**REPORT : PROJECT 1**

**Submitted By**

1. <**update this with name and id**>

2. Jawad Chowdhury, 801135477

1. **Introduction to File Format and Data Structure**

In this project, we have separate folders for inputs and outputs. All the inputs are being located at the folder **input/** andallthe output files are being located at folder **output/.**

Now, each of these input and output folders contain 3 different formatted input and output folders named as 1. **random/** 2. **sorted/** and 3. **rev\_sorted/** (Reversely sorted).

Each of these folders contain different sample sized files ranging from 5000, 10000, 15000, ……………. to 50000.

So for example:

File **input/random/in\_5k.txt** is an input file containing 5000 samples in random order and file **output/rev\_sorted/out\_45k.txt** is an output file containing the sorted output of an input file which was reversely sorted with 45000 sample inputs.

Other than these, we have the **.cpp** file for Insertion sort (as **insertion\_sort.cpp**), Merge sort (as **merge\_sort.cpp**), Heapsort (as **heapsort.cpp**), In-place quicksort (as **inplace\_quicksort.cpp)** and Modified quicksort (as **modified\_quicksort.cpp).**

Also, we have different data generator .cpp files such as **data\_generator.cpp, data\_generator\_sorted.cpp, data\_generator\_rev\_sorted.cpp** for generating input data, one .cpp file to compare the solution of the algorithm with the library based sorted output named as **judge\_solution.cpp** and different shell scripts to calculate the time needed for each of the sorts such as **calculate\_time\_random\_io.sh, calculate\_time\_sorted\_io.sh** and **calculate\_time\_rev\_sorted\_io.sh.**

To implement the sorting algorithms, in most of the cases we used a globally declared array named **input\_arr** with a max size of 50005 (as the max size of our input samples is 50000). We used to run the sorting algorithms through the shell scripts specifying the sorting algorithm file to use, the input and the output files.

1. **Complexity Analysis**

For **Insertion sort** the running time of different formatted input files are as follows:

1. Sorted inputs (best case):
2. Reversely sorted inputs (worst case):
3. Random inputs (average case):

For **Merge sort** the running time of different formatted input files are as follows:

1. Sorted inputs:
2. Reversely sorted inputs:
3. Random inputs:

For **Heapsort** the running time of different formatted input files are as follows:

1. Sorted inputs (best case):
2. Reversely sorted inputs:
3. Random inputs:

For **In-place quicksort** the running time of different formatted input files are as follows:

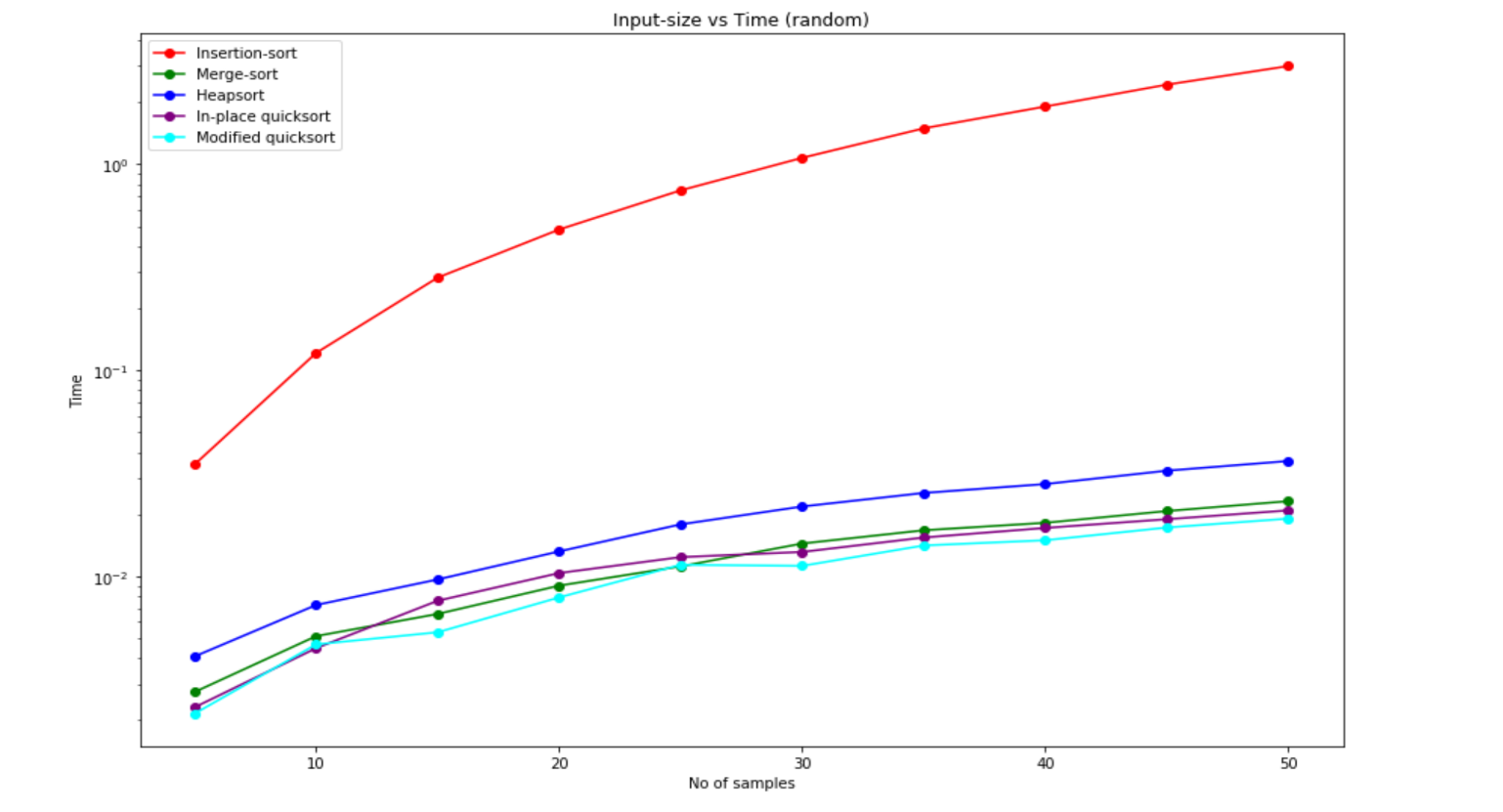
1. Sorted inputs (worst case):
2. Reversely sorted inputs (worst case):
3. Random inputs (average case):

