Example Project 1
Group 1

Lim Li Ping Joey, U2321331C
Vinodhithaa, U2322818J

a) Algorithm Implementation - hybridSort()

```
void hybridSort(int start, int end, int arr[], int S)
{   int mid;
    if(end-start+1> S)
    {
        mid = (start+end)/2;

        hybridSort(start,mid,arr,S);
        hybridSort(mid+1,end,arr,S);

        // merge
        merge(arr, start, mid, end);

}
    else //length of array smaller than S
    {
        insertionSort(start,end,arr);
    }
    return;
}
```

a) Algorithm Implementation - merge()

```
void merge(int arr[], int start, int mid, int end) {
        int i, j, k;
        // size of left array
        int n1 = mid - start + 1;
        // size of right array
        int n2 = end - mid;
        int *leftArr = malloc(sizeof(int) * n1);
        int *rightArr = malloc(sizeof(int) * n2);

        // copy data to 2 separate arrays so we can
merge   for (i = 0; i < n1; i++)
            leftArr[i] = arr[start + i];
        for (j = 0; j < n2; j++)
            rightArr[j] = arr[mid + 1 + j];

....
```

a) Algorithm Implementation - merge()

```
...
// merge the 2 arrays back into arr from arr[start] to arr[end]
        i = 0;
        j = 0;
        k = start;
        while (i < n1 && j < n2) {
            comp++;
            if (leftArr[i] <= rightArr[j]) {
                arr[k] = leftArr[i];
                i++;
            }
            else {
                arr[k] = rightArr[j];
                j++;
            }
            k++;
        }

        // if leftArr not empty, copy remaining elements
        while (i < n1) {
            arr[k] = leftArr[i];
            i++;
            k++;
        }

        // if rightArr not empty, copy remaining elements
        while (j < n2) {
            arr[k] = rightArr[j];
            j++;
            k++;
        }
        free(leftArr);
        free(rightArr);
}
```

a) Algorithm Implementation - insertionSort()

```
void insertionSort(int start, int end, int
arr[])
    int i,j,temp;

    for(i=start+1;i<=end;i++)
    {
        for(j=i;j>start;j--)
        {
            comp++;
            if(arr[j]<= arr[j-1])
            {
                temp= arr[j];
                arr[j]= arr[j-1];
                arr[j-1]=temp;
            }
            else
                break;
        }
    }
    return;
}
```

b) Data Generation

Generated 7 datasets in the range of [0, 10,000,000) with seed=2001

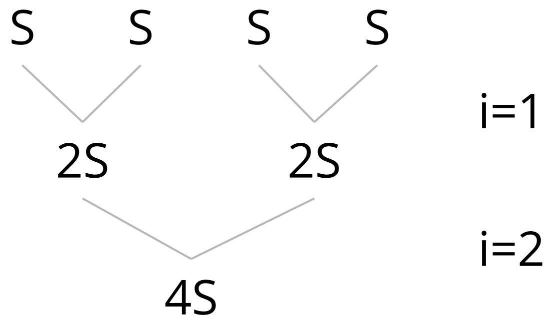
```
from numpy import random
max = 10000000
random.seed(2001)
with open("dataset.txt", 'w') as f:
    for cap in [1000, 10000, 100000, 500000,
                1000000, 5000000, 10000000]:

        arr = random.randint(max,size=(cap))
        string = ",".join(map(str, arr))
        f.write(string)
        f.write("\n")
```

c) Theoretical Analysis

Merge Sort

Let i = number of iterations of MergeSort function

$$2^i(S) = n$$
$$2^i = \frac{n}{S}$$
$$i \log_2 2 = log_2(\frac{n}{S})$$
$$i = \log_2(\frac{n}{S})$$



