**Mini project on sorting in C**

**Name: Mayank Gajwe**

**Roll no: 21052080**

**Section: CSE-12**

**Code**:

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

*int* count\_merge = 0,count\_heap = 0;

*int* quicksort(*int* *number*[],*int* *first*,*int* *last*){

*int* count = 0;

*int* i, j, pivot, temp;

   if(*first*<*last*){

      pivot=*first*;

      i=*first*;

      j=*last*;

      while(i<j){

         while(*number*[i]<=*number*[pivot]&&i<*last*)

         i++;count++;

         while(*number*[j]>*number*[pivot])

         j--;count++;

         if(i<j){

            temp=*number*[i];

*number*[i]=*number*[j];

*number*[j]=temp;

         }

      }

      temp=*number*[pivot];

*number*[pivot]=*number*[j];

*number*[j]=temp;

      quicksort(*number*,*first*,j-1);

      quicksort(*number*,j+1,*last*);

   }

   return count;

}

*void* swap(*int*\* *a*, *int*\* *b*)

{

*int* temp = \**a*;

    \**a* = \**b*;

    \**b* = temp;

}

*void* merge(*int* *arr*[], *int* *l*, *int* *m*, *int* *r*)

{

*int* i, j, k;

*int* n1 = *m* - *l* + 1;

*int* n2 = *r* - *m*;

*int* L[n1], R[n2];

    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;

    j = 0;

    k = *l*;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

*arr*[k] = L[i];

            i++;

            count\_merge++;

        }

        else {

*arr*[k] = R[j];

            j++;

            count\_merge++;

        }

        k++;

        count\_merge++;

    }

    while (i < n1) {

*arr*[k] = L[i];

        i++;

        k++;

        count\_merge++;

    }

    while (j < n2) {

*arr*[k] = R[j];

        j++;

        k++;

        count\_merge++;

    }

}

*void* heapify(*int* *arr*[], *int* *N*, *int* *i*)

{

*int* largest = *i*;

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

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

    if (left < *N* && *arr*[left] > *arr*[largest])

        largest = left;

        count\_heap++;

    if (right < *N* && *arr*[right] > *arr*[largest])

        largest = right;

        count\_heap++;

    if (largest != *i*) {

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

        heapify(*arr*, *N*, largest);

        count\_heap++;

    }

}

clock\_t heapSort(*int* *arr*[], *int* *N*)

{

    for (*int* i = *N* / 2 - 1; i >= 0; i--)

        heapify(*arr*, *N*, i);

    clock\_t t;

    t = clock();

    for (*int* i = *N* - 1; i >= 0; i--) {

        swap(&*arr*[0], &*arr*[i]);

        heapify(*arr*, i, 0);

    }

    t = clock() - t;

    return t;

}

*void* mergeSort(*int* *arr*[], *int* *l*, *int* *r*)

{

    if (*l* < *r*) {

        count\_merge++;

*int* m = *l* + (*r* - *l*) / 2;

        mergeSort(*arr*, *l*, m);

        mergeSort(*arr*, m + 1, *r*);

        merge(*arr*, *l*, m, *r*);

    }

}

*void* print\_array(*int* *arr*[],*int* *n*){

    printf("\nPrinting Array:\n");

    for(*int* i=0;i<*n*;i++){

        printf("%d ",*arr*[i]);

    }

}

*int* main(){

    clock\_t t;

    srand(time(0));

*int* lower = 1000, upper = 100000;

    printf("Enter total number of elements: ");

*int* n;

    scanf("%d",&n);

*int* \*number = (*int*\*)malloc(n\*sizeof(*int*));

*int* \*number2 = (*int*\*)malloc(n\*sizeof(*int*));

*int* \*number3 = (*int*\*)malloc(n\*sizeof(*int*));

    for(*int* i = 0;i<n;i++){

        number[i] = (rand() % (upper - lower + 1)) + lower;

    }

    for(*int* i = 0;i<n;i++){

        number2[i] = (rand() % (upper - lower + 1)) + lower;

    }

    for(*int* i = 0;i<n;i++){

        number3[i] = (rand() % (upper - lower + 1)) + lower;

    }

    t = clock();

*int* c = quicksort(number,0,n-1);

    t = clock() - t;

*double* time\_taken = (((*double*)t)/CLOCKS\_PER\_SEC);

    printf("\nQuicksort: ");

    //print\_array(number,n);

    printf("\nQuicksort took %lf seconds to execute \nTotal comparisons: %d", time\_taken,c);

    printf("\nMergesort: ");

    t = clock();

    mergeSort(number,0,n-1);

    t = clock() - t;

    //print\_array(number,n);

    time\_taken = (((*double*)t)/CLOCKS\_PER\_SEC);

    printf("\nMergesort took %lf seconds to execute \nTotal comparisons: %d", time\_taken,count\_merge);

