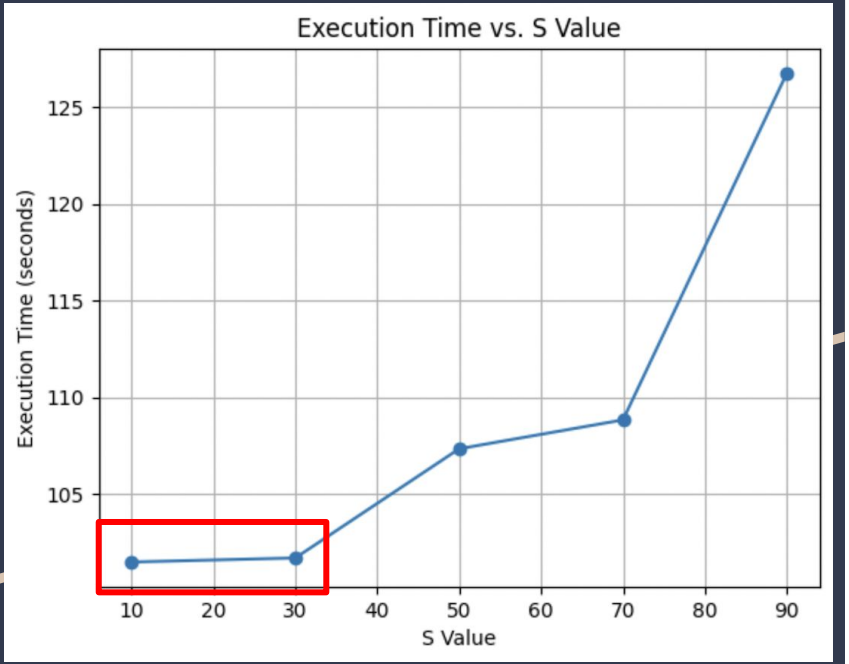
General positive correlation between key comparisons and S value

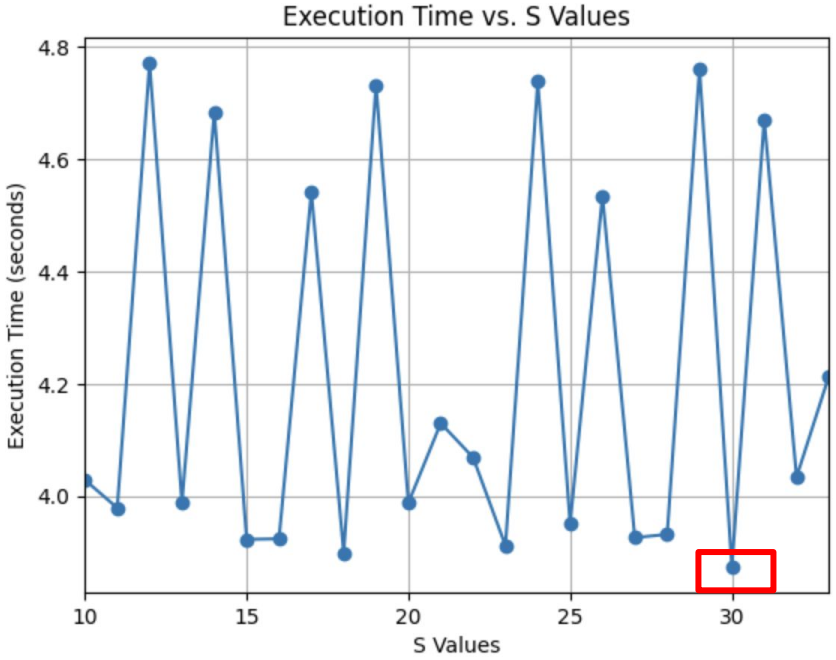


(c)(ii) Comparing empirical with theoretical time complexity



(c)(iii) Determining Optimal Value of S





(d) Compare With Original MergeSort

Using S = 30,

Average runtime of merge > hybrid by ~6.12s

Average key comparisons of hybrid > merge by ~14.1 Million

Run	Hybrid Time(s)	Hybrid Comparisons	Merge Time (s)	Merge Comparisons
1	38.20369768	234,186,077.00	42.28232884	220,099,219.00
2	36.24358535	234,179,331.00	41.29616714	220,100,583.00
3	36.1101501	234,166,103.00	41.34010768	220,102,648.00
4	36.06550932	234,168,587.00	41.46116328	220,099,778.00
5	42.72397804	234,179,013.00	48.21979666	220,098,641.00
6	42.21138477	234,161,868.00	48.46556187	220,097,427.00
7	42.76040959	234,175,144.00	48.92901587	220,100,901.00
8	42.17361879	234,183,612.00	50.13672876	220,102,423.00
9	42.41261292	234,183,602.00	49.20402265	220,101,214.00
10	42.03107071	234,186,273.00	50.84549928	220,100,510.00
Average	40.09360173	234,176,961.00	46.2180392	220,100,334.40
		Avg comp diff	Avg time diff	
		14,076,626.60	6.124437475	
		Hybrid > Merge	Merge > Hybrid	