For **Modified quicksort** the running time of different formatted input files are as follows:

1. Sorted inputs (worst case):
2. Reversely sorted inputs (worst case):
3. Random inputs (average case):
4. **Results**

Due to the **huge scaling difference** of time consumptions between the sorting algorithms for the different cases, we have used the **logarithmic** scale in the **y-axis** for the **input-size vs time** data plotting.

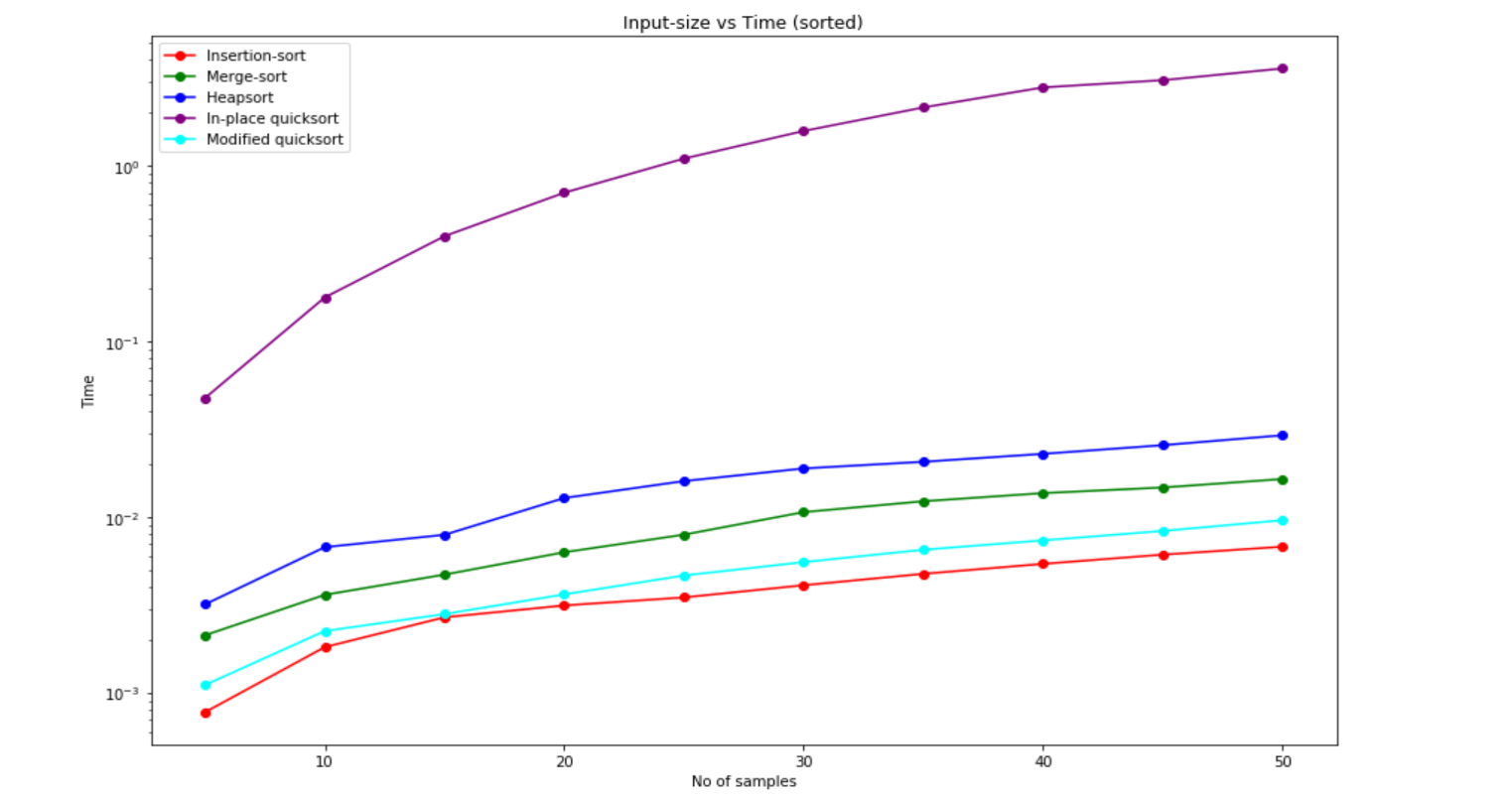
Figure 1: For randomly ordered data (input-size vs time)



