**Muhammad Asad Ali**

**Reg # 34248**

**Pointers:**

**1D Array**

#include <iostream>

using namespace std;

int main()

{

int\* Ap= new int[5];

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

Ap[i] = 100 \* (i + 1);

}

cout << \*Ap << endl;

cout << \*Ap + 1 << endl;

cout << 2 [Ap] << endl;

cout << Ap[2] << endl;

cout << \*Ap;

return 0;

}

**2D Array**

#include <iostream>

using namespace std;

int main()

{

int\*\* pointer = new int\*[8];

int x = 1;

for (int i = 0; i < 8; i++) {

pointer[i] = new int[8];

for (int j = 0; j < 8; j++, x++) {

pointer[i][j] = 2+x;

}

}

cout << \*pointer << endl;

cout << \*\*pointer<< endl;

cout << \*pointer + 1 << endl;

cout << \*\*pointer + 1 << endl;

cout << pointer[2][2] << endl;

return 0;

}

**Searching Algorithm**

#include <iostream>

using namespace std;

int main()

{

int arr[] = { 1, 2, 3, 4, 5, 6, 7, 9 };

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

int no = getMissingNumber(arr, n);

cout << no;

}

int getMissingNumber(int a[], int n)

{

int total = (n + 1) \* (n + 2) / 2;

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

total -= a[i];

return total;

}

**Linear Algorithm :**

#include <iostream>

using namespace std;

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

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

int res = searching(arr, n, x);

if (res == -1)

{

cout << "Not Present ";

}else {

cout << "Present " << res;

}

return 0;

}

int searching(int arr[], int n, int x)

{

int i;

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

if (arr[i] == x)

return i;

return -1;

}

**Recursive:**

#include <iostream>

using namespace std;

int main()

{

int n;

cin >> n;

int a[n];

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

cin >> a[i];

}

int x;

cin >> x;

if(chkNo(a, n, x)) {

cout << "true" << endl;

}

else {

cout << "false" << endl;

}

}

bool chkNo(int arr[], int size, int x)

{

if(size==0)

return false;

else{

if(arr[size]==x)

return true;

else{

chkNo(arr,size-1,x);

}

}

}

**Exponential:**

#include <iostream>

using namespace std;

int binarySearching(int arr[], int, int, int);

int main()

{

int arr[] = {2, 3, 4, 10, 40};

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

int x1 = 30;

int res = expoSearching(arr, n, x1);

if(res == -1){ cout <<"Not Present";

}else{ cout <<"Is present " << res;

}

return 0;

}

int expoSearching(int arr[], int n, int x)

{

if (arr[0] == x)

return 0;

int i = 1;

while (i < n && arr[i] <= x)

i = i\*2;

return binarySearching(arr, i/2,

min(i, n-1), x);

}

int binarySearching(int arr[], int l, int r, int x)

{

if (r >= l)

{

int mid = l + (r - l)/2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearching(arr, l, mid-1, x);

return binarySearching(arr, mid+1, r, x);

}

return -1;

}

**FActorial:**

#include <iostream>

using namespace std;

int main()

{

int number, in, factorial=1;

cout<<"Enter the Number to find factorial: ";

cin>>number;

for(in=number; in>=1; in--)

factorial = factorial\*in;

cout<<"Factorial = "<<factorial;

cout<<endl;

return 0;

}

**Power:**

#include <iostream>

using namespace std;

int main(){

int base, expo, i, res = 1;

cout << "Enter the base and exponent to find power \n";

cin >> base >> expo;

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

res = res \* base;

}

cout << base << "^" << expo << " = " << res;

return 0;

}

**Binary Search:**

#include <iostream>

using namespace std;

int main(void)

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 50;

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

int res = binarySearching(arr, 0, n - 1, x);

if(res == -1)

{ cout << "Not Present";

}else{ cout << "Is Present " << res;

}

return 0;

}

int binarySearching(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

if (arr[mid] == x)

return mid;

if (arr[mid] > x)

return binarySearching(arr, l, mid - 1, x);

return binarySearching(arr, mid + 1, r, x);

}

return -1;

}

**Analysis Of Algorithm**

int main()

{

int arr[] = { 1, 10, 30, 15 };

int x = 30;

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

cout << x << " Number is present at index "

<< searching(arr, n, x);

return 0;

}

int searching(int arr[], int n, int x)

{

int i;

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

if (arr[i] == x)

return i;

}

return -1;

}

Average case= O(n)

**Bubble Sort:**

#include <iostream>

using namespace std;

int main()

{

int arr[] = {64, 342, 100, 200};

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

printArray(arr, n);

bubbleSort(arr, n);

cout<<"\nSorted array is: \n";

printArray(arr, n);

return 0;

}

void swaping(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void bubbleSort(int arr[], int n)

{

int i, j;

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

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

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

swaping(&arr[j], &arr[j+1]);

}

void printArray(int arr[], int size)

{

int i;

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

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

cout << endl;

}

**Selection Sort:**

#include <iostream>

using namespace std;

int main()

{

int arra[] = {64, 100, 200, 500, 400, 120 ,230, 90, 40, 10};

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

Sort sorty;

sorty. selectionSorting(arra, n);

cout << "Sorted array is: \n";

sorty.printArray(arra, n);

return 0;

}

class Sort{

public:

void swaping(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSorting(int arra[], int n)

{

int i, j, minidex;

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

{

minidex = i;

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

if (arra[j] < arra[minidex])

minidex = j;

swaping(&arra[minidex], &arra[i]);

}

}

void printArray(int arra[], int size)

{

int i;

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

cout << arra[i] << " ";

cout << endl;

}

};

**Insertion Sort:**

#include <iostream>

using namespace std;

int main()

{

int arra[] = { 10, 120, 190, 400, 200, 300, 230, 408, 189, 30, 50};

