**Diya Goyal 102215255 2NC11**

**Lab Assignment 6**

**UCS 406 Data Structures and Algorithms**

Note: Use C/C++ or JAVA programming language.

Q1. Write a program to perform sorting of a given array using the following algorithms:

• Bubble sort

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Bubble sort

Acknowledgement: NA

\*/

#include<iostream>

using namespace std;

void printArray(int arr[],int n){

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

{

cout<<arr[i]<<" ";

}

}

void BUbbleSort(int arr[],int n){

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

{

bool swapped = false;

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

{

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

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

swapped= true;

}

}

if(swapped== false){

//already sorted

break;

}

}

}

int main(){

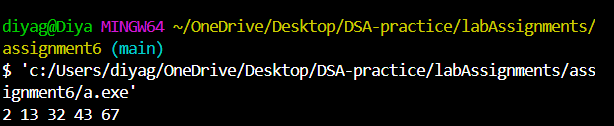
int arr[5]={13, 2, 67,43,32};

BUbbleSort(arr,5);

printArray(arr,5);

return 0;

}



Insertion Sort using arrays

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Insertion sort using Array

Acknowledgement: NA

\*/

#include<iostream>

using namespace std;

void printArray(int arr[],int n){

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

{

cout<<arr[i]<<" ";

}

}

void InsertionSort(int arr[], int n){

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

int temp = arr[i];

int j = i-1;

for (int j=n; j>=0; j--)

{

if(arr[j]>temp){

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

}

else{

break;

}

}

arr[j+1]= temp;

}

}

int main(){

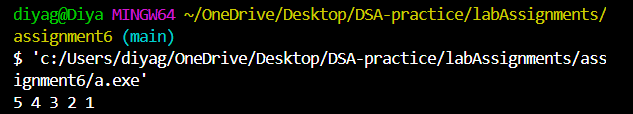
int arr[5]={5,4,3,2,1};

InsertionSort(arr,5);

printArray(arr,5);

return 0;

}



Inesertion Sort using Linked list/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Insertion sort using Linked List

Acknowledgement: GeeksForGeeks

\*/

// C++ program to sort link list

// using insertion sort

#include <bits/stdc++.h>

using namespace std;

struct Node {

int val;

struct Node\* next;

Node(int x)

{

val = x;

next = NULL;

}

};

class LinkedlistIS {

public:

Node\* head;

Node\* sorted;

void push(int val)

{

Node\* newnode = new Node(val);

newnode->next = head;

head = newnode;

}

void insertionSort(Node\* headref)

{

sorted = NULL;

Node\* current = headref;

while (current != NULL) {

Node\* next = current->next;

sortedInsert(current);

current = next;

}

head = sorted;

}

void sortedInsert(Node\* newnode)

{

if (sorted == NULL || sorted->val >= newnode->val) {

newnode->next = sorted;

sorted = newnode;

}

else {

Node\* current = sorted;

while (current->next != NULL

&& current->next->val < newnode->val) {

current = current->next;

}

newnode->next = current->next;

current->next = newnode;

}

}

void printlist(Node\* head)

{

while (head != NULL) {

cout << head->val << " ";

head = head->next;

}

}

};

int main()

{

LinkedlistIS list;

list.head = NULL;

list.push(5);

list.push(20);

list.push(4);

list.push(3);

list.push(30);

cout << "Linked List before sorting" << endl;

list.printlist(list.head);

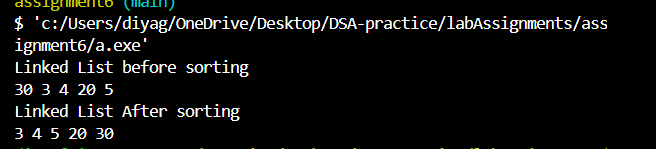
cout << endl;

list.insertionSort(list.head);

cout << "Linked List After sorting" << endl;

list.printlist(list.head);

}



• Selection Sort

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Selection sort

Acknowledgement: NA

\*/

#include<iostream>

using namespace std;

void printArray(int arr[], int n){

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

{

cout<<arr[i]<<" ";

}

}

void SelectionSort(int arr[],int n){

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

int minIndex = i;

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

{

if(arr[j]<arr[minIndex]){

minIndex=j;

}

}

swap(arr[minIndex],arr[i]);

}

}

int main(){

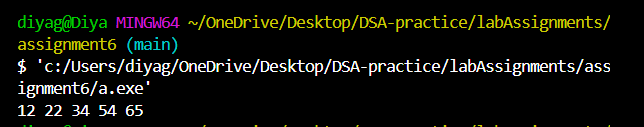
int arr[5]={22,34,54,65,12};

