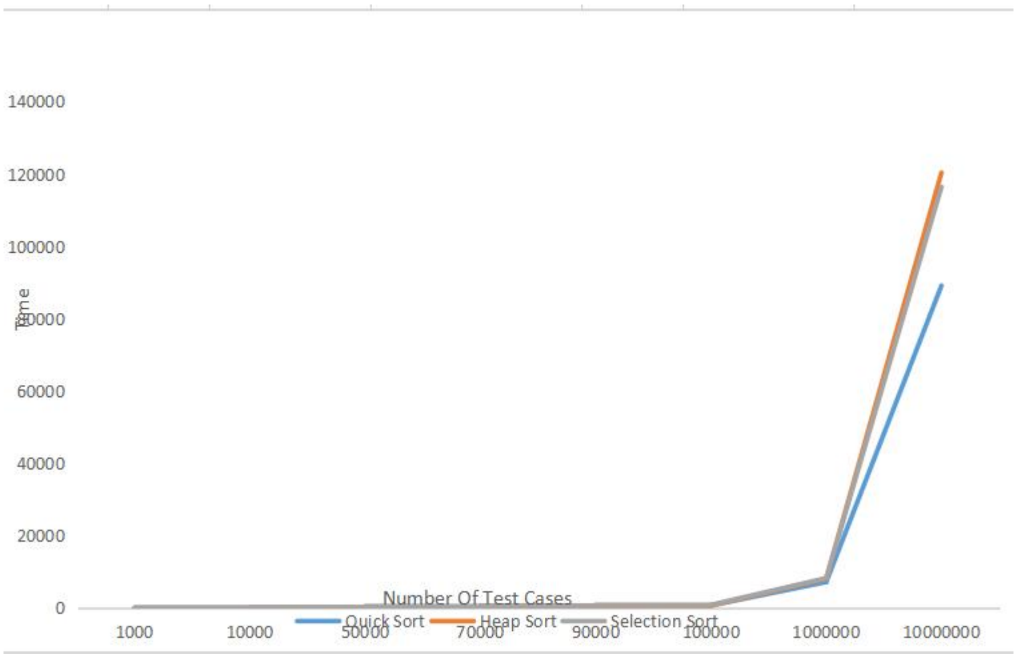
| Quick sort | | Heap Sort | | Selection Sorting | |
|---|---|---|---|---|---|
| No. of data | Time taken(ms) | No. of Data | Time Taken(ms) | No. of Data | Time Taken (ms) |
| 1000 | 36 | 1000 | 13 | 1000 | 34 |
| 10000 | 87 | 10000 | 67 | 10000 | 124 |
| 50000 | 377 | 50000 | 330 | 50000 | 373 |
| 70000 | 515 | 70000 | 491 | 70000 | 489 |
| 90000 | 683 | 90000 | 639 | 90000 | 830 |
| 100000 | 675 | 100000 | 657 | 100000 | 829 |
| 1000000 | 7153 | 1000000 | 8071 | 1000000 | 8234 |
| 10000000 | 89138 | 10000000 | 120371 | 10000000 | 116427 |



CONCLUSION-

Merge sort uses the divide and conquer approach. It is one of the most efficient algorithm for sorting. In this algorithm, we divide the list into two from the middle and keep dividing the list until the sub-problem has only one element list that's why it takes less time as compared to others.

