**Lab Exercise - 1**

* AIM ::

WAP in C++ to implement Bubble, Merge, Quick & Insertion Sort and also evaluate time in each.

**1. Bubble Sort**

Source\_Code ::

### #include <ctime>

### #include <iostream>

### #include <vector>

### using namespace std;

### void bubbleSort(vector<int>& arr)

### {

### int n = arr.size();

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

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

### if (arr[j] > arr[j + 1]) {

### swap(arr[j], arr[j + 1]);

### }

### }

### }

### }

### int main()

### {

### cout << "5C6 - Amit Singhal (11614802722)" << endl;

### vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };

### cout << "Unsorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### clock\_t start = clock();

### bubbleSort(arr);

### clock\_t end = clock();

### cout << "Bubble Sort:" << endl;

### cout << "Sorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### double time\_taken\_ms = double(end - start) \* 1000.0

### / CLOCKS\_PER\_SEC; // Convert to milliseconds

### cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;

### return 0;

### }

### 

Output ::

**2. Merge Sort**

Source\_Code ::

### #include <ctime>

### #include <iostream>

### #include <vector>

### using namespace std;

### void merge(vector<int>& arr, int l, int m, int r)

### {

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

### int n2 = r - m;

### vector<int> L(n1), R(n2);

### for (int i = 0; i < n1; i++)

### L[i] = arr[l + i];

### for (int i = 0; i < n2; i++)

### R[i] = arr[m + 1 + i];

### int i = 0, j = 0, k = l;

### 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(vector<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);

### }

### }

### int main()

### {

### cout << "5C6 - Amit Singhal (11614802722)" << endl;

### vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };

### cout << "Unsorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### clock\_t start = clock();

### mergeSort(arr, 0, arr.size() - 1);

### clock\_t end = clock();

### cout << "Merge Sort:" << endl;

### cout << "Sorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### double time\_taken\_ms = double(end - start) \* 1000.0

### / CLOCKS\_PER\_SEC; // Convert to milliseconds

### cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;

### return 0;

### }

Output ::

### 

**3. Quick Sort**

Source\_Code ::

### #include <ctime>

### #include <iostream>

### #include <vector>

### using namespace std;

### int partition(vector<int>& arr, int low, int high)

### {

### int pivot = arr[high];

### int i = (low - 1);

### for (int j = low; j <= high - 1; j++) {

### if (arr[j] < pivot) {

### i++;

### swap(arr[i], arr[j]);

### }

### }

### swap(arr[i + 1], arr[high]);

### return (i + 1);

### }

### void quickSort(vector<int>& arr, int low, int high)

### {

### if (low < high) {

### int pi = partition(arr, low, high);

### quickSort(arr, low, pi - 1);

### quickSort(arr, pi + 1, high);

### }

### }

### int main()

### {

### cout << "5C6 - Amit Singhal (11614802722)" << endl;

### vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };

### cout << "Unsorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### clock\_t start = clock();

### quickSort(arr, 0, arr.size() - 1);

### clock\_t end = clock();

### cout << "Quick Sort:" << endl;

### cout << "Sorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### double time\_taken\_ms = double(end - start) \* 1000.0

### / CLOCKS\_PER\_SEC; // Convert to milliseconds

### cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;

### return 0;

### }

### 

Output ::

**4. Insertion Sort**

Source\_Code ::

### #include <ctime>

### #include <iostream>

### #include <vector>

### using namespace std;

### void insertionSort(vector<int>& arr)

### {

### int n = arr.size();

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

### int key = arr[i];

### int j = i - 1;

### while (j >= 0 && arr[j] > key) {

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

### j = j - 1;

### }

### arr[j + 1] = key;

### }

### }

### int main()

### {

### cout << "5C6 - Amit Singhal (11614802722)" << endl;

### vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };

### cout << "Unsorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### clock\_t start = clock();

### insertionSort(arr);

### clock\_t end = clock();

### cout << "Insertion Sort:" << endl;

### cout << "Sorted Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### double time\_taken\_ms = double(end - start) \* 1000.0

### / CLOCKS\_PER\_SEC; // Convert to milliseconds

### cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;

### return 0;

### }

### 

Output ::

Output ::

**Lab Exercise - 2**

* AIM ::

WAP in C++ to implement Linear & Binary Search and also evaluate time in each.