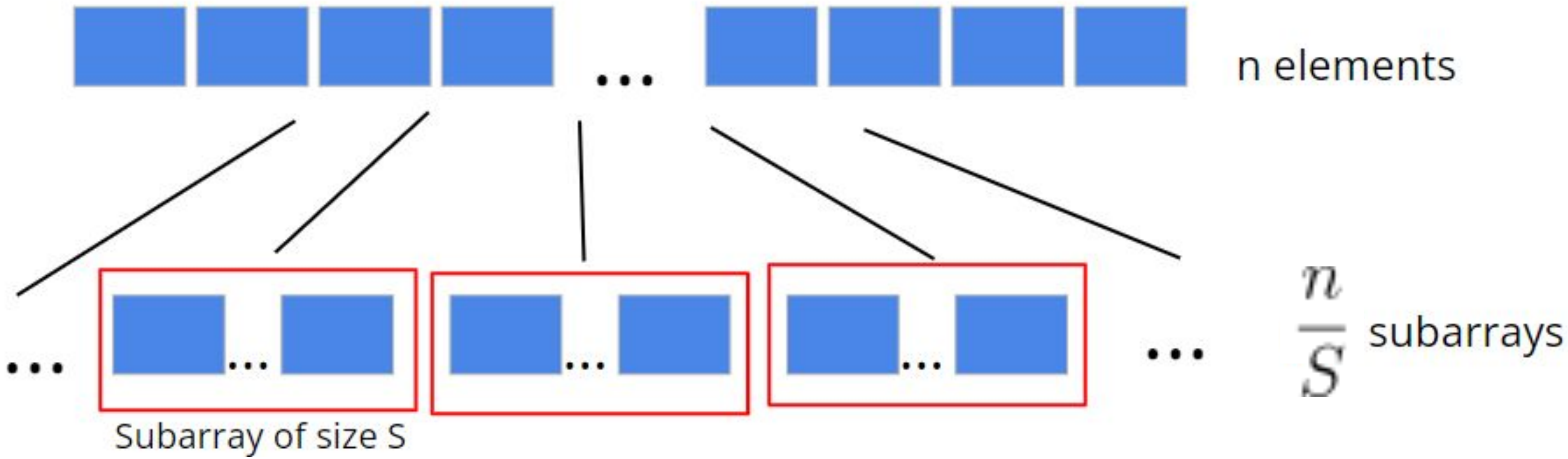
- Number of comparisons per iteration= O(n)

- Time complexity = $n \log_2(\frac{n}{S}) = O[nlog_2(\frac{n}{S})]$

c) Theoretical Analysis

Insertion Sort

- Number of subarrays to sort: $\frac{n}{S}$



c) Theoretical Analysis

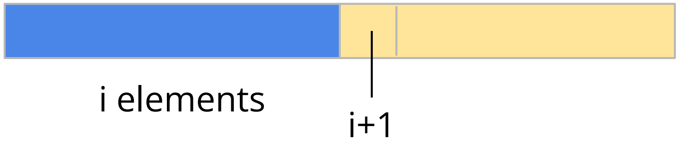
Insertion Sort

Best Case: $$\frac{n(S-1)}{S}$$

Worst Case: $$\frac{n(S-1)}{2}$$

c) Theoretical Analysis

Insertion Sort: Average Case



Number of comparisons in the ith iteration= $\frac{1}{i}(1+2+...+i)$

If size of subarray is S, there are (S-1) iterations

Number of comparisons = per subarray $1+\frac{1}{2}(1+2)+\frac{1}{3}(1+2+3)+...+\frac{1}{S-1}(1+...+S-1)=\frac{1}{4}(S-1)(S+2)$

Number of comparisons = for $\frac{n}{S}$ subarrays $\frac{n(S-1)(S+2)}{4S}=O(\frac{nS}{4})$

