KING OF SORT

The Problem

Standard C library provides qsort function that can be used for sorting , qsort function uses Quick Sort algorithm to sort . C might be the fastest language but qsort is very slow. Quick sort is not a stable algorithm.

The Solution

Hybrid Sort

Step 1

Insertion sort efficient for small data sets and provides stability but for large data sets it is inefficient

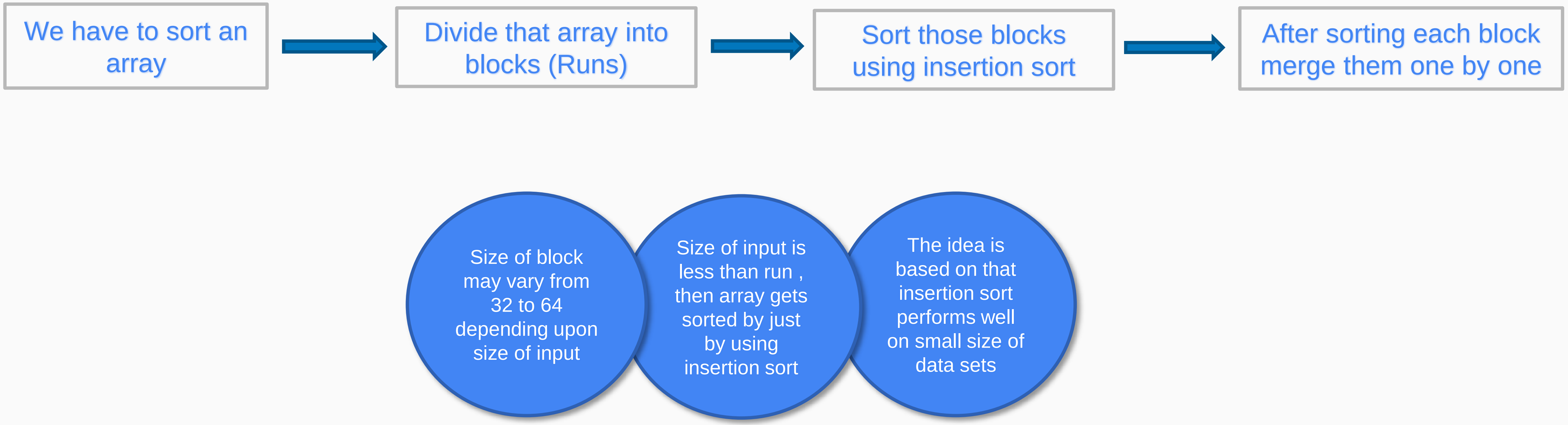
Step 2

Merge Sort is not in place and it also provides stability but slow for small size of arrays and takes O(nlogn) time for already sorted data

Step 3

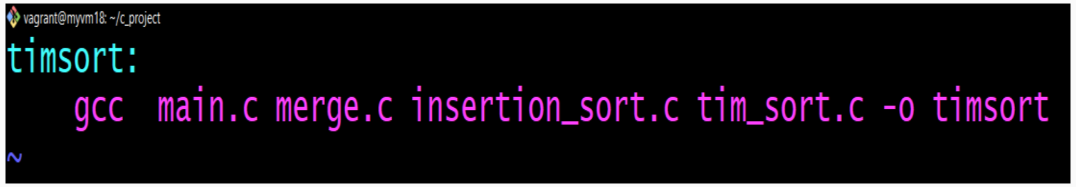
Tim Sort is a hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data

How it works (Architecture Diagram)


We have to sort an array
Divide that array into blocks (Runs)
Sort those blocks using insertion sort
After sorting each block merge them one by one
Size of block may vary from 32 to 64 depending upon size of input
Size of input is less than run , then array gets sorted by just by using insertion sort
The idea is based on that insertion sort performs well on small size of data sets

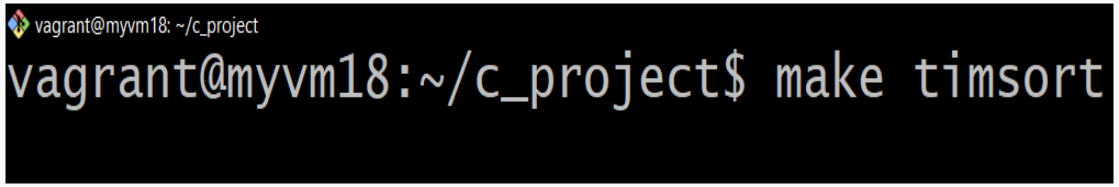
Demo

Makefile

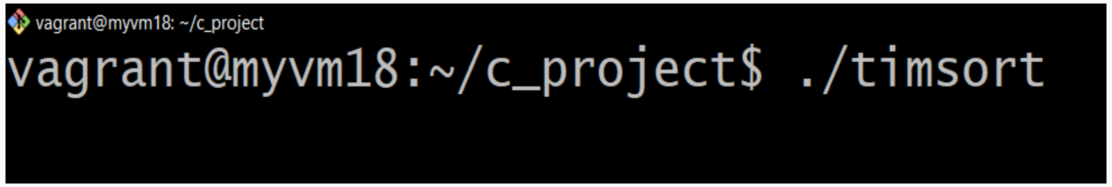


```
vagrant@myvm18: ~/c_project
timsort:
	gcc  main.c merge.c insertion_sort.c tim_sort.c -o timsort
~
```

To execute Makefile

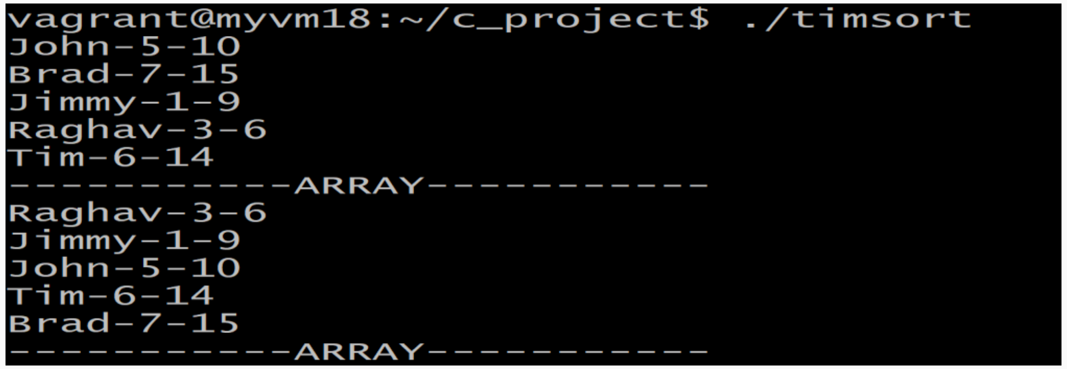


```
vagrant@myvm18: ~/c_project
vagrant@myvm18:~/c_project$ make timsort
```



```
vagrant@myvm18: ~/c_project
vagrant@myvm18:~/c_project$ ./timsort
```

To execute code



```
vagrant@myvm18:~/c_project$ ./timsort
John-5-10
Brad-7-15
Jimmy-1-9
Raghav-3-6
Tim-6-14
-----------ARRAY-----------
Raghav-3-6
Jimmy-1-9
John-5-10
Tim-6-14
Brad-7-15
-----------ARRAY-----------
```

Output

Drawbacks

1. It efficiency is less for random data.
2. It is not Inplace, merge functions uses extra space
3. It can become more efficient by using binary insertion sort .

Quad Sort is better than Tim Sort it performs better on both random data as well as on ordered data