CZ2101

PROJECT 1: Analysis of Hybrid Sort

Agarwala Pratham
Caleb Cheam
Chen Xingyu

Table of contents

01 Hybrid Sort

02 Numerical Analysis

03 Time Analysis

04 Choosing optimal S

05 VS Merge Sort

06 Conclusion

01 Hybrid Sort

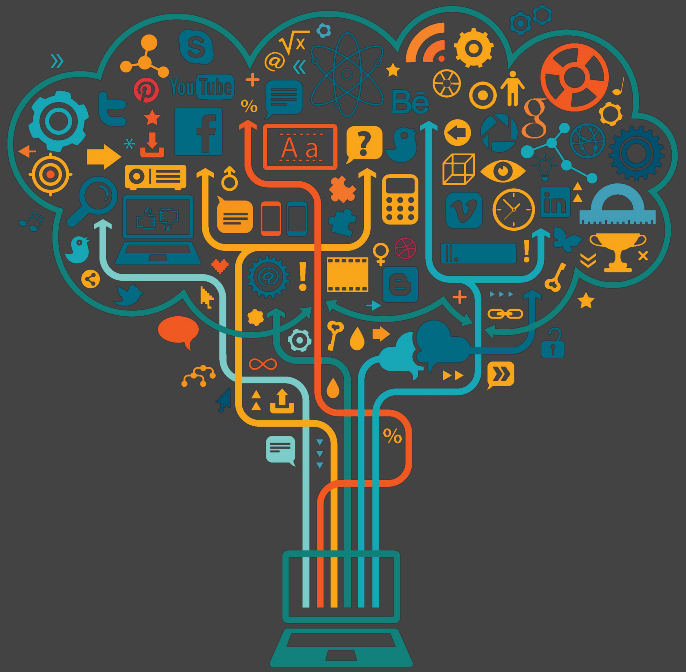
Parameters	Merge Sort	Insertion Sort
Worst Case Complexity	O(N*log N)	$O(N^2)$
Average Case Complexity	O(N*log N)	$O(N^2)$
Best Case Complexity	O(N*log N)	O(N)
Auxiliary Space Complexity	O(N)	O(1)
Works well on	On huge dataset.	On small dataset.
Efficiency	Comparitively Efficient.	Comparitively Inefficient.
Inplace Sorting	No	Yes
Algorithm Paradigm	Divide and Conquer	Incremental Approach
Uses	It is used for sorting linked list in O(N*log N), for Inversion Count problem, External sorting, etc.	It is used when number of elements is small. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

Algorithm

```
public void hybridSort(Integer[] list, int threshold)
{
    if (list.length <= threshold)
    {
        insertionSort(list);
        return;
    }
    else
    {
        int i, size = list.length, mid = size/2;
        Integer[] l = new Integer[mid];
        Integer[] r = new Integer[size - mid];
        for (i = 0; i < mid; i++)
            l[i] = list[i];
        for (i = mid; i < size; i++)
            r[i - mid] = list[i];
        hybridSort(l, threshold);
        hybridSort(r, threshold);

        merge(list, l, r);
    }
}
```

- Integration of merge sort and insertion sort
- Main skeleton is merge sort recursive algorithm
- Algorithm changes to insertion sort when sizes of subarrays are small



02 Theoretical Analysis

General Approach

```
public void hybridSort(Integer[] list, int threshold)
{
    if (list.length <= threshold)
    {
        insertionSort(list);
        return;
    }
    else
    {
        ...

        hybridSort(l, threshold);
        hybridSort(r, threshold);

        merge(list, l, r);
    }
}
```

Base case: W(s)

W(n/2)