c) Theoretical Analysis

Hybrid Sort

Worst case = $nlog_2(\frac{n}{S}) + nS$

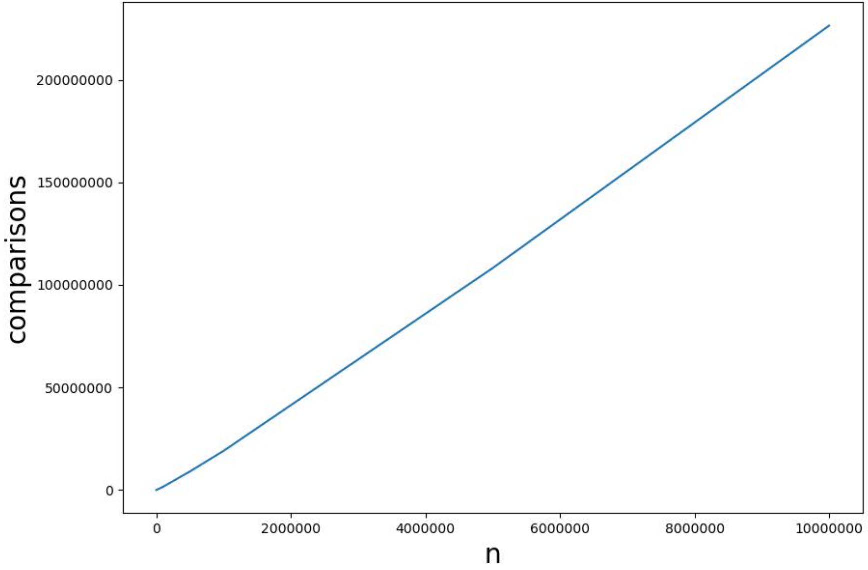
Average case = $nlog_2(\frac{n}{S}) + \frac{nS}{4}$

Best case = $nlog_2(\frac{n}{S}) + \frac{n}{S}(S-1)$

c i) Fix S, Vary n: comps over S

For randomly chosen S=10:

n	1,000	10,000	100,000	500,000	1,000,000	5,000,000	10,000,000
comps	9,117	126,761	1,558,302	9,036,737	19,071,221	108,215,228	226,414,009

