

Output:-   
Exp for:- file1.txt

Bubble sort time: 5 milliseconds

Insertion sort time: 1 milliseconds

Selection sort time: 3 milliseconds

Heap sort time: 0 milliseconds

Qucik sort time: 0 milliseconds

Exp for:- file2.txt

Bubble sort time: 32 milliseconds

Insertion sort time: 7 milliseconds

Selection sort time: 14 milliseconds

Heap sort time: 0 milliseconds

Qucik sort time: 0 milliseconds

Exp for:- file3.txt

Bubble sort time: 198 milliseconds

Insertion sort time: 64 milliseconds

Selection sort time: 81 milliseconds

Heap sort time: 1 milliseconds

Qucik sort time: 1 milliseconds

Exp for:- file4.txt

Bubble sort time: 584 milliseconds

Insertion sort time: 270 milliseconds

Selection sort time: 186 milliseconds

Heap sort time: 2 milliseconds

Qucik sort time: 2 milliseconds

Exp for:- file5.txt

Bubble sort time: 1056 milliseconds

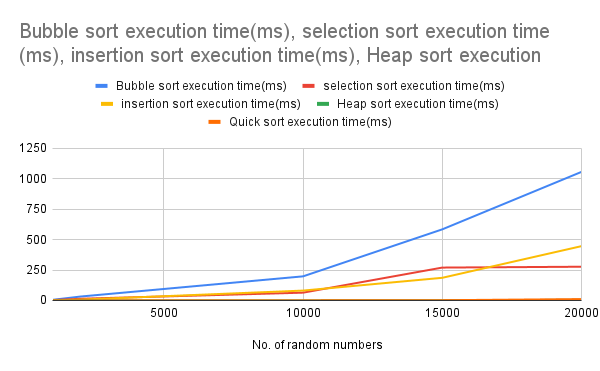
Insertion sort time: 277 milliseconds

Selection sort time: 446 milliseconds

Heap sort time: 5 milliseconds

Qucik sort time: 11 milliseconds

Visualization :-



Code :-

Main.cpp >>

#include <bits/stdc++.h>

using namespace std;

#include <chrono> // Include the chrono library

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

int temp =\*a;

\*a = \*b;

\*b = temp;

}

void selection\_sort(vector<int> &v)

{

int n = v.size();

for (int i = 0; i < n - 1; i++)

{

int min\_idx = i;

for (int j = i + 1; j < n; j++)

{

if (v[j] < v[min\_idx])

{

min\_idx = j;

}

}

swap(v[i], v[min\_idx]);

}

}

void insertion\_sort(vector<int> &v)

{

int n = v.size();

for (int i = 1; i < n; i++)

{

int key = v[i];

int j = i - 1;

while (j >= 0 && v[j] > key)

{

v[j + 1] = v[j];

j = j - 1;

}

v[j + 1] = key;

}

}

void bubble\_sort(vector<int> &v)

{

int n = v.size();

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (v[j] > v[j + 1])

{

swap(v[j], v[j + 1]);

}

}

}

}

void heapifyUp(vector<int> &v,int index){

cout<<"HeapifyUp"<<endl;

while(index!=0 && v[(index-1)/2]<v[index]){

swap(&v[(index-1)/2],&v[index]);

index = (index-1) / 2;

}

}

void heapifyDown(vector<int> &v,int index,int heapSize){

//cout<<"heapifyDown"<<endl;

int largest = index;

int left = 2\*index+1;

int right = 2\*index+2;

if(left < heapSize && v[left] > v[largest] )

largest = left;

if(right < heapSize && v[right] > v[largest])

largest = right;

if (largest != index){

swap(&v[index],&v[largest]);

heapifyDown(v,largest,heapSize);

}

}

auto build\_heap(vector<int>&v){

//cout<<"build\_heap"<<endl;

int heapSize= v.size();

int last\_parent = (heapSize/2)-1;

for(int i=last\_parent;i>=0;i--)

heapifyDown(v,i,heapSize);

return v;

}

void heap\_sort(vector<int> &arr){

//cout<<"Heap\_sort time"<<endl;

int count = 0;

int heapSize=arr.size();

vector<int> v=build\_heap(arr);

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

int x=v[0];

v[0]=v[i];

heapifyDown(v,0,heapSize);

heapSize--;

v[i]=x;

count++;

}

// for(int i=0;i<=count;i++){

// printf("%d ",v[i]);

// }

// printf("\n");

return;

}

int part(vector<int> &v,int li,int ui){

int start = li;

int end = ui;

int pivot = v[start];

while(start<end){

while(v[start]<=pivot)

start++;

while(v[end]>pivot)

end--;

if(start<end)

swap(v[start],v[end]);

}

swap(v[end],v[li]);

return end;

}

void quick\_sort(vector<int> &v,int li,int ui){

int loc;

if(li<ui){

loc=part(v,li,ui);

quick\_sort(v,li,loc-1);

quick\_sort(v,loc+1,ui);

}

}

vector<int> numberfetch(const string &filename)

{

// Open the input file

ifstream i\_data(filename);

if (!i\_data)

{

cout << "Error opening file" << endl;

}

string line;

vector<int> v;

while (getline(i\_data, line))

{

// Remove extra spaces, if any, around commas

stringstream ss(line);

string token;

while (getline(ss, token, ','))