Thus if we are given a choice on a large no of Data then we will simply use merge sort to reduce the Execution Time and No of Comparison.

```c
#include <sys/time.h>
#include <stdio.h>
#include<stdlib.h>
void merge(int arr[], int l, int m, int r)
{
	int i, j, k;
	int n1 = m - l + 1;
	int n2 = r - m;

	/* create temp arrays */
	int L[n1], R[n2];

	/* Copy data to temp arrays L[] and R[] */
	for (i = 0; i < n1; i++)
		L[i] = arr[l + i];
	for (j = 0; j < n2; j++)
		R[j] = arr[m + 1 + j];

	i = 0; // Initial index of first subarray
	j = 0; // Initial index of second subarray
	k = l; // Initial index of merged subarray
	while (i < n1 && j < n2) {
		if (L[i] <= R[j]) {
			arr[k] = L[i];
			i++;
		}
		else {
			arr[k] = R[j];
			j++;
		}
		k++;
	}

	while (i < n1) {
		arr[k] = L[i];
		i++;
		k++;
	}

	while (j < n2) {
		arr[k] = R[j];
		j++;
		k++;
	}
}

void mergeSort(int arr[], int l, int r)
{
	if (l < r) {
		int m = l + (r - l) / 2;

		mergeSort(arr, l, m);
		mergeSort(arr, m + 1, r);

		merge(arr, l, m, r);
	}
}
```

```
void printArray(int A[], int size)
{
        int i;
        for (i = 0; i < size; i++)
                printf("%d ", A[i]);
        printf("\n");
}

int main()
{
    struct timeval current_time;
  double ts,dts,tms,dtms;
  gettimeofday(&current_time, NULL);
  ts=current_time.tv_sec;
  tms=current_time.tv_usec;
  int arr = (int)malloc(4 * 10000000);
  for(int i=0;i<10000000;i++)
  {
    arr[i]=rand();
  }
        int arr_size = sizeof(arr) / sizeof(arr[0]);
        printf("Given array is \n");
        printArray(arr, arr_size);
        mergeSort(arr, 0, arr_size - 1);
    gettimeofday(&current_time, NULL);
    dts=current_time.tv_sec-ts;
    dtms=current_time.tv_usec-tms;
        printf("\nSorted array is \n");
        printArray(arr, arr_size);
    printf("seconds : %lf\nmicro seconds : %lf",dts, dtms);
        return 0;
}
```

```
// Quick sort in C

#include <sys/time.h>
#include <stdio.h>
#include<stdlib.h>

void swap(int *a, int *b) {
  int t = *a;
  *a = *b;
  *b = t;
}

int partition(int array[], int low, int high) {

  int pivot = array[high];

  int i = (low - 1);

  for (int j = low; j < high; j++) {
    if (array[j] <= pivot) {

      i++;

      swap(&array[i], &array[j]);
    }
  }

  swap(&array[i + 1], &array[high]);

  return (i + 1);
}

void quickSort(int array[], int low, int high) {
  if (low < high) {

    int pi = partition(array, low, high);

    quickSort(array, low, pi - 1);

    quickSort(array, pi + 1, high);
  }
}

void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i) {
    printf("%d  ", array[i]);
  }
  printf("\n");
}

// main function
int main() {
  struct timeval current_time;
  double ts,dts,tms,dtms;

  gettimeofday(&current_time, NULL);
  ts=current_time.tv_sec;
  tms=current_time.tv_usec;

  int data = (int)malloc(4 * 100000);
  for(int i=0;i<100000;i++)
  {
    data[i]=rand();
  }

  int n = sizeof(data) / sizeof(data[0]);

  printf("Unsorted Array\n");
  printArray(data, n);
  quickSort(data, 0, n - 1);

  gettimeofday(&current_time, NULL);
  dts=current_time.tv_sec-ts;
  dtms=current_time.tv_usec-tms;

  printf("Sorted array in ascending order: \n");
  printArray(data, n);

  printf("seconds : %lf\nmicro seconds : %lf",dts, dtms);

  return 0;
}
```

```
#include <sys/time.h>
#include <stdio.h>
#include<stdlib.h>
void swap(int* a, int* b)
{

        int temp = *a;

        *a = *b;

        *b = temp;
}

// To heapify a subtree rooted with node i
// which is an index in arr[].
// n is size of heap
void heapify(int arr[], int N, int i)
{
        // Find largest among root, left child and right child

        // Initialize largest as root
        int largest = i;

        // left = 2*i + 1
        int left = 2 * i + 1;

        // right = 2*i + 2
        int right = 2 * i + 2;

        // If left child is larger than root
        if (left < N && arr[left] > arr[largest])

                largest = left;

        // If right child is larger than largest
        // so far
        if (right < N && arr[right] > arr[largest])

                largest = right;

        // Swap and continue heapifying if root is not largest
        // If largest is not root
        if (largest != i) {

                swap(&arr[i], &arr[largest]);

                // Recursively heapify the affected
                // sub-tree
                heapify(arr, N, largest);
        }
}

// Main function to do heap sort
void heapSort(int arr[], int N)
{
        // Build max heap
        for (int i = N / 2 - 1; i >= 0; i--)

                heapify(arr, N, i);

        // Heap sort
        for (int i = N - 1; i >= 0; i--) {

                swap(&arr[0], &arr[i]);

                // Heapify root element to get highest element at
                // root again
                heapify(arr, i, 0);
        }
}

// A utility function to print array of size n
void printArray(int arr[], int N)
{
        for (int i = 0; i < N; i++)
                printf("%d ", arr[i]);
        printf("\n");
}

// Driver's code
int main()
{
         struct timeval current_time;
  double ts,dts,tms,dtms;
  gettimeofday(&current_time, NULL);
  ts=current_time.tv_sec;
  tms=current_time.tv_usec;
  int arr = (int)malloc(4 * 10000000);
  for(int i=0;i<10000000;i++)
  {
   arr[i]=rand();
  }
        int N = sizeof(arr) / sizeof(arr[0]);

        // Function call
        heapSort(arr, N);
   gettimeofday(&current_time, NULL);
  dts=current_time.tv_sec-ts;
  dtms=current_time.tv_usec-tms;
        printf("Sorted array is\n");
        printArray(arr, N);
    printf("seconds : %lf\nmicro seconds : %lf",dts, dtms);
}
```