In this figure, we can see the time consumption rate for insertion sort is significantly higher (even in the **logarithmic** scale) than all other sorting algorithms (merge sort, heapsort, in-place quicksort and modified quicksort).

As here, the plotting is being done for randomly ordered inputs, the insertion sorting algorithm takes where as most of the other algorithm does it in times and this is being reflected in the results of this figure.

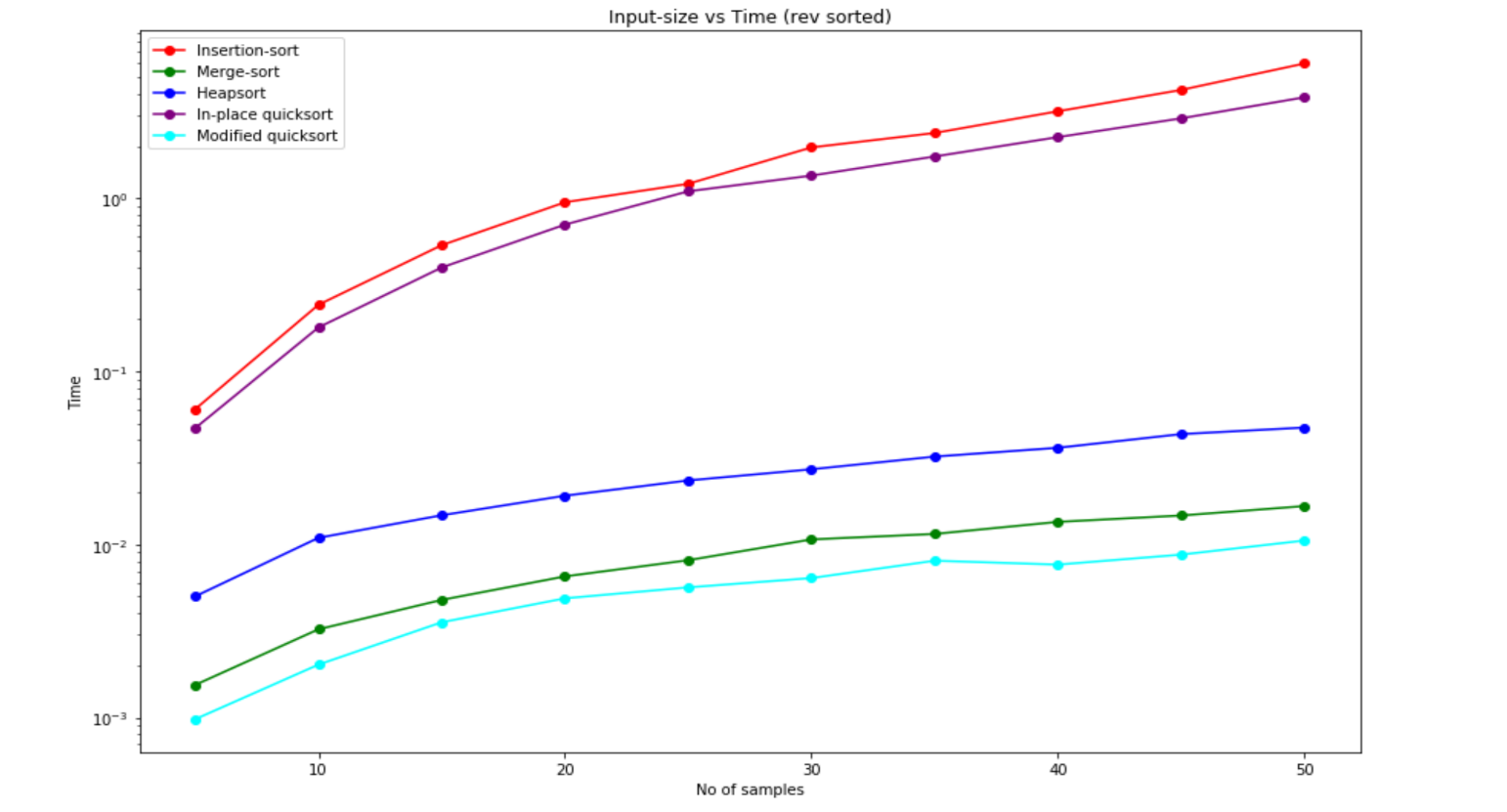
Figure 2: For sorted ordered data (input-size vs time)



