Program Structures and Algorithms

Spring 2024

NAME: Siddharth Dumbre
NUID: 002247119
GITHUB LINK: https://github.com/dumbresi/Info-6205-spring2024

Task: Hits as time predictor

Observations:

	A	B	C	D	E	F	G	H
1	Array length	Sorting Algori	Hits	Copies	Swaps	Compares	Runtime	Normalized runtim
2								
3	10000	Merge sort	269,794	110,000	9,764	121,509	4.33	6.10
4	20000	Merge sort	579,551	240,000	19,514	263,004	8.27	5.36
5	40000	Merge sort	1,239,112	520,000	39,034	566,010	18.06	5.85
6	80000	Merge sort	2,638,374	1,120,000	78,111	1,212,047	43.1	6.05
7	160000	Merge sort	5,596,625	2,400,000	156,181	2,584,091	95.35	7.86
8	10000	Quick Sort	424,100	0	66,458	156,487	4.36	6.14
9	20000	Quick Sort	914,320	0	142,098	340,834	9.2	5.97
10	40000	Quick Sort	1,951,782	0	298,533	741,806	19.6	5.90
11	80000	Quick Sort	4,222,065	0	653,087	1,587,877	42	5.85
12	160000	Quick Sort	8,909,004	0	1,356,021	3,416,383	86.48	5.73
13	10000	Heap sort	967,589	0	124,209	235,377	4.88	6.87
14	20000	heap sort	2,095,073	0	268,398	510,740	10.88	7.06
15	40000	Heap sort	4,510,222	0	576,806	1,101,499	24	7.08
16	80000	Heap sort	9,660,262	0	1,233,593	2,362,946	56.52	7.93
17	160000	Heap sort	20,600,874	0	2,627,231	5,045,976	119.59	7.86
18								

Comparison:
- In most cases, Merge Sort exhibits the highest number of comparisons, followed by Quick Sort and then Heap Sort. However, as the number of elements increases, the disparity in comparisons between Merge Sort and Quick Sort diminishes.

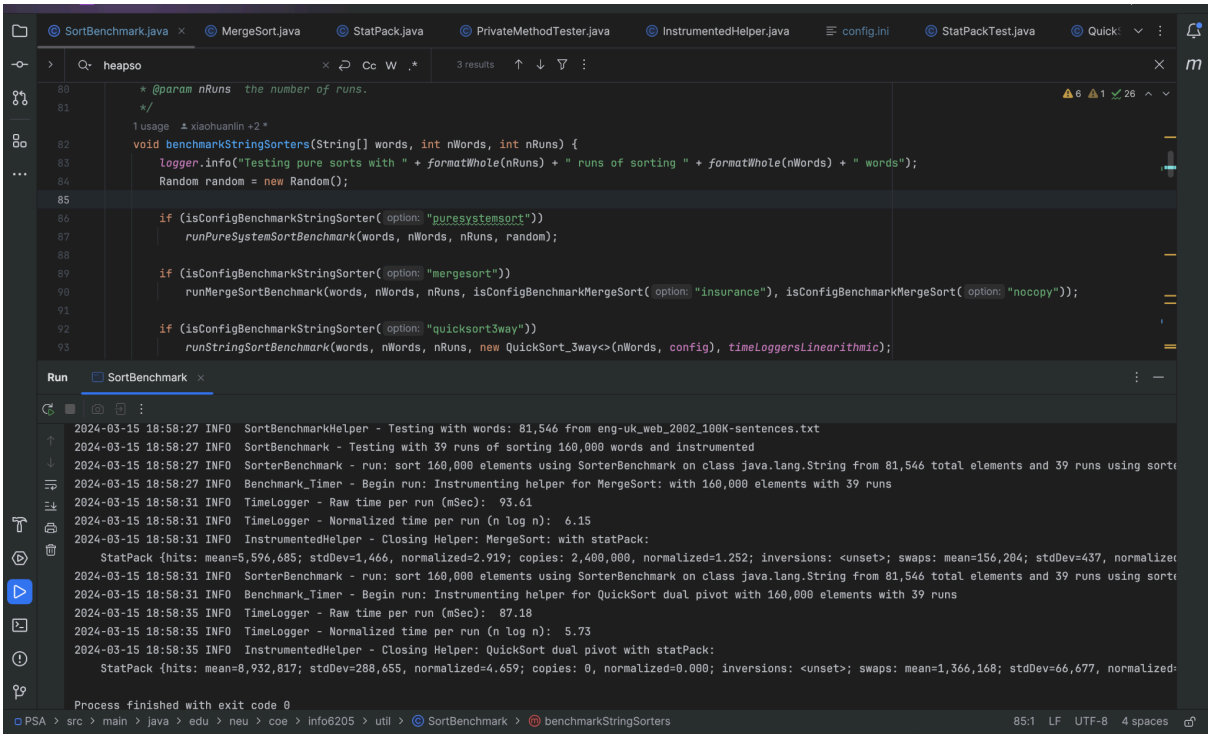
Swapping:
- Among the three algorithms, Heap Sort necessitates the greatest number of swaps, succeeded by Quick Sort and then Merge Sort. With an increasing number of elements, the distinction in swaps between Heap Sort and Quick Sort becomes more pronounced.

