# Topic - Graphically Comparing Big-O-runtime of Sorting Algorithms

* **Problem Statement**

Use the random number generator program written earlier to do the following:  
  
1. Implement Insertion Sort, Bubble Sort, Selection Sort, Merge Sort and Quick Sort each with the following signature:  
    void x\_sort(int arr[], int l, int r);  
where x is one of the above sorting algorithms, and l represents the index of the left-most element and r represents the index of the right-most element.  
Thus, to sort int arr[5] = {42, 17, 0, -20, -1}, using Merge Sort you would call  
merge\_sort(arr, 0, 4);  
2. Generate and store N random numbers in the range  [INT\_MIN, INT\_MAX] in a file, storing one number per line.  
3. Produce one such file for each of these values of N = {2^0, 2^1, 2^2, ..., 2^15, 2^16, 2^17, 2^18, 2^19}.  
3.1 For each of these files, make two additional copies per file, where the randomly generated numbers are sorted in ascending order, and another in descending order. For example, for a N = 2^2 = 4, we might have the first randomly generated file as:  
19  
-7  
3  
So, the other two files for that would be:  
-7  
3  
19  
and  
19  
3  
-7  
4. For each file, read the numbers in that file into an array, and sort that array using each of the above mentioned algorithms, recording the number of comparisons taken by each algorithm for that value of N.   
An example spreadsheet for the recorded values may look like:  
algorithm, N, comparisons\_taken  
merge, 1024, 10311  
quick, 1024, ...  
(You may skip the highest few powers of 2 for values of N if the sorting takes more than 15 mins to complete.)  
5. Plot the above observations as a line graph, taking X-axis = log2(N) and Y-axis = log2(mean\_comparisons\_taken), with one line plot per sorting algorithm, and plot all of them on the same graph with the same scale.

**Input example:** /\* Here user will mention the exe file, size and whole directory path of the fiile and the ouput csv file as a command line argument in command prompt\*/

“E:\sortingAssignment\sort\_graph.exe” 32 “E:\sortingAssignment\sortingfiles\input\_size-(2^5).txt” “E:\sortingAssignment\sort\_graph.csv”

**Output example:**

As ouput the output csv file will be get updated for the file input\_size-(2^5).txt as input command as

select,32,496

bubble,32,475

insert,32,235

merge,32,122

quick,32,122

/\* This data will keep on adding for various input.txt or ascending or descending input.txt files \*/

* **Proposed C Code**

**There are 2 codes one for the generation of the files (2^0 to 2^19) conataining random numbers and their ascending,descending files which contain the numbers in ascending, descending order. The other c code is for entering details in the sort\_graph.csv file i.e the ouput csv file.**

**/\* ---------- sortingfile\_generation.c--------------- \*/**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#include<time.h>

#include<limits.h>

/\* quick\_sort algorithm and the partition function \*/

int partition(int\* arr,int left,int right)

{

int pivot = arr[right];

int i = left-1;

for ( int j = left; j < right ; j++ )

{

if ( arr[j] < pivot )

{

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i+1];

arr[i+1] = arr[right];

arr[right] = temp;

return i+1;

}

void quick\_sort(int\* arr,int left,int right)

{

if ( left < right )

{

int part = partition(arr,left,right);

quick\_sort(arr,left,part-1);

quick\_sort(arr,part+1,right);

}

}

int main(){

/\* Generating 2^0 to 2^19 files which will contain random numbers and their 2 copies one for ascending,one for descending \*/

for( int i = 0 ; i < 20 ; i++ )

{

int length=pow(2,i);

int\* arr=(int \*)malloc(length\*sizeof(int));

FILE \*fptr,\*fasc,\*fdesc;/\* 3 FILE pointers for three types \*/

char\* str=(char\*)malloc(100\*sizeof(char));

sprintf(str,"E:\\sortingAssignment\\sortingfiles\\input\_size-(2^%d).txt",i);

fptr=fopen(str,"w");/\* creating or opening file in "write" mode \*/

srand(time(0));/\* setting time(0) in srand() to generate different random numbers at different time \*/

int r,k;

for(int i = 0;i < length;i++)

{

r = rand()%INT\_MAX + 1;

k = -r;

/\* desciding randomly when to give negative and positive random numbers \*/

if(r % 2 == 0)

{

fprintf(fptr,"%d\n",r);

arr[i] = r;

}

else

{

fprintf(fptr,"%d\n",k);

arr[i] = k;

}

}

for ( int j = 0 ; j < length ; j++ )

{

fscanf(fptr,"%d",&arr[j]);