In this figure, we can see the time consumption rate for in-place quicksort is much higher than all other sorting algorithms (insertion sort, merge sort, heapsort) as this is the worst case for this algorithm and it takes times.

On the other hand, the sorted input data makes the best case scenario for insertion sorting algorithm as from also the figure we can see the reflection cause in this case the algorithm take times.

Figure 3: For reversely sorted ordered data (input-size vs time)



In this figure, we can see the time consumption rate for both insertion and in-place quicksort is much higher than all other sorting algorithms ( merge sort, heapsort) as this is the worst case for both of these algorithm and they take times.

On the other hand, most of the other algorithm takes much lower time of having a time complexity of .

1. **Code**

1.Code for **insertion\_sort.cpp**

#pragma warning ( disable : 4786 )

#include <iostream>

#include <sstream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <cstring>

#include <string>

#include <algorithm>

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

#include <fstream>

#include <ctime>

using namespace std;

//#define \_rep( i, a, b, x ) for( \_\_typeof(b) i = ( a ); i <= ( b ); i += x )

#define \_rep(i, a, b, x) for( i = ( a ); i <= ( b ); i += x )

#define rep(i, n) \_rep( i, 0, n - 1, 1 )

#define ff first

#define ss second

#define pii pair< int, int >

#define psi pair< string, int >

#define pb(x) push\_back(x)

#define mp(x, y) make\_pair(x, y)

#define set(p) memset(p, -1, sizeof(p))

#define clr(p) memset(p, 0, sizeof(p))

//typedef long long i64;

//typedef \_\_int64 i64;

typedef long double ld;

//const double pi = (2.0\*acos(0.0));

const double pi = acos(-1.0);

const double eps = 1e-9;

const int inf = (1 << 28);

const int MAX = 50005;

int input\_arr[MAX];

int x;

int insertionSort(int n, int start, int end) {

int i,j,k;

for(j=start+1; j<=end; j++){

k=input\_arr[j];

i=j-1;

while(i>=0 && input\_arr[i]>k){

input\_arr[i+1]=input\_arr[i];

i--;

}

input\_arr[i+1]=k;

}

return 0;

}

int main() {

//freopen("in.txt", "r", stdin);

//freopen("out.txt", "w", stdout);

int i, j, k;

int test, t = 0, kase = 0;

//

int n;

scanf("%d", &x);

n = x;

for(i=0; i<n; i+=1) {

scanf("%d", &input\_arr[i]);

}

double st = clock();

int start =0;

int end = n-1;

insertionSort(n, start, end);

for(i=0; i<n; i+=1) {

printf("%d\n", input\_arr[i]);

}

cerr << (clock() - st) / CLOCKS\_PER\_SEC << endl;

//

return 0;

}

2.Code for **merge\_sort.cpp**

#pragma warning ( disable : 4786 )

#include <iostream>

#include <sstream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <cstring>

#include <string>

#include <algorithm>

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

#include <fstream>

#include <ctime>

using namespace std;

//#define \_rep( i, a, b, x ) for( \_\_typeof(b) i = ( a ); i <= ( b ); i += x )

#define \_rep(i, a, b, x) for( i = ( a ); i <= ( b ); i += x )

#define rep(i, n) \_rep( i, 0, n - 1, 1 )

#define ff first

#define ss second

#define pii pair< int, int >

#define psi pair< string, int >

#define pb(x) push\_back(x)

#define mp(x, y) make\_pair(x, y)

#define set(p) memset(p, -1, sizeof(p))

#define clr(p) memset(p, 0, sizeof(p))

//typedef long long i64;

//typedef \_\_int64 i64;

typedef long double ld;