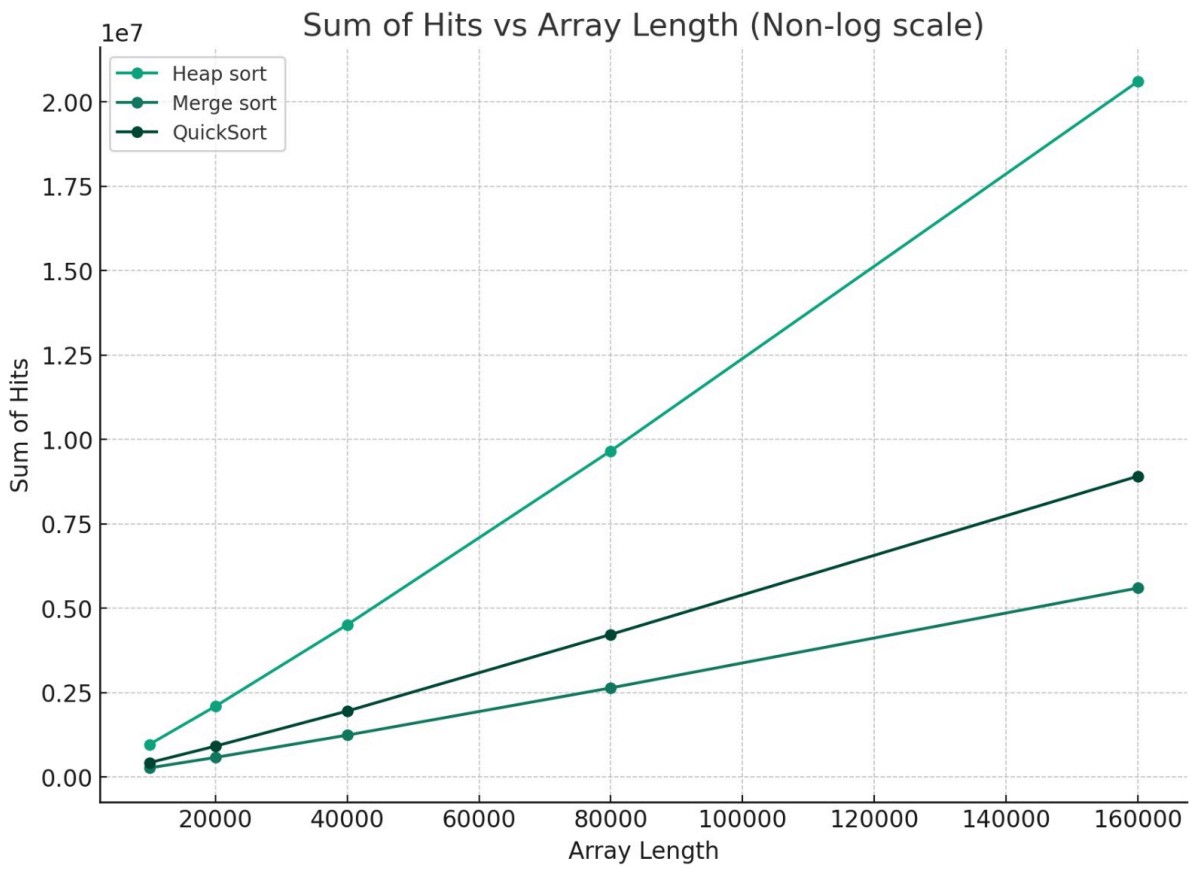
Hits:
- Solely Merge Sort employs the concept of "hits" or merges to combine the two subarrays during the merge phase. The frequency of hits escalates with the number of elements and correlates directly with the number of merges required. In contrast, Heap Sort and Quick Sort do not utilize hits.