W(n/2)

$$W(n) = 2W\left(\frac{n}{2}\right) + merge$$

$$base\ case: W(s)\ of\ insertion\ sort$$

Different input cases

worst case

Assumption: n and s are powers of 2

$$W(n) = 2W\left(\frac{n}{2}\right) + (n-1)$$

Worst case of *Merge*

$$W(n) = 2^k W\left(\frac{n}{2^k}\right) + kn - (1 + 2 + 4 + \cdots + 2^{k-1})$$

Base case $W(s) = \frac{s(s-1)}{2}$ Worst case of *Insertion Sort*

Hence

$$\frac{n}{2^k} = s$$

$$2^k = \frac{n}{s}$$

$$k = \log_2\left(\frac{n}{s}\right)$$

Sub. k $W(n) = \frac{n}{s} \cdot \frac{s(s-1)}{2} + n\log_2\left(\frac{n}{s}\right) - \left(\frac{n}{s} - 1\right)$

$$\in O\left(ns + n\log_2\left(\frac{n}{s}\right)\right)$$

Different input cases

worst case: finding optimal s

To find the threshold (s) that minimize key comparisons for a **fixed input size (n)**,
s is either the value that makes partial derivative w.r.t threshold(s) 0
or s is at the boundaries (s=1 and s=n)

W(n) simplified to $W(n) = \frac{ns}{2} - \frac{n}{2} + n\log_2 n - n\log_2 s - \frac{n}{s} + 1$

$$\frac{\partial W(n)}{\partial s} = \frac{n}{2} - \frac{n}{\ln 2} \cdot \frac{1}{s} + \frac{n}{s^2} = 0$$

solutions $s \approx 1.73 \approx 2$ $s \approx 1.16 \approx 1$

boundaries $s = 1 \; and \; s = n$

Optimal s: among solutions to the equation and s=1 and s=n, choose the s that minimizes overall key comparisons

optimal $s = 1 \; or \; 2$

Different input cases

best case

Assumption: n and s are powers of 2

$$W(n) = 2W\left(\frac{n}{2}\right) + \frac{n}{2}$$

Best case of *Merge*

$$W(n) = 2^k W\left(\frac{n}{2^k}\right) + \frac{kn}{2}$$

Base case $W(s) = s - 1$ Best case of *Insertion Sort*

Hence

$$\frac{n}{2^k} = s$$

$$2^k = \frac{n}{s}$$

$$k = \log_2\left(\frac{n}{s}\right)$$

Sub. k $W(n) = \frac{n}{s} \cdot (s-1) + \frac{n}{2}\log_2\left(\frac{n}{s}\right)$

$$\in O\left(n + n\log_2\left(\frac{n}{s}\right)\right)$$

Different input cases

best case: finding optimal s

Optimal s: among solutions to the equation and s=1 and s=n, choose the s that minimizes overall key comparisons

W(n) simplified to $W(n) = n\frac{s-1}{s} + \frac{n}{2}\log_2 n - \frac{n}{2}\log_2 s$

$$\frac{\partial W(n)}{\partial s} = \frac{n}{s^2} - \frac{n}{2\ln 2}\cdot\frac{1}{s} = 0$$

solution $s = 2\ln 2 \approx 1$

boundaries $s = 1 \; and \; s = n$

optimal $s = n$

Different input cases

average case

Assumption: every number of key comparisons in *Merge* (from n/2 to n-1) is equally likely

There are $(n-1)-\frac{n}{2}+1=\frac{n}{2}$ terms

So each with probability $\frac{1}{n/2}=\frac{2}{n}$

$$E(comparisons)=\frac{2}{n}\times\left(\frac{n}{2}+n-1\right)\times\frac{n}{2}\times\frac{1}{2}=\frac{3n-2}{4}$$

Different input cases

average case

Assumption: n and s are powers of 2

$$W(n) = 2W\left(\frac{n}{2}\right) + \frac{3n-2}{4}$$

Average case of *Merge*

$$W(n) = 2^k W\left(\frac{n}{2^k}\right) + \frac{3nk - 2^{k+1} + 2}{2}$$

Base case $W(s) = \frac{s^2 + s - 2}{4}$

Average case of *Insertion Sort*

Hence

$$\frac{n}{2^k} = s$$

$$2^k = \frac{n}{s}$$

$$k = \log_2\left(\frac{n}{s}\right)$$

Sub. k $W(n) = \frac{n}{s} \cdot \frac{(s+2)(s-1)}{4} + \frac{3}{4} n \log_2\left(\frac{n}{s}\right) - \frac{n}{2s} + \frac{1}{2}$

$$\in O\left(ns + n\log_2\left(\frac{n}{s}\right)\right)$$

Different input cases

average case: finding optimal s

- Optimal s: among solutions to the equation and s=1 and s=n, choose the s that minimizes overall key comparisons