//const double pi = (2.0\*acos(0.0));

const double pi = acos(-1.0);

const double eps = 1e-9;

const int inf = (1 << 28);

const int MAX = 50005;

int input\_arr[MAX];

int x;

int merge(int n, int start, int mid, int end){

int n1= mid - start +1;

int n2= end-mid;

int leftArray[n1];

int rightArray[n2];

int i,j,k;

for(i=0; i<n1; i++){

leftArray[i]=input\_arr[start+i];

}

for(j=0; j<n2; j++){

rightArray[j]=input\_arr[mid+1+j];

}

i=0, j=0, k=start;

while(i<n1 && j<n2){

if(leftArray[i]<=rightArray[j]){

input\_arr[k++]=leftArray[i++];

}else{

input\_arr[k++]=rightArray[j++];

}

}

while(i<n1){

input\_arr[k++]=leftArray[i++];

}

while(j<n2){

input\_arr[k++]=rightArray[j++];

}

return 0;

}

int mergeSort(int n, int start, int end) {

if (start < end){

int mid = (end+start)/2;

mergeSort(n, start, mid);

mergeSort(n, mid+1, end);

merge(n, start, mid, end);

}

return 0;

}

int main() {

//freopen("in.txt", "r", stdin);

//freopen("out.txt", "w", stdout);

int i, j, k;

int test, t = 0, kase = 0;

//

int n;

scanf("%d", &x);

n = x;

for(i=0; i<n; i+=1) {

scanf("%d", &input\_arr[i]);

}

double st = clock();

int start =0;

int end = n-1;

mergeSort(n, start, end);

for(i=0; i<n; i+=1) {

printf("%d\n", input\_arr[i]);

}

cerr << (clock() - st) / CLOCKS\_PER\_SEC << endl;

//

return 0;

}

3.Code for **heapsort.cpp**

#pragma warning ( disable : 4786 )

#include <iostream>

#include <sstream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <cstring>

#include <string>

#include <algorithm>

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

using namespace std;

//#define \_rep( i, a, b, x ) for( \_\_typeof(b) i = ( a ); i <= ( b ); i += x )

#define \_rep( i, a, b, x ) for( i = ( a ); i <= ( b ); i += x )

#define rep( i, n ) \_rep( i, 0, n - 1, 1 )

#define ff first

#define ss second

#define pii pair< int, int >

#define psi pair< string, int >

#define pb(x) push\_back(x)

#define mp(x, y) make\_pair(x, y)

#define set(p) memset(p, -1, sizeof(p))

#define clr(p) memset(p, 0, sizeof(p))

//typedef long long i64;

//typedef \_\_int64 i64;

typedef long double ld;

//const double pi = (2.0\*acos(0.0));

const double pi = acos(-1.0);

const double eps = 1e-9;

const int inf = (1<<28);

const int MAX = 50005;

int heap[MAX];

int input\_arr[MAX];

int n;

void heap\_insert(int key, int pos) {

heap[pos] = key;

int i = pos;

while(i>1 && heap[i/2] > heap[i]) {

swap(heap[i], heap[i/2]);

i = i / 2;

}

}

int heap\_remove\_min() {

int j, i = 1, tmp = heap[1];

heap[1] = heap[n];

n -= 1;

while(i < n) {

if(2\*i+1 <= n) { //heap node has 2 child

if(heap[i] <= heap[2\*i] && heap[i] <= heap[2\*i+1]) {

return tmp;

}

else {

if(heap[2\*i] < heap[2\*i+1]) j = 2\*i;

else j = 2\*i+1;

swap(heap[i], heap[j]);

i = j;

}

}

else { //heap node has 0 or 1 child

if(2\*i <= n && heap[i] > heap[2\*i]) {

swap(heap[i], heap[2\*i]);

}

return tmp;

}

}

return tmp;

}

int main() {

//freopen("input/in\_50k.txt", "r", stdin);

//freopen("out.txt", "w", stdout);

int i, j, k;

int test, t = 0, kase=0;

int key, \_n;

scanf("%d", &n);

\_n = n;

//printf("%d\n", n);

for(i=1; i<=\_n; i+=1) {

scanf("%d", &input\_arr[i]);

}

double st=clock();

for(i=1; i<=\_n; i+=1) {

heap\_insert(input\_arr[i], i);

}

for(i=0; i<\_n; i+=1) {

//if(i) printf(" ");

printf("%d\n", heap\_remove\_min());

}

//printf("\n");

cerr << (clock()-st)/CLOCKS\_PER\_SEC << endl;

return 0;

}

4.Code for **inplace\_quicksort.cpp**

#pragma warning ( disable : 4786 )

#include <iostream>

#include <sstream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <cstring>

#include <string>

#include <algorithm>

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

using namespace std;

//#define \_rep( i, a, b, x ) for( \_\_typeof(b) i = ( a ); i <= ( b ); i += x )