**1. Linear Search**

### #include <ctime>

Source\_Code ::

### #include <iostream>

### #include <vector>

### using namespace std;

### int linearSearch(const vector<int>& arr, int x)

### {

### for (int i = 0; i < arr.size(); i++) {

### if (arr[i] == x) {

### return i;

### }

### }

### return -1; // Element not found

### }

### int main()

### {

### cout << "5C6 - Amit Singhal (11614802722)" << endl;

### vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };

### cout << "Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### int x;

### cout << "Enter the element to search: ";

### cin >> x;

### clock\_t start = clock();

### int index = linearSearch(arr, x);

### clock\_t end = clock();

### if (index != -1) {

### cout << "Element found at index: " << index << endl;

### } else {

### cout << "Element not found" << endl;

### }

### double time\_taken\_ms = double(end - start) \* 1000.0

### / CLOCKS\_PER\_SEC; // Convert to milliseconds

### cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;

### return 0;

### }

Output ::

### 

**2. Binary Search**

### #include <algorithm>

Source\_Code ::

### #include <ctime>

### #include <iostream>

### #include <vector>

### using namespace std;

### int binarySearch(const vector<int>& arr, int x)

### {

### int left = 0, right = arr.size() - 1;

### while (left <= right) {

### int mid = left + (right - left) / 2;

### if (arr[mid] == x) {

### return mid;

### }

### if (arr[mid] < x) {

### left = mid + 1;

### } else {

### right = mid - 1;

### }

### }

### return -1; // Element not found

### }

### int main()

### {

### cout << "5C6 - Amit Singhal (11614802722)" << endl;

### vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };

### sort(arr.begin(), arr.end()); // Binary search requires a sorted array

### cout << "Array: ";

### for (int num : arr) {

### cout << num << " ";

### }

### cout << endl;

### int x;

### cout << "Enter the element to search: ";

### cin >> x;

### clock\_t start = clock();

### int index = binarySearch(arr, x);

### clock\_t end = clock();

### if (index != -1) {

### cout << "Element found at index: " << index << endl;

### } else {

### cout << "Element not found" << endl;

### }

### double time\_taken\_ms = double(end - start) \* 1000.0

### / CLOCKS\_PER\_SEC; // Convert to milliseconds

### cout << "Time taken: " << time\_taken\_ms << " milliseconds" << endl;

### return 0;

### }

Output ::

### 

**Lab Exercise - 3**

* AIM ::

WAP in C++ to implement Huffman Coding & also evaluate its time complexity.

#include <ctime>

Source\_Code ::

#include <iomanip>

#include <iostream>

#include <queue>

#include <unordered\_map>

#include <vector>

using namespace std;

// Node of Huffman Tree

struct Node {

char ch;

int freq;

Node \*left, \*right;

Node(char ch, int freq, Node\* left = nullptr, Node\* right = nullptr)

{

this->ch = ch;

this->freq = freq;

this->left = left;

this->right = right;

}

};

// Comparison function for priority queue

struct compare {

bool operator()(Node\* left, Node\* right)

{

return left->freq > right->freq;

}

};

// Function to build the Huffman Tree

Node\* buildHuffmanTree(const unordered\_map<char, int>& freq)

{

priority\_queue<Node\*, vector<Node\*>, compare> pq;

for (auto pair : freq) {

pq.push(new Node(pair.first, pair.second));

}

while (pq.size() != 1) {

Node\* left = pq.top();

pq.pop();

Node\* right = pq.top();

pq.pop();

int sum = left->freq + right->freq;

pq.push(new Node('\0', sum, left, right));