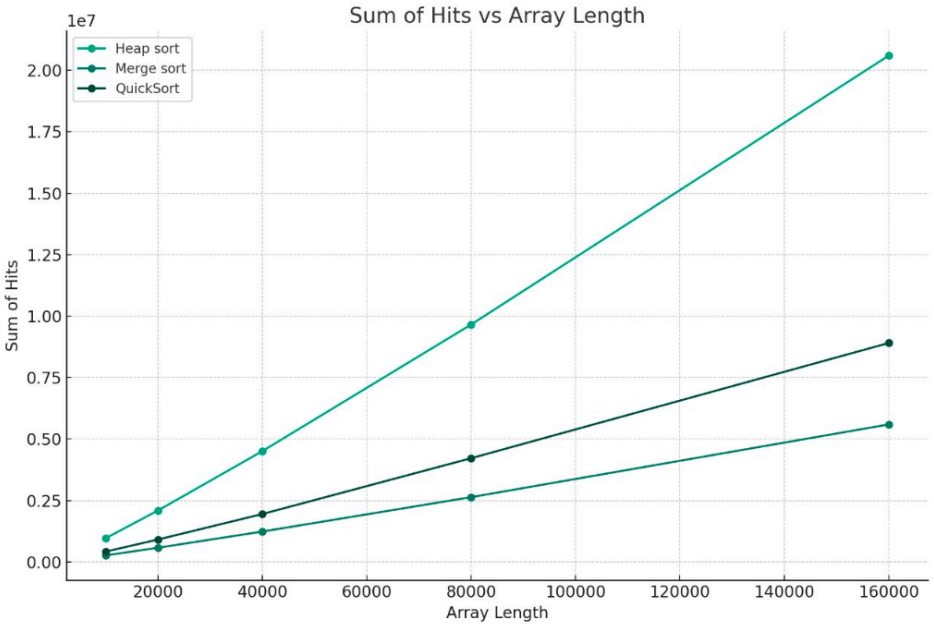
Copying:
- Merge Sort and Heap Sort exclusively incorporate the notion of copies to retain temporary arrays throughout the sorting procedure. Merge Sort entails a higher count of copies compared to Heap Sort.

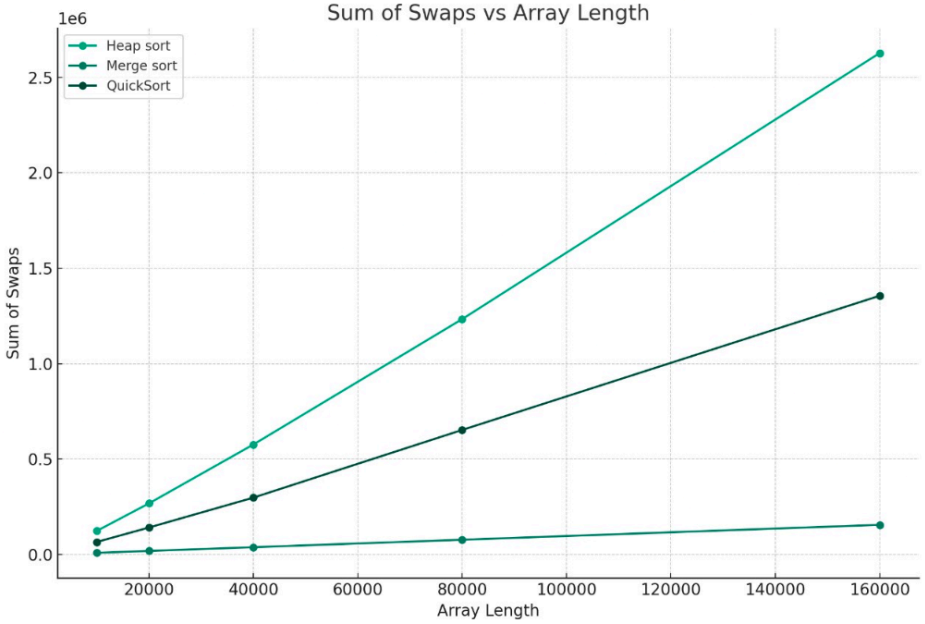
In case of heap sort, the number of hits drastically increases as the array length increases. For quick sort the increase is relatively lower and for the merge sort it is the lowest. Hence we can conclude that for higher array lengths, merge sort is better as it has lower number of hits and has lower execution time.