}

quick\_sort(arr,0,length-1);/\*sorting the array by quick\_sort \*/

char\* str1=(char\*)malloc(100\*sizeof(char));

char\* str2=(char\*)malloc(100\*sizeof(char));

sprintf(str1,"E:\\sortingAssignment\\sortingfiles\\ascending\_size-(2^%d).txt",i);/\* for the ascending file \*/

sprintf(str2,"E:\\sortingAssignment\\sortingfiles\\descending\_size-(2^%d).txt",i);/\* for the descending file \*/

fasc = fopen(str1,"w");

fdesc = fopen(str2,"w");

for ( int j = 0 ; j < length ; j++ )

{

fprintf(fasc,"%d\n",arr[j]);/\* storing the sorted numbers \*/

}

for ( int j = length-1 ; j >= 0 ; j-- )

{

fprintf(fdesc,"%d\n",arr[j]);/\* storing the sorted numbers in descending order \*/

}

/\* closing the files and freeing the memory used \*/

fclose(fptr);

fclose(fasc);

fclose(fdesc);

free(arr);

free(str);

free(str1);

free(str2);

}

return 0;

}

**/\*------------------------------------------------------------------------------------------------------------------------- \*/**

**/\* ---------- sort\_graph.c--------------- \*/**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<limits.h>**

**#include<time.h>**

**#include<math.h>**

**unsigned long int compare\_count = 0;/\* global variable for counting comparison\*/**

**/\* insertion sort function \*/**

**void insertion\_sort(int\* arr,int left,int right)**

**{**

**compare\_count = 0;**

**int insert;**

**int check = 0;**

**for ( int i = left+1 ; i <= right ; i++ )**

**{**

**insert = arr[i];**

**int j;**

**check = 0;**

**for ( j = i-1 ; (( j >= 0 ) && ( arr[j] > insert )); j-- )**

**{**

**arr[j+1] = arr[j];**

**compare\_count++;**

**check = 1;**

**}**

**if ( check == 0 )**

**{**

**compare\_count++;**

**}**

**arr[j+1] = insert;**

**}**

**}**

**/\* bubble sort function \*/**

**void bubble\_sort(int \*arr,int left,int right )**

**{**

**int temp,sort=1;**

**compare\_count = 0;**

**for ( int i = left ; i < right ; i++ )**

**{**

**sort=1;**

**for ( int j = 0 ; j < right-i ; j++ )**

**{**

**if ( arr[j] > arr[j+1] )**

**{**

**temp=arr[j];**

**arr[j]=arr[j+1];**

**arr[j+1]=temp;**

**sort=0;**

**}**

**compare\_count++;**

**}**

**if(sort==1)**

**break;**

**}**

**}**

**/\* selection sort function \*/**

**void selection\_sort(int\* arr,int left,int right)**

**{**

**int minindex,select;**

**compare\_count = 0;**

**for ( int i = 0 ; i < right ; i++ )**

**{**

**minindex = i;**

**for ( int j= i+1 ; j <= right ; j++ )**

**{**

**if ( arr[j] < arr[minindex] )**

**{**

**minindex = j;**

**}**

**compare\_count++;**

**}**

**select = arr[i];**

**arr[i] = arr[minindex];**

**arr[minindex] = select;**

**}**

**}**

**/\* function for merge 2 sorted arrays \*/**

**void merging(int\* arr,int left,int mid,int right)**

**{**

**int i = left;**

**int j = mid + 1;**

**int k = 0;**

**int\* copy = (int\*)malloc((right - left + 1)\*sizeof(int));**

**while (( i <= mid ) && ( j <= right ))**

**{**

**if ( arr[i] < arr[j])**

**{**

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

**k++;**

**i++;**

**}**

**else**

**{**

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

**k++;**

**j++;**

**}**

**compare\_count++;**

**}**

**while ( i <= mid )**

**{**

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

**k++;**

**i++;**

**}**

**while ( j <= right )**

**{**

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

**k++;**

**j++;**

**}**

**for ( int i = 0 ; i <= (right - left) ; i++)**

**{**

**arr[left+i] = copy[i];**

**}**

**free(copy);/\* freeing the extra space \*/**

**}**

**/\* recursive merge sort function \*/**

**void merge\_sort(int\* arr,int left,int right)**

**{**

**int mid;**

**if ( left < right )**

**{**

**mid = (left + right)/2;**

**merge\_sort(arr,left,mid);**

**merge\_sort(arr,mid+1,right);**

**merging(arr,left,mid,right);**

**}**

**}**

**/\* partition for quick sort \*/**

**int partition(int\* arr,int left,int right)**

**{**

**int pivot = arr[right];**

**int i = left-1;**

**for ( int j = left; j < right ; j++ )**

**{**

**if ( arr[j] < pivot )**

**{**

**i++;**

**int temp = arr[i];**

**arr[i] = arr[j];**

**arr[j] = temp;**

**}**

**compare\_count++;**

**}**

**int temp = arr[i+1];**

**arr[i+1] = arr[right];**