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

Sort insertSort;

insertSort. insertionSort(arra, n);

insertSort.printArray(arra, n);

return 0;

}

class Sort{

public:

void insertionSort(int arra[], int n)

{

int i, key, j;

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

{

key = arra[i];

j = i - 1;

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

{

arra[j + 1] = arra[j];

j = j - 1;

}

arra[j + 1] = key;

}

}

void printArray(int arra[], int n)

{

int i;

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

cout << arra[i] << " ";

cout << endl;

}

};

**Merge Sort:**

#include <iostream>

using namespace std;

int main()

{

int arra[] = { 12, 11, 13, 5, 6, 7 };

auto arr\_sizes = sizeof(arra) / sizeof(arra[0]);

Sort sorty;

cout << "Given array \n";

sorty. printArray(arra, arr\_sizes);

sorty.mergeSorting(arra, 0, arr\_sizes - 1);

cout << "\nSorted array \n";

sorty.printArray(arra, arr\_sizes);

return 0;

}

class Sort{

public:

void mergeing(int array[], int const left, int const mid, int const right)

{

auto const subArrayOne = mid - left + 1;

auto const subArrayTwo = right - mid;

auto \*leftArray = new int[subArrayOne],

\*rightArray = new int[subArrayTwo];

for (auto i = 0; i < subArrayOne; i++)

leftArray[i] = array[left + i];

for (auto j = 0; j < subArrayTwo; j++)

rightArray[j] = array[mid + 1 + j];

auto indexOfSubArrayOne = 0,

indexOfSubArrayTwo = 0;

int indexOfMergedArray = left;

while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {

if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {

array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

}

else {

array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

}

indexOfMergedArray++;

}

while (indexOfSubArrayOne < subArrayOne) {

array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

}

while (indexOfSubArrayTwo < subArrayTwo) {

array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

}

}

void mergeSorting(int array[], int const begin, int const end)

{

if (begin >= end)

return;

auto mid = begin + (end - begin) / 2;

mergeSorting(array, begin, mid);

mergeSorting(array, mid + 1, end);

mergeing(array, begin, mid, end);

}

void printArray(int A[], int size)

{

for (auto i = 0; i < size; i++)

cout << A[i] << " ";

}

};

**Quick Sort:**

#include <iostream>

using namespace std;

int main()

{

int arra[] = {100, 20, 40, 40, 60, 1003, 400, 520, 520};

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

Sort sorty;

sorty.quickSorting(arra, 0, n - 1);

cout << "Sorted array: \n";

sorty.printArrays(arra,n);

return 0;

}

class Sort{

public:

void swaping(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partitioning (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++;

swaping(&arr[i], &arr[j]);

}

}

swaping(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSorting(int arr[], int low, int high)

{

if (low < high)

{

int pi = partitioning(arr, low, high);

quickSorting(arr, low, pi - 1);

quickSorting(arr, pi + 1, high);

}

}

void printArrays(int arr[], int size)

{

int i;

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

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

cout << endl;

}

};

**Radix Sort:**

#include <iostream>

using namespace std;

int main()

{

int arra[] = { 100, 200, 30, 50, 40, 200};

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

Sort sort;

sort. radixsorting(arra, n);

sort. print(arra, n);

return 0;

}

class Sort{

public:

void countSorting(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 radixsorting(int arr[], int n)

{

int m = getMaximum(arr, n);

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

countSorting(arr, n, exp);

}

int getMaximum(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 print(int arr[], int n)

{

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

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

}

};

**Bin Sort:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main()

{

float arar[]

= { 0.900, 0.88, 0.66, 0.100, 0.6003, 0.500 };

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

Sort sorty;

sorty. bucketSorting(arar, n);

cout << "Sorted array is\n";

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

cout << arar[i] << " ";

return 0;

}

class Sort{

public:

void bucketSorting(float arr[], int n)

{

vector<float> b[n];

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

int bi = n \* arr[i];

b[bi].push\_back(arr[i]);

}

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

sort(b[i].begin(), b[i].end());

int index = 0;

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

for (int j = 0; j < b[i].size(); j++)

arr[index++] = b[i][j];

}

};

**Stack with Queue:**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

struct Queues {

stack<int> s1, s2;

void enQueues(int x)

{

while (!s1.empty()) {

s2.push(s1.top());

s1.pop();

}

s1.push(x);

while (!s2.empty()) {

s1.push(s2.top());

s2.pop();

}

}

int deQueues()

{

if (s1.empty()) {

cout << "Q is Empty";

exit(0);

}

int x = s1.top();

s1.pop();

return x;

}

};

int main()

{

Queues q;

q.enQueues(3);

q.enQueues(4);

q.enQueues(10);

cout << q.deQueues() << '\n';

cout << q.deQueues() << '\n';

cout << q.deQueues() << '\n';

return 0;

}

**Reverse with Stack:**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

stack<char> st;

string ns;

void insertatbottom(char x)

{

if(st.size() == 0)

st.push(x);

else

{

char a = st.top();

st.pop();

insertatbottom(x);

st.push(a);

}

}

void reverses()

{

if(st.size()>0)

{

char x = st.top();

st.pop();

reverses();

insertatbottom(x);

}

}

int main()

{

st.push('4');

st.push('5');

st.push('7');

st.push('1');

cout<<"Original Stack is"<<endl;

cout<<"1"<<" "<<"2"<<" "

<<"3"<<" "<<"4"

<<endl;

reverses();

cout<<"Reversed Stack"

<<endl;

while(!st.empty())

{

char p=st.top();

st.pop();

ns+=p;

}

cout<<ns[3]<<" "<<ns[2]<<" "

<<ns[1]<<" "<<ns[0]<<endl;

return 0;

}