W(n) simplified to $$W(n) = \frac{n}{4} \cdot \frac{s^2 + s - 2}{s} + \frac{3}{4} n \log_2 n - \frac{3}{4} n \log_2 s - \frac{n}{2s} + \frac{1}{2}$$

$$\frac{\partial W(n)}{\partial s} = \frac{n}{4} - \frac{3n}{4 \ln 2} \cdot \frac{1}{s} + \frac{n}{s^2} = 0$$

solution $s \approx 2.99 \approx 3$ $s \approx 1.34 \approx 1$

boundaries $s = 1 \; and \; s = n$

optimal $s = 3$

Different input cases

Best case

$$\in O\left(n + n\log_2\left(\frac{n}{s}\right)\right)$$

Average case

$$\in O\left(ns + n\log_2\left(\frac{n}{s}\right)\right)$$

Worst case

$$\in O\left(ns + n\log_2\left(\frac{n}{s}\right)\right)$$

Findings

S- threshold
N- size of input

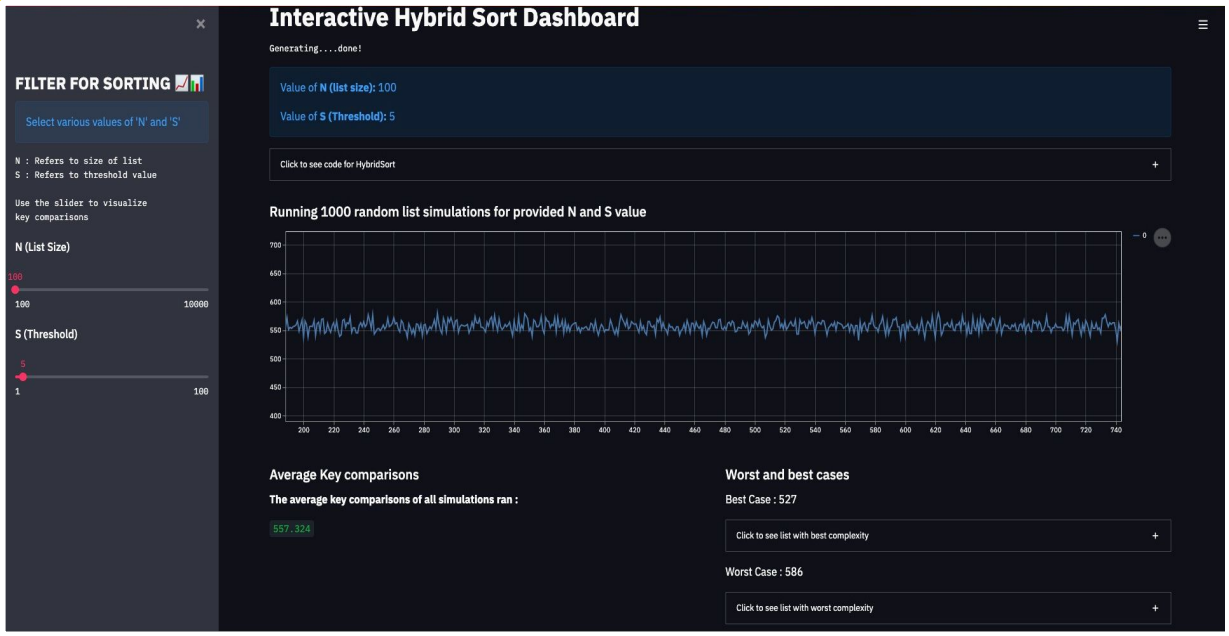
- The optimal value of S is dependent on the input cases
- Optimal S is N(=>insertion sort) in the best case scenario (list already sorted)
- For the average case and worst case, result suggests that optimal S should be 1 2 or 3
- Assumption of every possible number of comparisons being equally likely in average case may not be accurate
- We will investigate small S values in our empirical analysis since equations show that S should be a small value even when N is large