**arr[right] = temp;**

**return i+1;**

**}**

**/\* recursive quick sort function \*/**

**void quick\_sort(int\* arr,int left,int right)**

**{**

**if ( left < right )**

**{**

**int part = partition(arr,left,right);**

**quick\_sort(arr,left,part-1);**

**quick\_sort(arr,part+1,right);**

**}**

**}**

**/\* function to copy to an array from original array \*/**

**int\* array\_copy(int\* arr\_original,int size )**

**{**

**int\* copy = (int\*)malloc(sizeof(int)\*size);**

**for ( int i = 0 ; i < size ; i++ )**

**{**

**copy[i] = arr\_original[i];**

**}**

**return copy;**

**}**

**int main(int argc,char\* argv[])**

**{**

**if ( argc != 4 ) /\* user should give 4 commands \*/**

**{**

**fprintf(stderr,"Enter in the format [program.exe] [input\_size] [input\_file.txt] [output\_file.csv]\n");**

**/\* the .exe file is the sort\_graph.exe \*/**

**/\* input\_size is the number of random numbers in the text file of (2^0 to 2^18) files and their ascending and descending input files also\*/**

**/\* output\_file.csv is the csv file where the sorting algo,size,and number of comparisons are listed \*/**

**exit(0);**

**}**

**int size = 0;**

**if (( sscanf(argv[1],"%d",&size) != 1) || ( size <= 0 ) || ( size != 1 && size % 2 != 0 ))**

**{**

**fprintf(stderr,"Give size as exponents of 2 from 2^0 to 2^18 \n");**

**exit(0);**

**}**

**FILE\* fptr = fopen(argv[2],"r");/\* reading the input file \*/**

**if ( fptr == NULL )**

**{**

**fprintf(stderr,"%s file is not found\n",argv[2]);**

**exit(0);**

**}**

**int\* arr = (int\*)malloc(sizeof(int)\*size);/\* array to store the data from input file \*/**

**for ( int i = 0 ; i < size ; i++ )**

**{**

**fscanf(fptr,"%d",&arr[i]);**

**}**

**fclose(fptr);/\* closing the input file \*/**

**FILE\* fout = fopen(argv[3],"a");/\* opening in append mode so that no data is erased in csv file \*/**

**int\* copy = array\_copy(arr,size);/\* copying from original array \*/**

**/\* selection sort \*/**

**compare\_count = 0;**

**selection\_sort(copy,0,size-1);**

**fprintf(fout,"%s,%d,%lu\n","select",size,compare\_count);/\* entering in csv file in this style \*/**

**free(copy);**

**copy = array\_copy(arr,size);/\* copying from original array \*/**

**/\* bubble sort \*/**

**compare\_count = 0;**

**bubble\_sort(copy,0,size-1);**

**fprintf(fout,"%s,%d,%lu\n","bubble",size,compare\_count);**

**free(copy);**

**copy = array\_copy(arr,size);/\* copying from original array \*/**

**/\* insertion sort \*/**

**compare\_count = 0;**

**insertion\_sort(copy,0,size-1);**

**fprintf(fout,"%s,%d,%lu\n","insert",size,compare\_count);**

**free(copy);**

**copy = array\_copy(arr,size);/\* copying from original array \*/**

**/\* merge sort \*/**

**compare\_count = 0;**

**merge\_sort(copy,0,size-1);**

**fprintf(fout,"%s,%d,%lu\n","merge",size,compare\_count);**

**free(copy);**

**copy = array\_copy(arr,size);/\* copying from original array \*/**

**/\* quick sort \*/**

**compare\_count = 0;**

**quick\_sort(copy,0,size-1);**

**fprintf(fout,"%s,%d,%lu\n","quick",size,compare\_count);**

**free(copy);**

**free(arr);/\* freeing the original array \*/**

**fclose(fout);/\* closing the output file \*/**

**return 0;**

**}**

**/\*------------------------------------------------------------------------------------------------------------------------- \*/**

* **Conclusion**

**The proposed algorithm for the different sorting have tme complexity as follows-**

1. **The selection sort algorithm has time comlexity of O(n^2).**
2. **The bubble sort algorithm has time comlexity of O(n^2).**
3. **The insertion sort algorithm has time comlexity of O(n^2).**
4. **The merge sort algorithm has time comlexity of O(nlog(n)).**
5. **The quick sort algorithm has average time comlexity of O(nlog(n)).**

* **Limitations : This program will take commands from the user .If the source code and the text files are not in same folder then we have to mention the whole path of directories i.e the whole location of the text file as command. For the files from 2^17 to 2^19 ,the program takes a lot more time to execute for various systems which is a limitation.**
* **Assumptions: We are skipping the files from 2^17 to 2^19 as the program takes a lot more time to execute .**