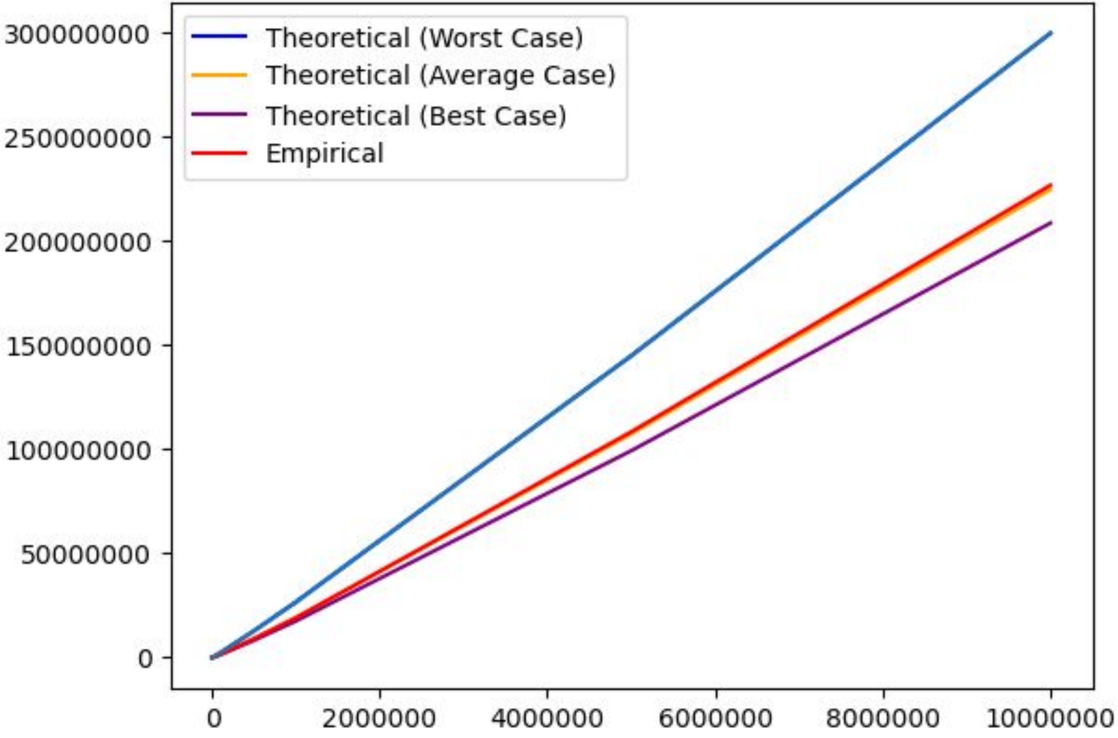


c i) Fix S, Vary n: Theoretical vs Empirical

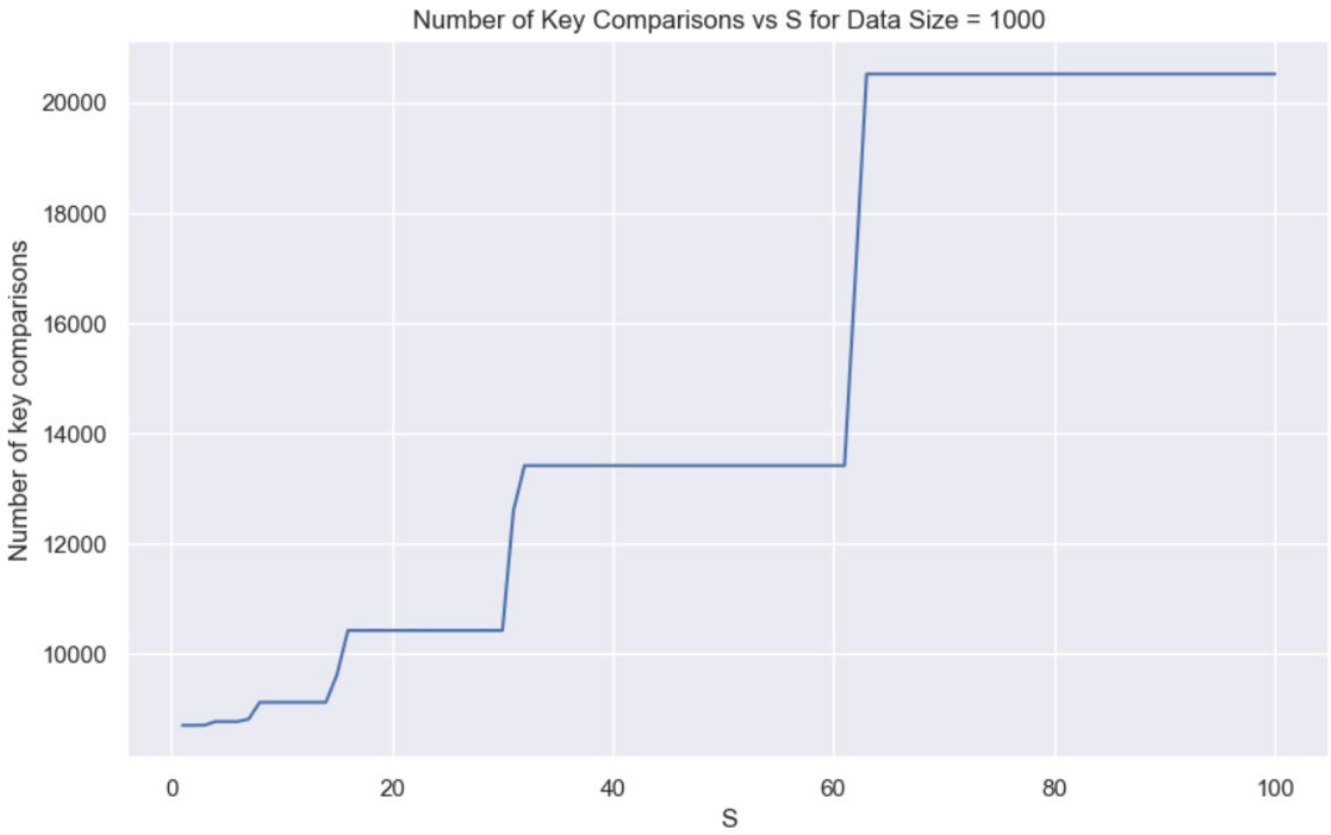
Worst Case: $nlog_2(\frac{n}{S}) + nS$