SelectionSort(arr, 5);

printArray(arr,5);

return 0;

}



• Quick Sort

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Quick Sort

Acknowledgement: NA

\*/

#include<iostream>

using namespace std;

int partition(int arr[], int s, int e){

int pivot= arr[s];

int cnt=0;

for (int i = s+1; i <=e; i++)

{

if(arr[i]<=pivot){

cnt++;

}

}

int pivotIndex= s+cnt;

swap(arr[pivotIndex], arr[s]);

int i=s;

int j=e;

while (i<pivotIndex && j>pivotIndex)

{

while (arr[i]<pivot)

{

i++;

}

while (arr[j]>pivot)

{

j--;

}

if (i<pivotIndex && j>pivotIndex)

{

swap(arr[i++], arr[j--]);

}

}

return pivotIndex;

}

void quickSort(int arr[], int s , int e){

//base case

if(s>=e){

return;

}

int p= partition(arr, s,e);

quickSort(arr,s,p-1);

quickSort(arr,p+1,e);

}

int main(){

int arr[5]={2,1,5,4,3};

int n=5;

quickSort(arr,0 ,n-1);

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

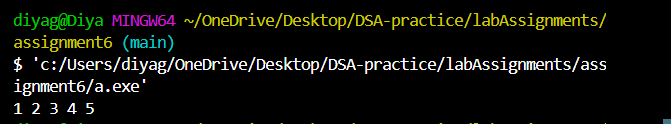
{

cout<<arr[i]<<" ";

}

return 0;

}



• MergeSort

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Merge Sort

Acknowledgement: NA

\*/

#include<iostream>

using namespace std;

void merge(int \*arr, int s, int e){

int mid= (e+s)/2;

int len1=mid-s+1;

int len2= e-mid;

int \*first= new int[len1];

int \*second= new int[len2];

int mainArrayIndex=s;

for (int i = 0; i < len1; i++)

{

first[i]= arr[mainArrayIndex++];

}

mainArrayIndex=mid+1;

for (int i = 0; i < len2; i++)

{

second[i]= arr[mainArrayIndex++];

}

//merge

int index1=0;

int index2=0;

mainArrayIndex=s;

while (index1<len1 && index2<len2)

{

if(first[index1]<second[index2]){

arr[mainArrayIndex++]= first[index1++];

}

else{

arr[mainArrayIndex++]= second[index2++];

}

}

while (index1<len1)

{

arr[mainArrayIndex++]= first[index1++];

}

while (index2<len2)

{

arr[mainArrayIndex++]= second[index2++];

}

delete []first;

delete []second;

}

void mergeSort(int \*arr, int s, int e){

if(s>=e)

{

return;

}

int mid=(e+s)/2;

mergeSort(arr,s,mid);

mergeSort(arr,mid+1,e);

merge(arr,s,e);

}

int main(){

int arr[5]={1,5,6,3,2};

int n=5;

mergeSort(arr,0,n-1);

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

{

cout<<arr[i]<<" ";

}

cout<<endl;

return 0;

}



• Shell Sort

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Shell sort

Acknowledgement: took help from geeksForGeeks

\*/

#include <iostream>

using namespace std;

int shellSort(int arr[], int n)

{

for (int gap = n/2; gap > 0; gap /= 2)

{

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

{

int temp = arr[i];

int j;

for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)

arr[j] = arr[j - gap];

arr[j] = temp;

}

}

return 0;

}

void printArray(int arr[], int n)

{

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

cout << arr[i] << " ";

}

int main()

{

int arr[] = {12, 34, 54, 2, 3}, i;

int n = sizeof(arr)/sizeof(arr[0]);

cout << "Array before sorting: "<<endl;;

printArray(arr, n);

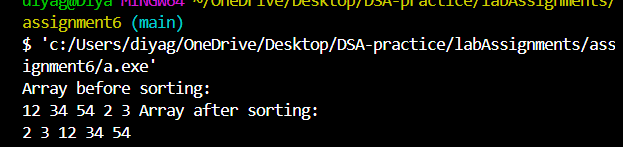
shellSort(arr, n);

cout << "Array after sorting: "<<endl;;

printArray(arr, n);

return 0;

}



• Count Sort

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Count Sort

Acknowledgement: GeeksForGeeks

\*/

#include <bits/stdc++.h>

using namespace std;

vector<int> countSort(vector<int>& inputArray)

{

int N = inputArray.size();

int M = 0;

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

M = max(M, inputArray[i]);

vector<int> countArray(M + 1, 0);