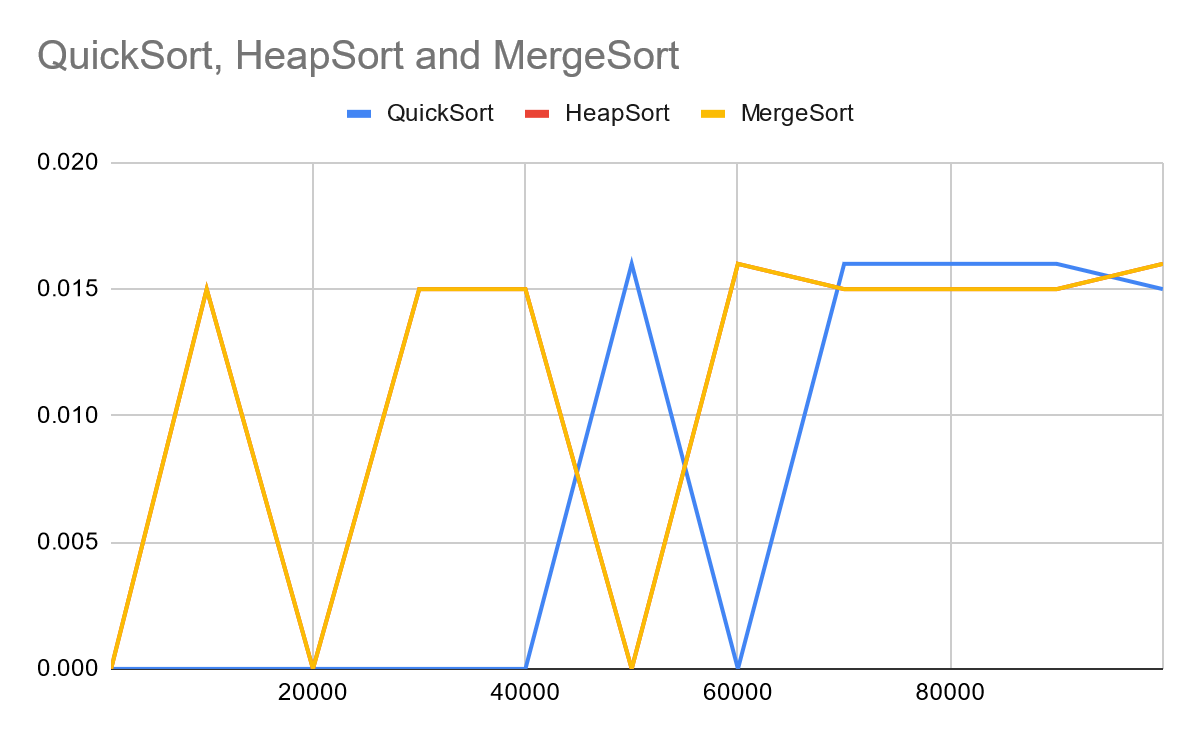
    printf("\nHeapsort: ");

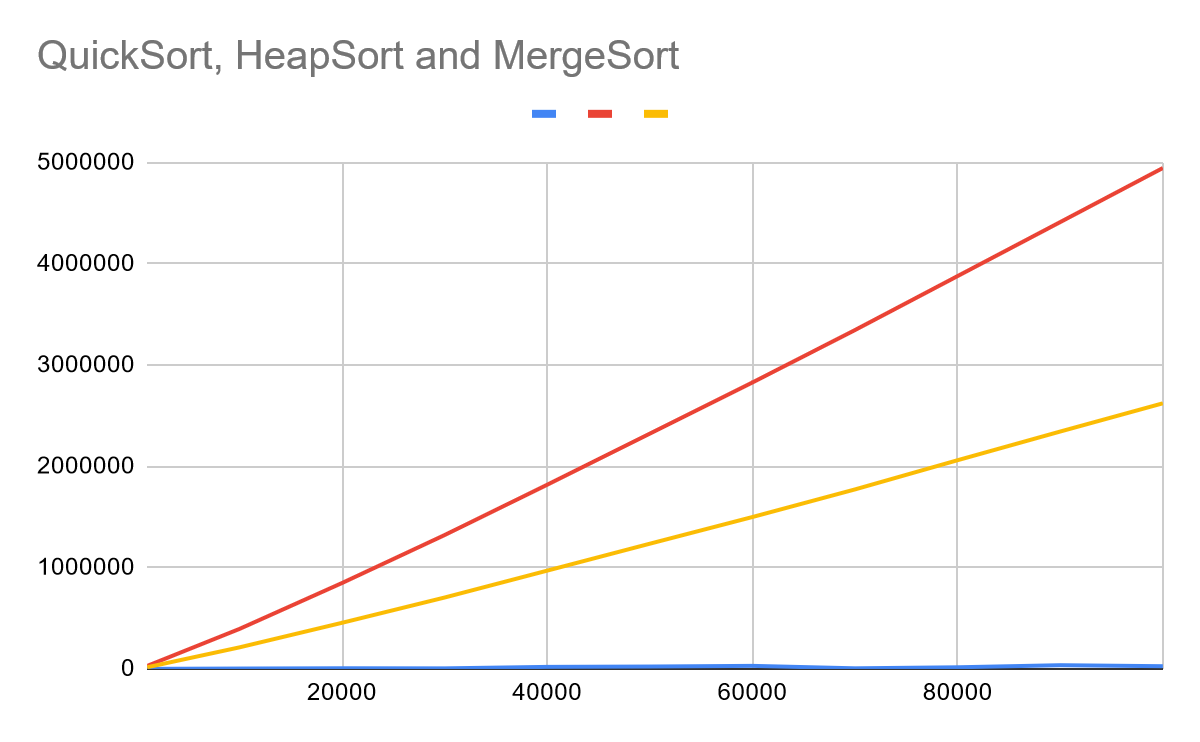
    clock\_t heaptime;

    heaptime = heapSort(number,n);

    //print\_array(number,n);

    printf("\nHeapsort took %lf seconds to execute \nTotal comparisons: %d", time\_taken,count\_heap);

**Graphs:**



**Conclusion:**

We can use quick sort for most cases as it has less total number of comparing steps making it easier to process, the comparison steps include all comparisons from comparing to add data to comparing to sort. In my opinion, Quicksort if fast and efficient among heap and merge, with merge taking second place.