}

return pq.top();

}

// Function to encode the input string

void encode(

Node\* root, const string& str, unordered\_map<char, string>& huffmanCode)

{

if (root == nullptr)

return;

if (!root->left && !root->right) {

huffmanCode[root->ch] = str;

}

encode(root->left, str + "0", huffmanCode);

encode(root->right, str + "1", huffmanCode);

}

// Function to decode the encoded string

string decode(Node\* root, const string& str)

{

string result = "";

Node\* curr = root;

for (char bit : str) {

if (bit == '0') {

curr = curr->left;

} else {

curr = curr->right;

}

if (!curr->left && !curr->right) {

result += curr->ch;

curr = root;

}

}

return result;

}

int main()

{

cout << "\n5C6 - Amit Singhal (11614802722)" << endl;

string text;

cout << "\nEnter the text to encode: ";

getline(cin, text);

unordered\_map<char, int> freq;

for (char ch : text) {

freq[ch]++;

}

clock\_t start = clock();

Node\* root = buildHuffmanTree(freq);

clock\_t end = clock();

double time\_taken\_build\_tree

= double(end - start) \* 1000.0 / CLOCKS\_PER\_SEC;

unordered\_map<char, string> huffmanCode;

start = clock();

encode(root, "", huffmanCode);

end = clock();

double time\_taken\_encoding = double(end - start) \* 1000.0 / CLOCKS\_PER\_SEC;

cout << "\nCharacter Encoding Table:" << endl;

cout << "------------------------------" << endl;

cout << setw(10) << "Character" << setw(20) << "Huffman Code" << endl;

cout << "------------------------------" << endl;

for (auto pair : huffmanCode) {

cout << setw(10) << pair.first << setw(20) << pair.second << endl;

}

cout << "------------------------------" << endl;

cout << "Time taken to build Huffman Tree: " << time\_taken\_build\_tree

<< " milliseconds" << endl;

string encodedString = "";

for (char ch : text) {

encodedString += huffmanCode[ch];

}

cout << "\nEncoded String: " << encodedString << endl;

cout << "Time taken for encoding: " << time\_taken\_encoding

<< " milliseconds" << endl;

start = clock();

string decodedString = decode(root, encodedString);

end = clock();

double time\_taken\_decoding = double(end - start) \* 1000.0 / CLOCKS\_PER\_SEC;

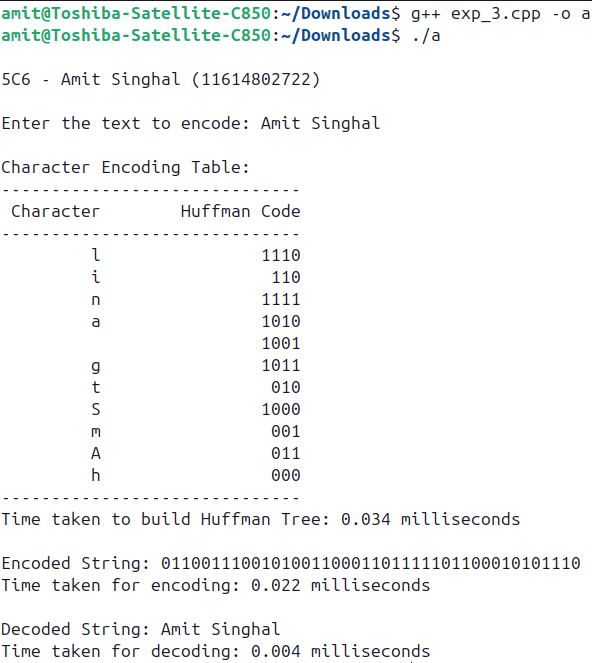
cout << "\nDecoded String: " << decodedString << endl;

cout << "Time taken for decoding: " << time\_taken\_decoding

<< " milliseconds" << endl;

return 0;

}



Output ::