{

// Trim potential spaces around the token

token.erase(0, token.find\_first\_not\_of(" \t"));

token.erase(token.find\_last\_not\_of(" \t") + 1);

v.push\_back(stoi(token));

}

}

// // Display the numbers

// for (size\_t i = 0; i < v.size(); ++i)

// {

// cout << v[i];

// if (i != v.size() - 1) cout << ", "; // Print a comma except for the last number

// }

// cout << endl;

return v;

}

void print\_vec(vector<int> v)

{

for (size\_t i = 0; i < v.size(); ++i)

{

cout << v[i];

if (i != v.size() - 1)

cout << ", "; // Print a comma except for the last number

}

cout << endl;

}

float bubble\_time(vector<int> &v)

{

// Bubble Sort

auto bubbleSortStart = chrono::high\_resolution\_clock::now();

bubble\_sort(v);

auto bubbleSortEnd = chrono::high\_resolution\_clock::now();

auto bubbleSortDuration = chrono::duration\_cast<chrono::milliseconds>(bubbleSortEnd - bubbleSortStart);

cout << "Bubble sort time: " << bubbleSortDuration.count() << " milliseconds" << endl;

return bubbleSortDuration.count();

}

float heap\_time(vector<int> &v)

{

// Heap Sort

auto heapSortStart = chrono::high\_resolution\_clock::now();

heap\_sort(v);

auto heapSortEnd = chrono::high\_resolution\_clock::now();

auto heapSortDuration = chrono::duration\_cast<chrono::milliseconds>(heapSortEnd - heapSortStart);

cout << "Heap sort time: " << heapSortDuration.count() << " milliseconds" << endl;

// print\_vec(v);

return heapSortDuration.count();

}

float quick\_time(vector<int> &v)

{

int n=v.size();

// Quick Sort

auto quickSortStart = chrono::high\_resolution\_clock::now();

quick\_sort(v,0,n);

auto quickSortEnd = chrono::high\_resolution\_clock::now();

auto quickSortDuration = chrono::duration\_cast<chrono::milliseconds>(quickSortEnd - quickSortStart);

cout << "Qucik sort time: " << quickSortDuration.count() << " milliseconds" << endl;

//print\_vec(v);

return quickSortDuration.count();

}

float selection\_time(vector<int> &v)

{

// Selection Sort

auto selectionSortStart = chrono::high\_resolution\_clock::now();

selection\_sort(v);

auto selectionSortEnd = chrono::high\_resolution\_clock::now();

auto selectionSortDuration = chrono::duration\_cast<chrono::milliseconds>(selectionSortEnd - selectionSortStart);

cout << "Selection sort time: " << selectionSortDuration.count() << " milliseconds" << endl;

return selectionSortDuration.count();

}

float insertion\_time(vector<int> &v)

{

// Insertion Sort

// Reset to original data

auto insertionSortStart = chrono::high\_resolution\_clock::now();

insertion\_sort(v);

auto insertionSortEnd = chrono::high\_resolution\_clock::now();

auto insertionSortDuration = chrono::duration\_cast<chrono::milliseconds>(insertionSortEnd - insertionSortStart);

cout << "Insertion sort time: " << insertionSortDuration.count() << " milliseconds" << endl;

return insertionSortDuration.count();

}

int main()

{

vector<int> original\_data = numberfetch("file1.txt");

vector<int> v = original\_data;

ofstream o\_data("output.txt", ios::app);

if (!o\_data)

{

cout << "Error creating output file" << endl;

return 1;

}

o\_data << "Exp no." << "," << "bubble sort(ms)" << "," << "selection sort(ms)" << "," << "insertion sort" << endl;

for (int i = 0; i < 5; i++)

{

string filename = "file" + to\_string(i + 1) + ".txt";

cout<<"Exp for:- "<<filename<<endl;

vector<int> original\_data = numberfetch(filename);

// 1. bubble sort

vector<int> v = original\_data;

float b\_time = bubble\_time(v);

// 2. insertion sort

v = original\_data;

float i\_time = insertion\_time(v);

// 3. selection sort

v = original\_data;

float s\_time = selection\_time(v);

// 4. heap sort

v = original\_data;

float h\_time = heap\_time(v);

// 5. quick sort

v = original\_data;

float q\_time = quick\_time(v);

o\_data << "Exp" + to\_string(i + 1) << "," << b\_time << "," << s\_time << "," << i\_time <<","<<h\_time<<","<<q\_time<< endl;

}

return 0;

}

Random.cpp >>>

#include <bits/stdc++.h>

using namespace std;

int random\_number(int min, int max)

{

return (rand() % (max - min + 1)) + min;

}

int main()

{

// Open the output file in append mode

for (int i = 0; i < 5; i++)

{

string filename = "file" + to\_string(i+1) + ".txt";

ofstream o\_data(filename, ios::app);

if (!o\_data)

{

cout << "Error creating output file" << endl;

return 1;

}

int min = 1;

int max = 100;

long int n;

cout << "enter number of random integers for"<<filename<<":-";

cin >> n;

int a[n];

for (int i = 0; i < n; i++) // genrating random numbers

{

a[i] = random\_number(min, max);

}

// saving random numbers

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

{

o\_data << a[i];

if (i != n - 1)

{

o\_data << ", ";

}

}

o\_data << endl;

// Close the output file

o\_data.close();

}

return 0;

}