Average Case: $nlog_2(\frac{n}{S}) + \frac{nS}{4}$

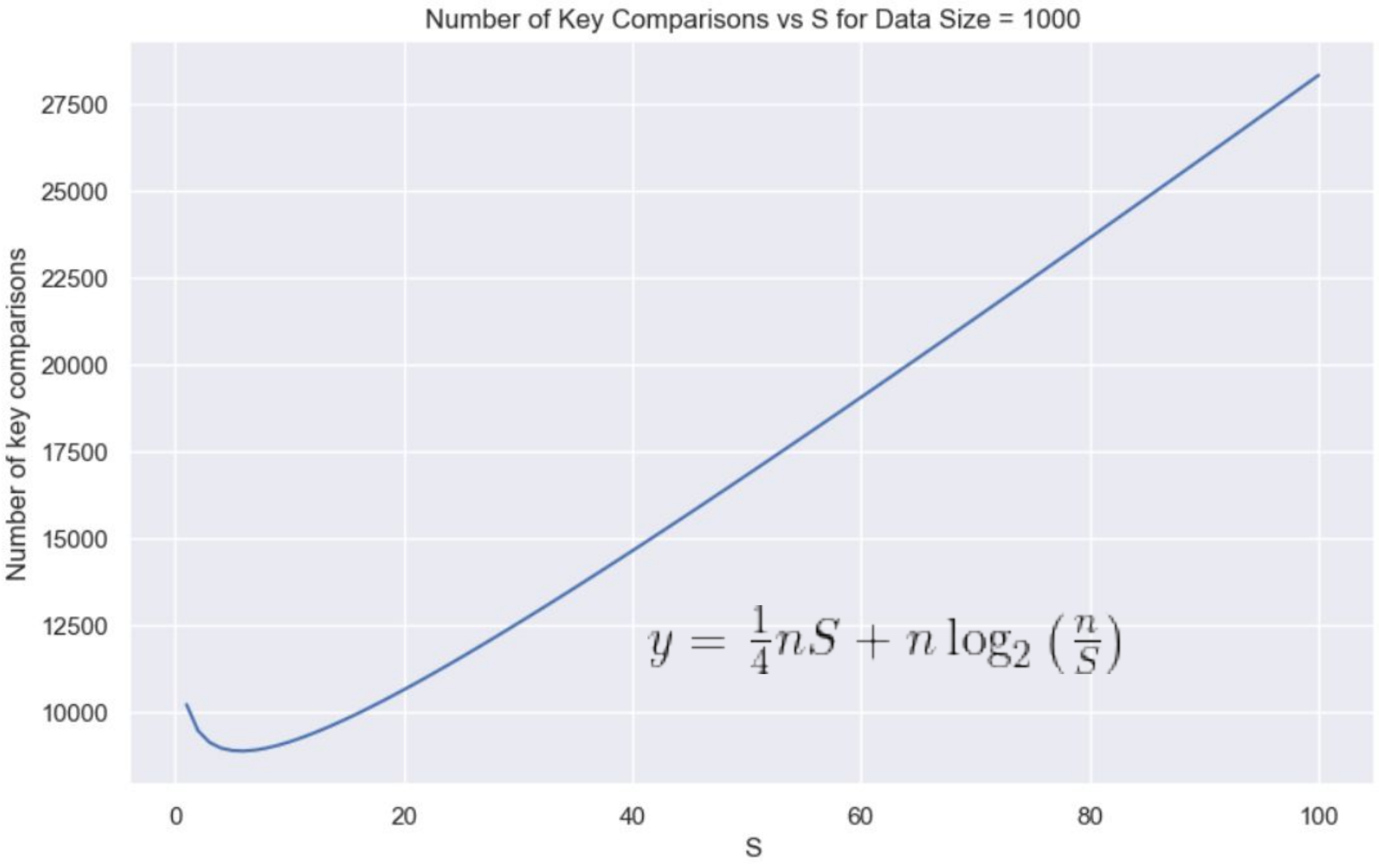
Best Case: $nlog_2(\frac{n}{S}) + \frac{n}{S}(S-1)$