03
Empirical Analysis

Approach

Plotted key comparisons against varying values of S and N

We made a dashboard displaying the different plots we did while investigating



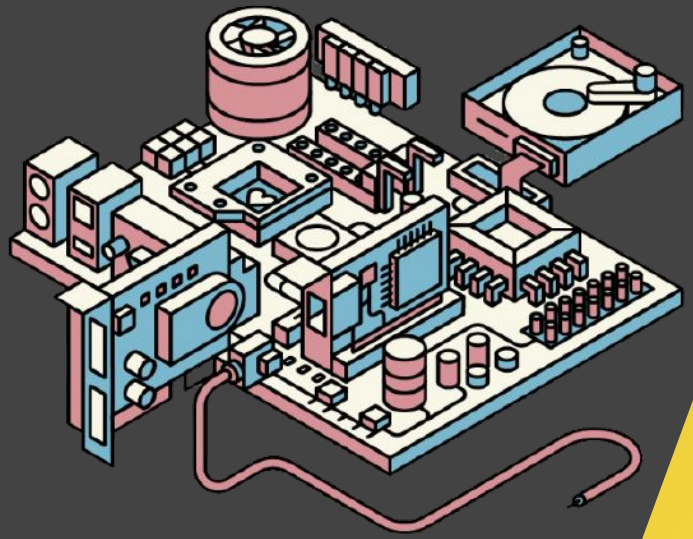
http://localhost:8501/

Findings

S- threshold
N- size of input

- There is greater variance in key comparisons as S increases
- S at higher values results in more key comparisons
- At theoretically optimal s(=3), key comparison is not minimized
- Hypothesized that key comparisons may not be the best way to choose optimal value of S since there may be other factors involved
- We will take another approach of choosing the optimal S value, by changing our metric of evaluating time complexity

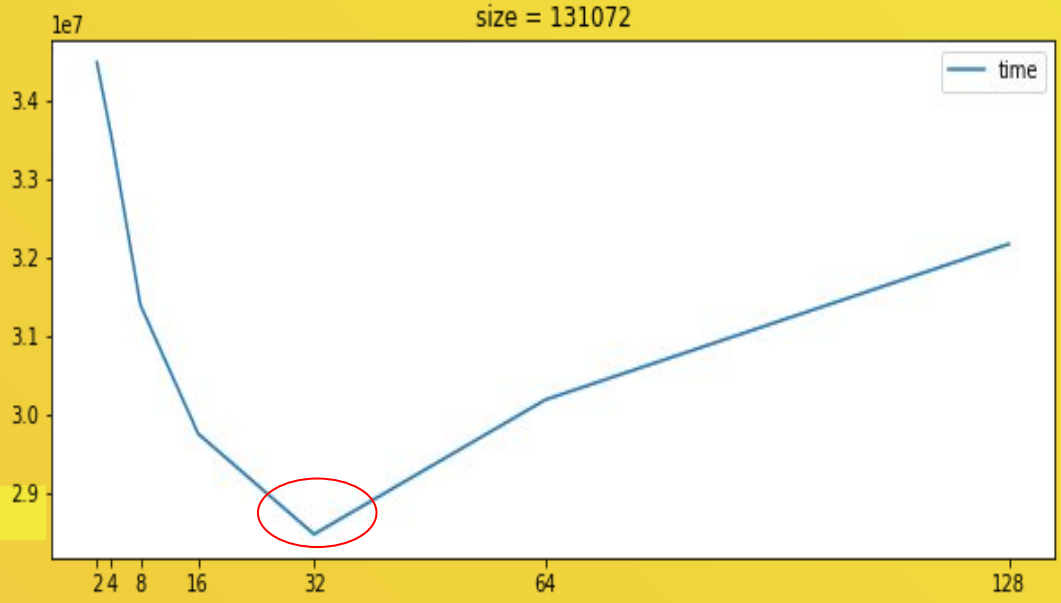
04
Choosing optimal S



Approach

We decided to choose our optimal S value based on the actual CPU run time of the algorithm, with an input size of N = powers of 2