#define \_rep( i, a, b, x ) for( i = ( a ); i <= ( b ); i += x )

#define rep( i, n ) \_rep( i, 0, n - 1, 1 )

#define ff first

#define ss second

#define pii pair< int, int >

#define psi pair< string, int >

#define pb(x) push\_back(x)

#define mp(x, y) make\_pair(x, y)

#define set(p) memset(p, -1, sizeof(p))

#define clr(p) memset(p, 0, sizeof(p))

//typedef long long i64;

//typedef \_\_int64 i64;

typedef long double ld;

//const double pi = (2.0\*acos(0.0));

const double pi = acos(-1.0);

const double eps = 1e-9;

const int inf = (1<<28);

const int MAX = 50005;

int n;

int input\_arr[MAX];

int get\_pivot(int left, int right) {

return input\_arr[left];

}

void inplace\_quick\_sort(int left, int right) {

//base case

if(left >= right) return;

int pivot = get\_pivot(left, right);

int i = left, j = right;

while(true) {

while(input\_arr[i] < pivot) i+=1;

while(pivot < input\_arr[j]) j -= 1;

if(i < j) {

swap(input\_arr[i], input\_arr[j]);

i += 1;

j -= 1;

}

else break;

}

inplace\_quick\_sort(left, i-1);

inplace\_quick\_sort(j+1, right);

}

int main() {

//freopen("input/random/in\_50k.txt", "r", stdin);

//freopen("out.txt", "w", stdout);

int i, j, k;

int test, t = 0, kase=0;

scanf("%d", &n);

for(i=0; i<n; i+=1) {

scanf("%d", &input\_arr[i]);

}

double st=clock();

inplace\_quick\_sort(0, n-1);

for(i=0; i<n; i+=1) {

printf("%d\n", input\_arr[i]);

}

cerr << (clock()-st)/CLOCKS\_PER\_SEC << endl;

return 0;

}

5.Code for **modified\_quicksort.cpp**

#pragma warning ( disable : 4786 )

#include <iostream>

#include <sstream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <cstring>

#include <string>

#include <algorithm>

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

using namespace std;

//#define \_rep( i, a, b, x ) for( \_\_typeof(b) i = ( a ); i <= ( b ); i += x )

#define \_rep(i, a, b, x) for( i = ( a ); i <= ( b ); i += x )

#define rep(i, n) \_rep( i, 0, n - 1, 1 )

#define ff first

#define ss second

#define pii pair< int, int >

#define psi pair< string, int >

#define pb(x) push\_back(x)

#define mp(x, y) make\_pair(x, y)

#define set(p) memset(p, -1, sizeof(p))

#define clr(p) memset(p, 0, sizeof(p))

//typedef long long i64;

//typedef \_\_int64 i64;

typedef long double ld;

//const double pi = (2.0\*acos(0.0));

const double pi = acos(-1.0);

const double eps = 1e-9;

const int inf = (1 << 28);

const int MAX = 50005;

int n;

int input\_arr[MAX];

int median\_of\_three(int a, int b, int c) {

if ((b <= a && a <= c) || (c <= a && a <= b)) return a;

if ((a <= b && b <= c) || (c <= b && b <= a)) return b;

return c;

}

int get\_pivot(int left, int right) {

int mid = (left + right) / 2;

return median\_of\_three(input\_arr[left], input\_arr[mid], input\_arr[right]);

}

int insertion\_sort(int start, int end) {

int i, j, k;

for (j = start + 1; j <= end; j++) {

k = input\_arr[j];

i = j - 1;

while (i >= 0 && input\_arr[i] > k) {

input\_arr[i + 1] = input\_arr[i];

i--;

}

input\_arr[i + 1] = k;

}

return 0;

}

void inplace\_quick\_sort(int left, int right) {

//base case

if (left + 10 <= right) {

int pivot = get\_pivot(left, right);

int i = left, j = right;

while (true) {

while (input\_arr[i] < pivot) i += 1;

while (pivot < input\_arr[j]) j -= 1;

if (i < j) {

swap(input\_arr[i], input\_arr[j]);

i += 1;

j -= 1;

} else break;

}

inplace\_quick\_sort(left, i - 1);

inplace\_quick\_sort(j + 1, right);

} else {

insertion\_sort(left, right);

}

}

int main() {

//freopen("input/random/in\_40k.txt", "r", stdin);

//freopen("out.txt", "w", stdout);

int i, j, k;

int test, t = 0, kase = 0;

scanf("%d", &n);

for (i = 0; i < n; i += 1) {

scanf("%d", &input\_arr[i]);

}

double st = clock();

inplace\_quick\_sort(0, n - 1);

for (i = 0; i < n; i += 1) {

printf("%d\n", input\_arr[i]);