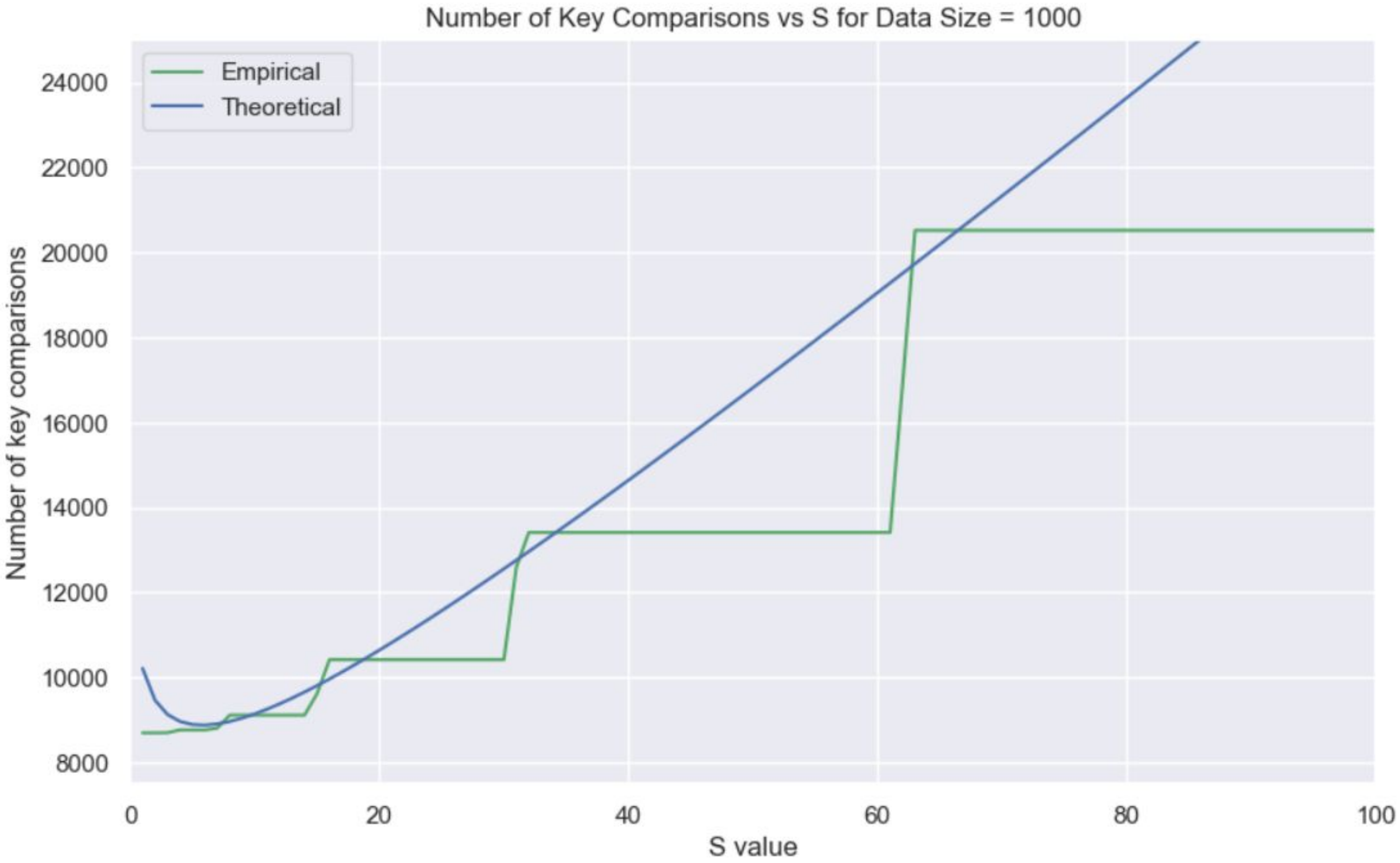
cii) Vary S, Fix n: Empirical



cii) Vary S, Fix n: Theoretical



cii) Vary S, Fix n: Theoretical vs Emp



cii) Vary S, Fix n: Theoretical vs Emp

1000

↓

500

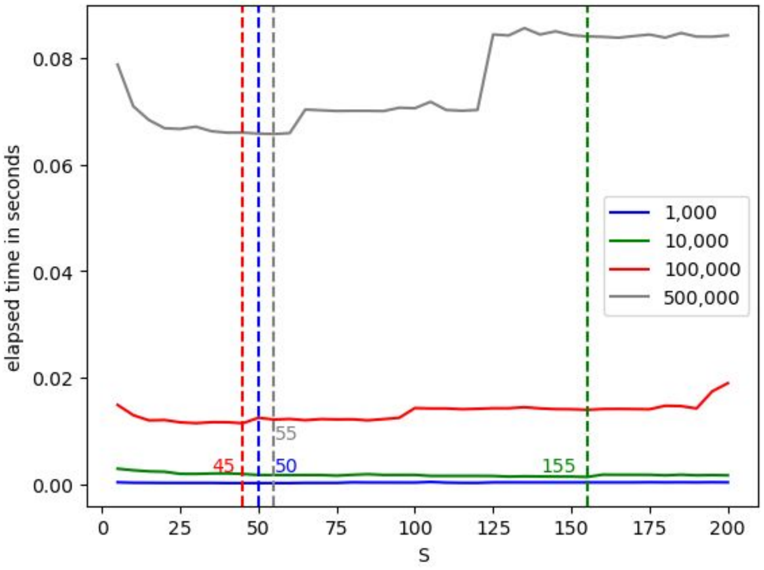
⋮

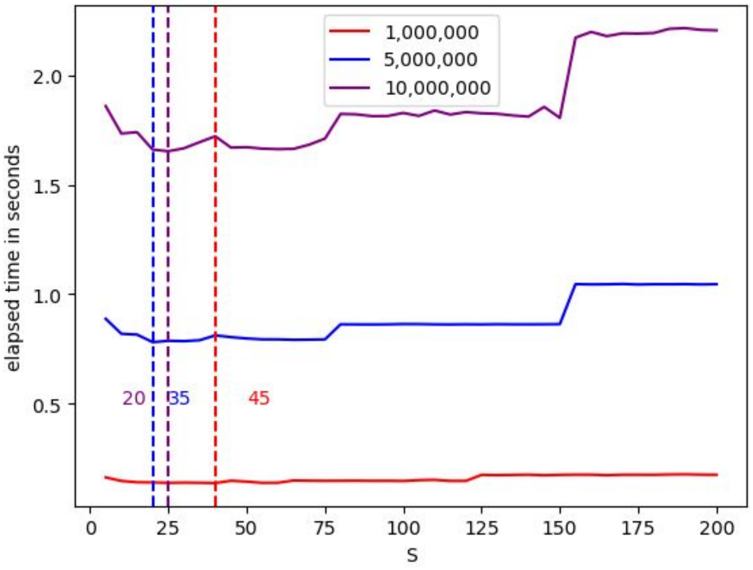
56 ⟶ When S= 56,57,58,...,111, subarray size is the same

↓

28 ⟶ When S= 28,29,...,55, subarray size is the same