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

countArray[inputArray[i]]++;

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

countArray[i] += countArray[i - 1];

vector<int> outputArray(N);

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

{

outputArray[countArray[inputArray[i]] - 1]

= inputArray[i];

countArray[inputArray[i]]--;

}

return outputArray;

}

int main()

{

vector<int> inputArray = { 4, 3, 12, 1, 5, 5, 3, 9 };

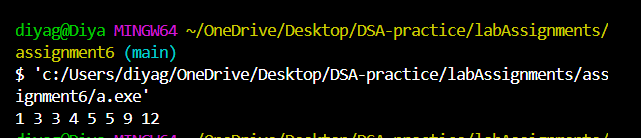
vector<int> outputArray = countSort(inputArray);

for (int i = 0; i < inputArray.size(); i++)

cout << outputArray[i] << " ";

return 0;

}



• Radix Sort

/\*

Roll Number: 102215255

Name: Diya Goyal

Description: Write a program to perform sorting of a given array using the following algorithms:

• Radix Sort

Acknowledgement: GeeksForGeeks

\*/

#include <iostream>

using namespace std;

int getMax(int arr[], int n)

{

int mx = arr[0];

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

if (arr[i] > mx)

mx = arr[i];

return mx;

}

void countSort(int arr[], int n, int exp)

{

int output[n];

int i, count[10] = { 0 };

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

count[(arr[i] / exp) % 10]++;

for (i = 1; i < 10; i++)

count[i] += count[i - 1];

for (i = n - 1; i >= 0; i--) {

output[count[(arr[i] / exp) % 10] - 1] = arr[i];

count[(arr[i] / exp) % 10]--;

}

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

arr[i] = output[i];

}

void radixsort(int arr[], int n)

{

int m = getMax(arr, n);

for (int exp = 1; m / exp > 0; exp \*= 10)

countSort(arr, n, exp);

}

void print(int arr[], int n)

{

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

cout << arr[i] << " ";

}

int main()

{

int arr[] = { 543, 986, 217, 765, 329 };

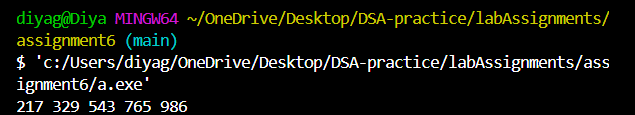
int n = sizeof(arr) / sizeof(arr[0]);

radixsort(arr, n);

print(arr, n);

return 0;

}



Q2

Write a program to perform Quick sort on an integer array of size 100000 whose elements are

chosen randomly. You may choose pivot as any element. Use time function (in your respective

languages) to note down the run time of your quick sort implementation for following cases and

compare them:

a) Take the random elements from range [1,100].

b) Take the random elements from range [1,10000000].

Discuss the disparity in the execution run times.

Hint for random number generation [C++]:

Libraries:

#include &lt;cstdlib&gt;

#include &lt;time.h&gt;

Function calls:

srand(time(0));

x=rand()%100+1;

Hint for runtime computation [C++]:

Libraries:

#include &lt;chrono&gt;

using namespace std::chrono;

Function calls:

// Get starting timepoint

auto start = high\_resolution\_clock::now();

// Call the function, here quicksort()

quicksort(arr,0,n-1);

// Get ending timepoint

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast&lt;microseconds&gt;(stop - start);

cout &lt;&lt; &quot;Time taken by function: &quot;

&lt;&lt; duration.count() &lt;&lt; &quot; microseconds&quot; &lt;&lt; endl;

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <chrono>

using namespace std;

using namespace std::chrono;

void swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

int partition(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(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);

}

}

void randomArray(int arr[], int n, int lower, int upper) {

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

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

}

}

int main() {

srand(time(0));

int n = 100000;

int arr1[n];

int arr2[n];

randomArray(arr1, n, 1, 100);

auto start1 = high\_resolution\_clock::now();

quickSort(arr1, 0, n - 1);

auto stop1 = high\_resolution\_clock::now();

auto duration1 = duration\_cast<microseconds>(stop1 - start1);

cout << "Time taken by Quick Sort for range [1,100]: "

<< duration1.count() << " microseconds" << endl;

randomArray(arr2, n, 1, 10000000);

auto start2 = high\_resolution\_clock::now();

quickSort(arr2, 0, n - 1);

auto stop2 = high\_resolution\_clock::now();

auto duration2 = duration\_cast<microseconds>(stop2 - start2);

cout << "Time taken by Quick Sort for range [1,10000000]: "

<< duration2.count() << " microseconds" << endl;

return 0;

}