}

cerr << (clock() - st) / CLOCKS\_PER\_SEC << endl;

return 0;

}

6.Code for **judge\_solution.cpp**

#pragma warning ( disable : 4786 )

#include <iostream>

#include <sstream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <cstring>

#include <string>

#include <algorithm>

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

using namespace std;

//#define \_rep( i, a, b, x ) for( \_\_typeof(b) i = ( a ); i <= ( b ); i += x )

#define \_rep( i, a, b, x ) for( i = ( a ); i <= ( b ); i += x )

#define rep( i, n ) \_rep( i, 0, n - 1, 1 )

#define ff first

#define ss second

#define pii pair< int, int >

#define psi pair< string, int >

#define pb(x) push\_back(x)

#define mp(x, y) make\_pair(x, y)

#define set(p) memset(p, -1, sizeof(p))

#define clr(p) memset(p, 0, sizeof(p))

//typedef long long i64;

//typedef \_\_int64 i64;

typedef long double ld;

//const double pi = (2.0\*acos(0.0));

const double pi = acos(-1.0);

const double eps = 1e-9;

const int inf = (1<<28);

const int MAX = 50005;

int arr[MAX];

int main() {

freopen("input/in\_50k.txt", "r", stdin);

freopen("output/out\_50k.txt", "w", stdout);

int i, j, k;

int test, t = 0, kase=0;

int n;

scanf("%d", &n);

for(i=0; i<n; i+=1) scanf("%d", &arr[i]);

sort(arr, arr+n);

for(i=0; i<n; i+=1) {

//if(i) printf(" ");

printf("%d\n", arr[i]);

}

//printf("\n");

return 0;

}

7.Code for Time Calculation (i.e **calculate\_time\_random\_io.sh**)

#!/bin/bash

#echo "Algorithm: Insertion Sort"

#echo "Input-set: random"

input\_path="input/random/"

g++ -o insertion\_sort insertion\_sort.cpp

for file in $input\_path\*; do

#echo "Input File: ${file##\*/}"

counter=1

execution\_time=0.0

while [ $counter -le 3 ]

do

./insertion\_sort < $input\_path${file##\*/} > output.txt

((counter++))

done

done

#echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

#echo "Algorithm: Merge Sort"

#echo "Input-set: random"

g++ -o merge\_sort merge\_sort.cpp

for file in $input\_path\*; do

#echo "Input File: ${file##\*/}"

counter=1

execution\_time=0.0

while [ $counter -le 3 ]

do

./merge\_sort < $input\_path${file##\*/} > output.txt

((counter++))

done

done

#echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

#echo "Algorithm: Heap Sort"

#echo "Input-set: random"

g++ -o heapsort heapsort.cpp

for file in $input\_path\*; do

#echo "Input File: ${file##\*/}"

counter=1

execution\_time=0.0

while [ $counter -le 3 ]

do

./heapsort < $input\_path${file##\*/} > output.txt

((counter++))

done

done

#echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

#echo "Algorithm: In-place Quick Sort"

#echo "Input-set: random"

g++ -o inplace\_quicksort inplace\_quicksort.cpp

for file in $input\_path\*; do

#echo "Input File: ${file##\*/}"

counter=1

execution\_time=0.0

while [ $counter -le 3 ]

do

./inplace\_quicksort < $input\_path${file##\*/} > output.txt

((counter++))

done

done

#echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

#echo "Algorithm: Modified Quick Sort"

#echo "Input-set: random"

g++ -o modified\_quicksort modified\_quicksort.cpp

for file in $input\_path\*; do

#echo "Input File: ${file##\*/}"

counter=1

execution\_time=0.0

while [ $counter -le 3 ]

do

./modified\_quicksort < $input\_path${file##\*/} > output.txt

((counter++))

done

done

8.Code for input data generation (i.e. **data\_generator.cpp**)

#pragma warning ( disable : 4786 )

#include <iostream>

#include <sstream>

#include <cstdio>

#include <cstdlib>

#include <cmath>

#include <cctype>

#include <cstring>

#include <string>

#include <algorithm>

#include <vector>

#include <list>

#include <map>

#include <set>

#include <deque>

#include <queue>

#include <stack>

#include <ctime>

using namespace std;

//#define \_rep( i, a, b, x ) for( \_\_typeof(b) i = ( a ); i <= ( b ); i += x )

#define \_rep( i, a, b, x ) for( i = ( a ); i <= ( b ); i += x )

#define rep( i, n ) \_rep( i, 0, n - 1, 1 )

#define ff first

#define ss second

#define pii pair< int, int >

#define psi pair< string, int >

#define pb(x) push\_back(x)

#define mp(x, y) make\_pair(x, y)

#define set(p) memset(p, -1, sizeof(p))

#define clr(p) memset(p, 0, sizeof(p))