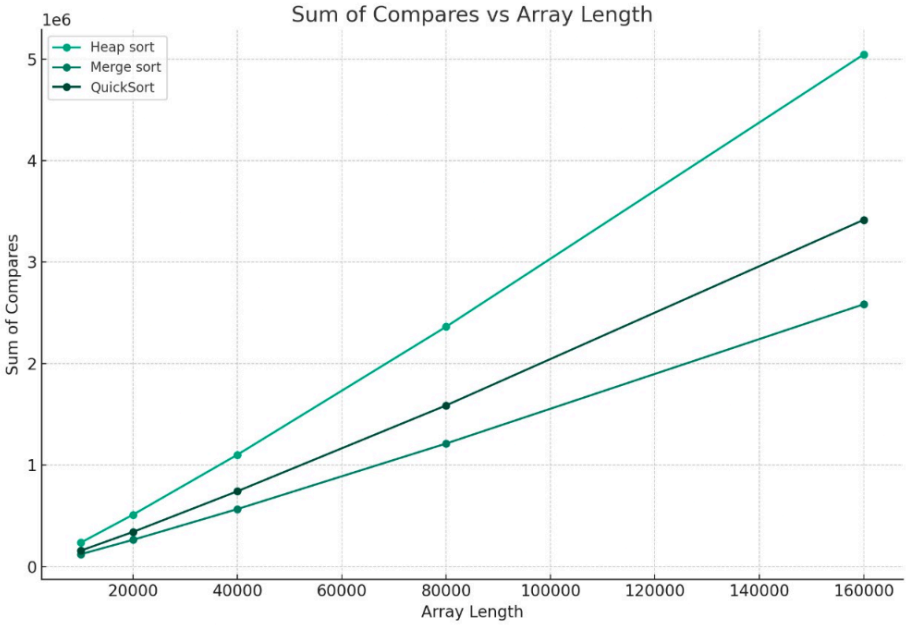
Supporting evidence and screenshoots:

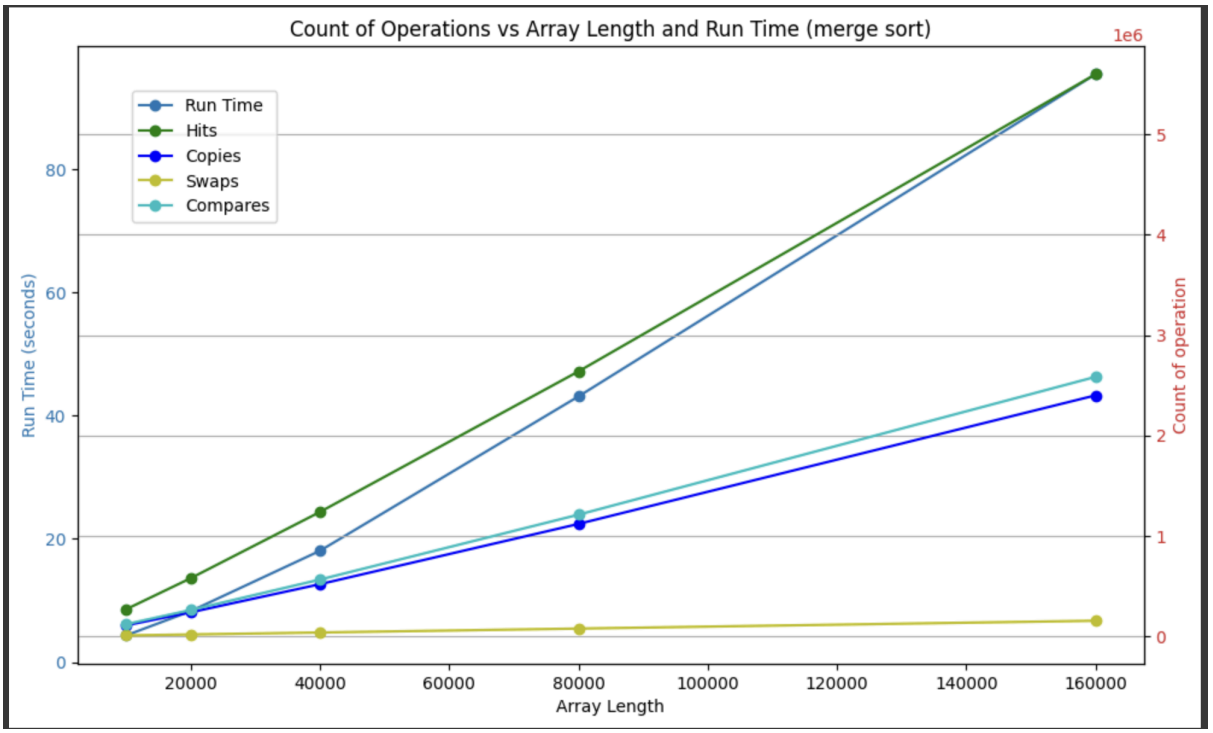


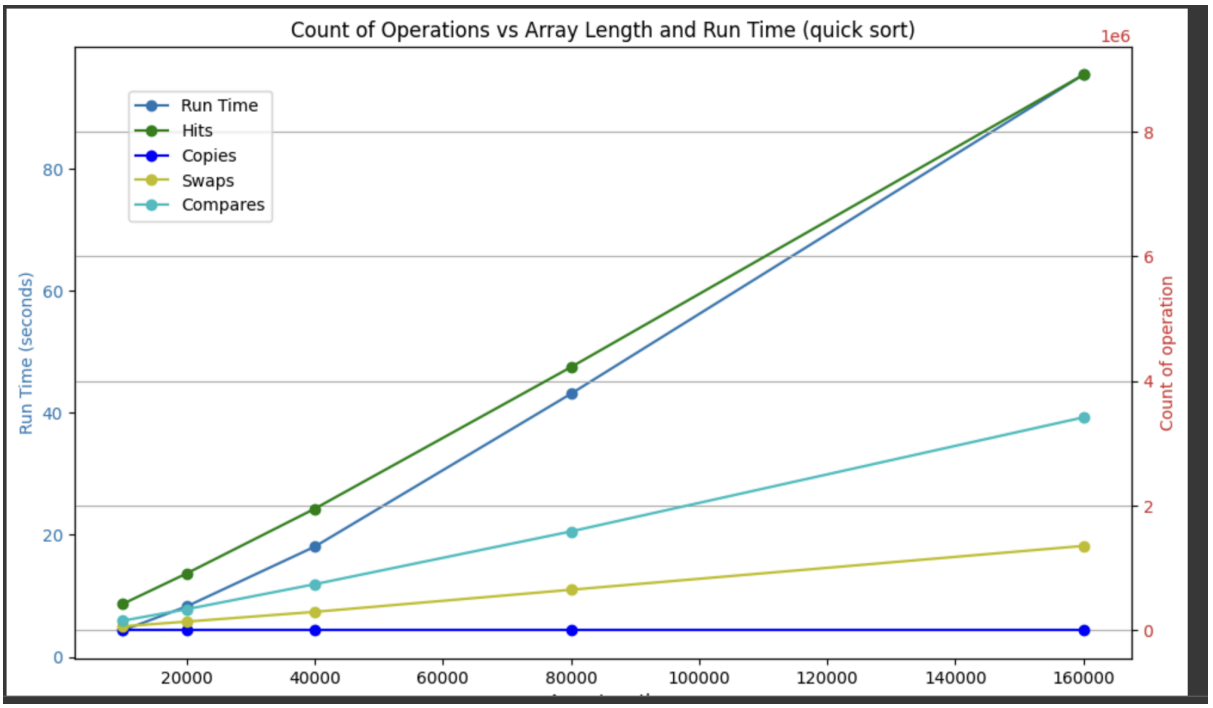


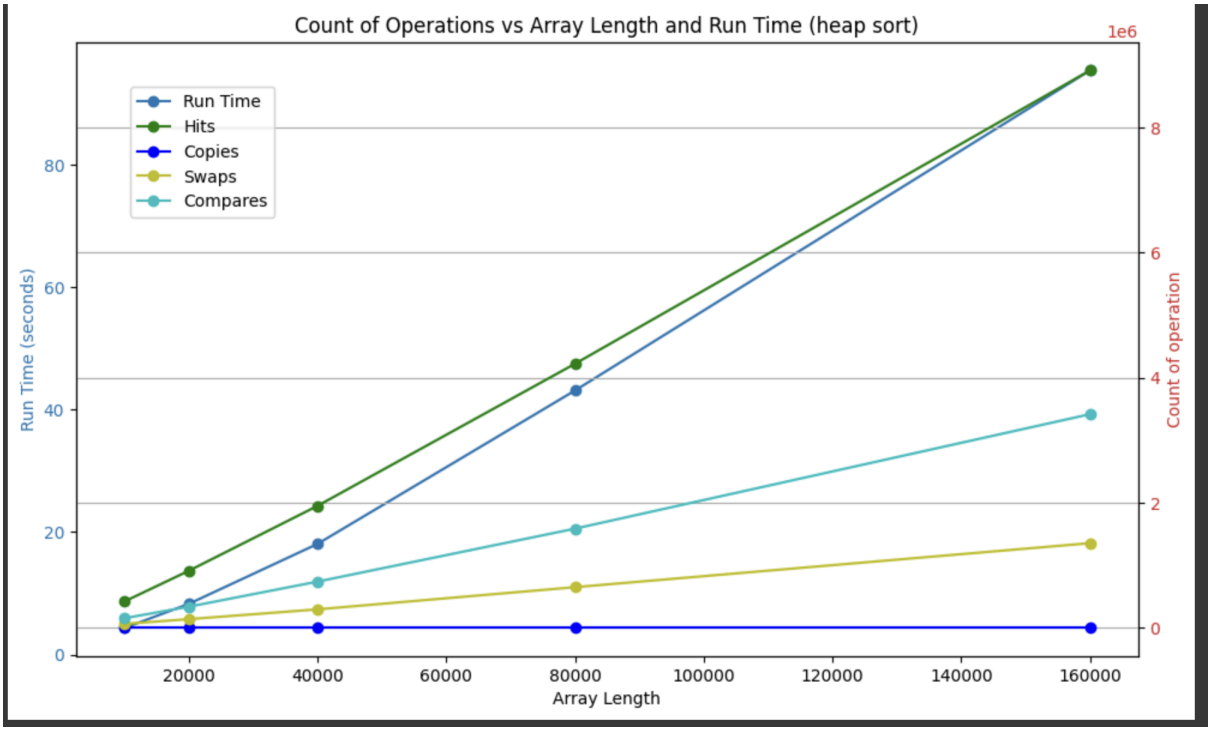

Sum of Hits vs Array Length
1e7
Heap sort
Merge sort
QuickSort
Sum of Hits
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
20000
40000
60000
80000
100000
120000
140000
160000
Array Length


Sum of Swaps vs Array Length
1e6
Heap sort
Merge sort
QuickSort
Sum of Swaps
2.5
2.0
1.5
1.0
0.5
0.0
20000
40000
60000
80000
100000
120000
140000
160000
Array Length


Sum of Compares vs Array Length
1e6
Heap sort
Merge sort
QuickSort
Sum of Compares
5
4
3
2
1
0
20000
40000
60000
80000
100000
120000
140000
160000
Array Length


Count of Operations vs Array Length and Run Time (merge sort)
1e6
Run Time
Hits
Copies
Swaps
Compares
Run Time (seconds)
Count of operation
Array Length
0
20
40
60
80
20000
40000
60000
80000
100000
120000
140000
160000
0
1
2
3
4
5


Count of Operations vs Array Length and Run Time (quick sort)
1e6
Run Time
Hits
Copies
Swaps
Compares
Run Time (seconds)
Count of operation
0
20
40
60
80
20000
40000
60000
80000
100000
120000
140000
160000
0
2
4
6
8



Conclusion:

As we can see that for all the sorting algorithms, as the number of hits, swaps, comparisons increase, the number of time increases. However, there is a very strong co relation between the time taken and the number of hits. As seen from the log log plots for different array lengths it can be seen that the number of hits can be considered as the best predictor for total execution time.