c iii) Vary S, Vary n: What is the optimal S?





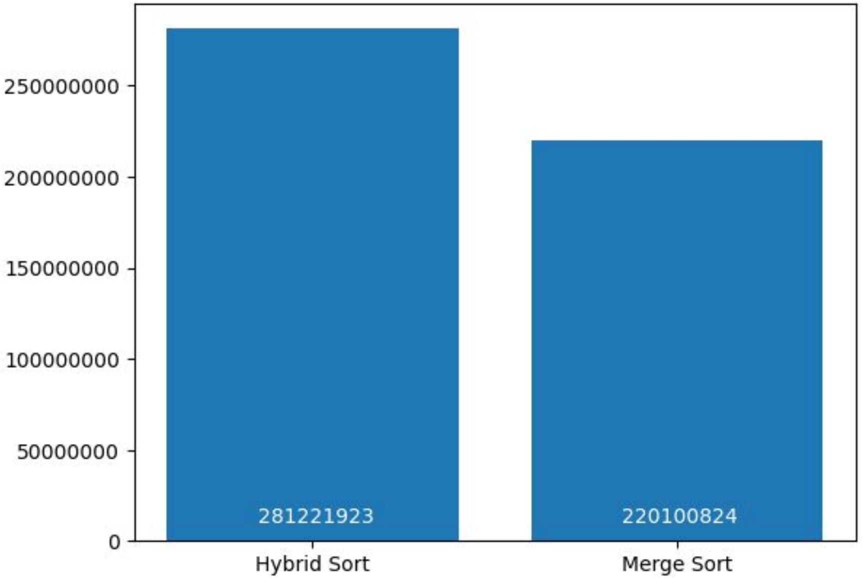
20, 35, 45, 45, 50, 55, 155

We will use the median S value of the 7 datasets as the optimal S

d) Hybrid Sort vs Merge Sort

```
clock_t start_time = clock();
hybridSort(0, index - 1, arr2, S);
clock_t end_time = clock();
double elapsed_time = (double)(end_time - start_time) /
CLOCKS_PER_SEC;
```

Key Comparisons



CPU Time