//typedef long long i64;

//typedef \_\_int64 i64;

typedef long double ld;

//const double pi = (2.0\*acos(0.0));

const double pi = acos(-1.0);

const double eps = 1e-9;

const int inf = (1<<28);

const int MAX = 50000;

const int MAXN = 100000000;

int main() {

freopen("in\_50k.txt", "w", stdout);

int i, j, k;

srand (time(NULL));

printf("%d\n", MAX);

for(i=0; i<MAX; i+=1) {

j = rand() % MAXN;

if(i) printf(" ");

printf("%d", j);

}

printf("\n");

return 0;

}

9.Code for data plotting **algo\_p1\_plot\_1.ipynb**

# importing

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

r\_df = pd.read\_fwf('execution\_time\_random\_io.txt', header=None)

s\_df = pd.read\_fwf('execution\_time\_sorted\_io.txt', header=None)

rs\_df = pd.read\_fwf('execution\_time\_rev\_sorted\_io.txt', header=None)

# numpy representation

# random

r\_m1=r\_df.values

r\_m2=r\_m1.reshape((5,30))

r\_m3=np.zeros((5,10))

for j in range(10):

r\_m3[:,j]= np.mean(r\_m2[:,j\*3:(j+1)\*3], axis=1)

r\_m3

# sorted

s\_m1=s\_df.values

s\_m2=s\_m1.reshape((5,30))

s\_m3=np.zeros((5,10))

for j in range(10):

s\_m3[:,j]= np.mean(s\_m2[:,j\*3:(j+1)\*3], axis=1)

s\_m3

# rev sorted

rs\_m1=rs\_df.values

rs\_m2=rs\_m1.reshape((5,30))

rs\_m3=np.zeros((5,10))

for j in range(10):

rs\_m3[:,j]= np.mean(rs\_m2[:,j\*3:(j+1)\*3], axis=1)

rs\_m3

# plotting

list\_is = [5,10,15,20,25,30,35,40,45,50]

# random

fig1 = plt.figure(figsize=(12,8))

plt.clf()

plt.title('Input-size vs Time (random)')

plt.plot(list\_is, r\_m3[0,:].reshape(-1,1), 'go-', color='red', label="Insertion-sort")

plt.plot(list\_is, r\_m3[1,:].reshape(-1,1), 'go-', color='green', label="Merge-sort")

plt.plot(list\_is, r\_m3[2,:].reshape(-1,1), 'go-', color='blue', label="Heapsort")

plt.plot(list\_is, r\_m3[3,:].reshape(-1,1), 'go-', color='purple', label="In-place quicksort")

plt.plot(list\_is, r\_m3[4,:].reshape(-1,1), 'go-', color='seagreen', label="Modified quicksort")

plt.yscale('log')

plt.ylabel('Time')

plt.xlabel('No of samples')

plt.legend(loc='best')

fig1.tight\_layout()

# # sorted

fig2 = plt.figure(figsize=(12,8))

plt.clf()

plt.title('Input-size vs Time (sorted)')

plt.plot(list\_is, s\_m3[0,:].reshape(-1,1), 'go-', color='red', label="Insertion-sort")

plt.plot(list\_is, s\_m3[1,:].reshape(-1,1), 'go-', color='green', label="Merge-sort")

plt.plot(list\_is, s\_m3[2,:].reshape(-1,1), 'go-', color='blue', label="Heapsort")

plt.plot(list\_is, s\_m3[3,:].reshape(-1,1), 'go-', color='purple', label="In-place quicksort")

plt.plot(list\_is, s\_m3[4,:].reshape(-1,1), 'go-', color='seagreen', label="Modified quicksort")

plt.yscale('log')

plt.ylabel('Time')

plt.xlabel('No of samples')

plt.legend(loc='best')

fig2.tight\_layout()

# # rev sorted

fig3 = plt.figure(figsize=(12,8))

plt.clf()

plt.title('Input-size vs Time (rev sorted)')

plt.plot(list\_is, rs\_m3[0,:].reshape(-1,1), 'go-', color='red', label="Insertion-sort")

plt.plot(list\_is, rs\_m3[1,:].reshape(-1,1), 'go-', color='green', label="Merge-sort")

plt.plot(list\_is, rs\_m3[2,:].reshape(-1,1), 'go-', color='blue', label="Heapsort")

plt.plot(list\_is, rs\_m3[3,:].reshape(-1,1), 'go-', color='purple', label="In-place quicksort")

plt.plot(list\_is, rs\_m3[4,:].reshape(-1,1), 'go-', color='seagreen', label="Modified quicksort")

plt.yscale('log')

plt.ylabel('Time')

plt.xlabel('No of samples')

plt.legend(loc='best')

fig3.tight\_layout()