We tried out different input sizes for ascending /descending/random lists. For each input size, we plotted a graph of different s vs CPU time


size = 131072
1e7
time
3.4
3.3
3.2
3.1
3.0
2.9
2
4
8
16
32
64
128

Optimal S values

N	Descending	Random	Ascending
2^7	8	16	N
2^10	16	64	N
2^12	16	64	N
2^14	4	32	N
2^16	16	32	N
2^17	8	16	N
2^19	16	32	N
2^20	16	32	N

Findings

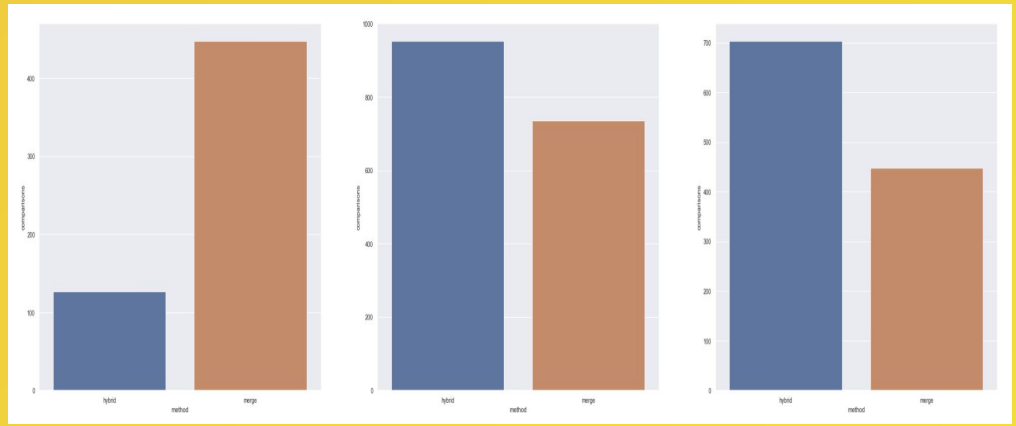
S- threshold
N- size of input

- Optimal S value is still dependent on input cases
- Previously, when using key comparison to choose optimal S, value of S was usually 2-4
- Upon changing our metric to CPU runtime, we have a conclusive optimal S value where time complexity converges, for different N values

05
Comparison with Merge Sort

N is small(n=128)

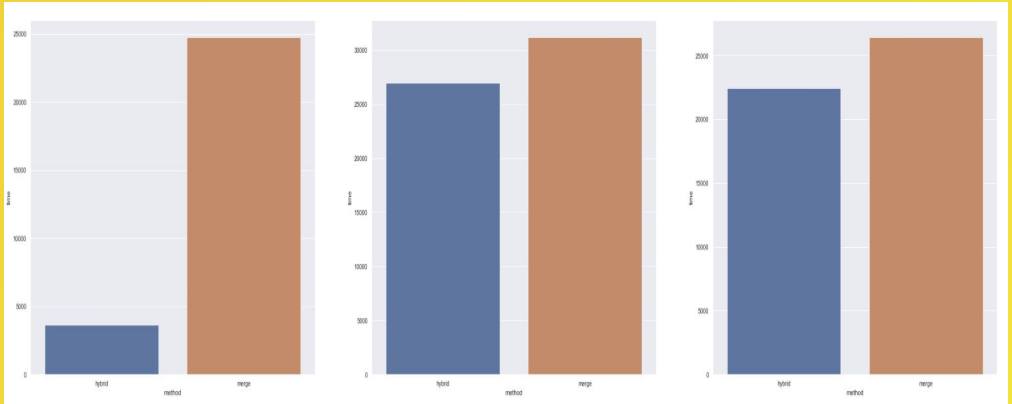
Metric
Key
Comparisons



HYBRID

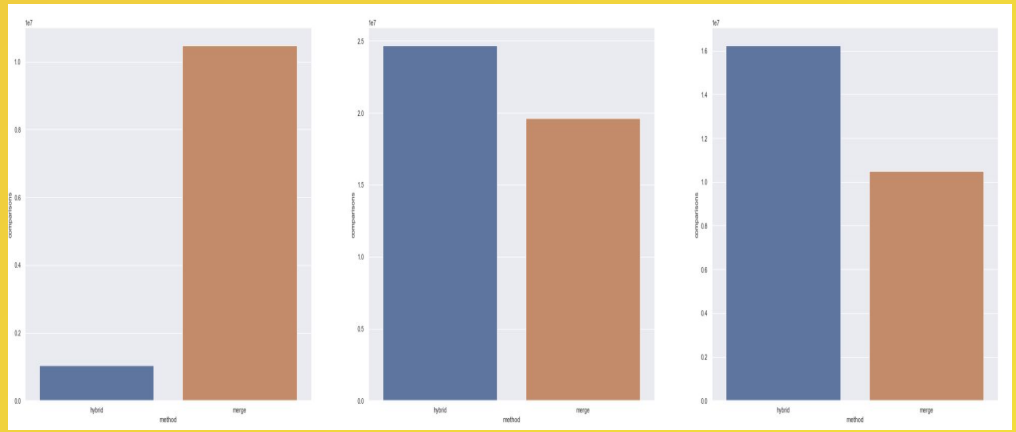
MERGE

Metric
Cpu Time



N is large(n=2^20)

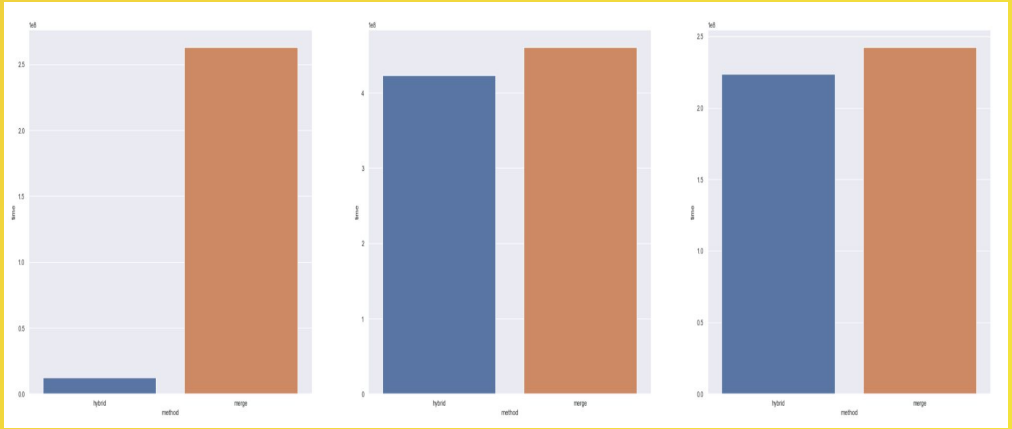
Metric
Key
Comparisons



HYBRID

MERGE

Metric
Cpu Time



Findings

- Hybrid Sort is 8.26% quicker than conventional Merge Sort algorithm for large input sizes
- Hybrid Sort is 13.66% quicker than conventional Merge Sort algorithm for small input sizes
- We see that key comparisons and CPU runtime are not proportional
- This may be due to Merge Sort having more recursive calls
- Despite Insertion Sort having more key comparisons in general, it is a fast algorithm for smaller sizes due to cache efficiency

06

Conclusion

Conclusion

Using numerical analysis and different metrics of time complexity, we were able to derive with an optimal S value for different input cases and different sizes

However, in reality, there is no “correct” S value since memory cache, background processes, and operating systems differ from machine to machine

Our findings support our hypothesis that hybrid sort should perform better than merge sort if we optimise S