**Code:-**

#include <iostream>

using namespace std;

int binarySearch(int arr[], int low, int high, int x) {

if (low > high) {

return -1;

}

int mid = low + (high - low) / 2;

if (arr[mid] == x) {

return mid;

} else if (arr[mid] < x) {

return binarySearch(arr, mid + 1, high, x);

} else {

return binarySearch(arr, low, mid - 1, x);

}

}

int main() {

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

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

int x;

cin >> x;

int index = binarySearch(arr, 0, sizeof(arr) / sizeof(arr[0]) - 1, x);

if (index != -1) {

cout << "The element is found at index " << index+1 << endl;

} else {

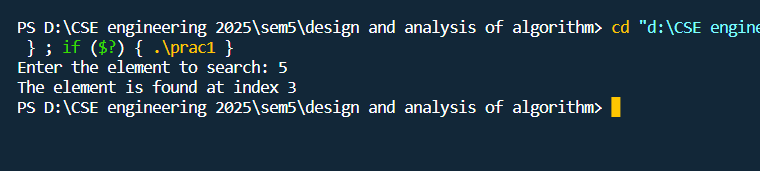
cout << "The element is not found" << endl;

}

return 0;

} 1

**Output:-**

****

1

**Code:- quick sort**

// Quick Sort

#include <iostream>

#include <chrono>

using namespace std;

using namespace chrono;

int partition(int arr[], int start, int end)

{

int pivot = arr[start];

int count = 0;

for (int i = start + 1; i <= end; i++)

{

if (arr[i] <= pivot)

count++;

}

int pivotIndex = start + count;

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

int i = start, j = end;

while (i < pivotIndex && j > pivotIndex)

{

while (arr[i] <= pivot)

{

i++; 2.1

}

while (arr[j] > pivot)

{

j--;

}

if (i < pivotIndex && j > pivotIndex)

{

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

}

}

return pivotIndex;

}

void quickSort(int arr[], int start, int end)

{

if (start >= end)

return;

int p = partition(arr, start, end);

quickSort(arr, start, p - 1);

quickSort(arr, p + 1, end);

}

void display(int \*array, int size)

{ 2.2

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

cout << array[i] << " ";

cout << endl;

}

int main()

{

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter elements:" << endl;

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

{

cin >> arr[i];

}

cout << "Array before Sorting: ";

display(arr, n);

// Start the timer

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

// Sort the array using merge sort

quickSort(arr, 0, n - 1);

// Stop the timer

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

// Get the duration

auto duration = duration\_cast<microseconds>(end - start); 2.3

cout << "Array after Sorting: ";

display(arr, n);

// Print the time taken to sort the array

cout << "Time taken to sort the array using merge sort: " << duration.count() << " microseconds" << endl;

return 0;

}

**Code:- merge sort**

// merge sort

#include <iostream>

#include <chrono>

using namespace std;

using namespace chrono;

void swapping(int &a, int &b)

{

int temp;

temp = a;

a = b;

b = temp;

}

void display(int \*array, int size)

{

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

cout << array[i] << " ";

cout << endl;

} 2.4

void merge(int \*array, int l, int m, int r)

{

int i, j, k, nl, nr;

nl = m - l + 1;

nr = r - m;

int larr[nl], rarr[nr];

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

larr[i] = array[l + i];

for (j = 0; j < nr; j++)

rarr[j] = array[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < nl && j < nr)

{

if (larr[i] <= rarr[j])

{

array[k] = larr[i];

i++;

}

else

{

array[k] = rarr[j];

j++;

}

k++;

}

while (i < nl) 2.5

{

array[k] = larr[i];

i++;

k++;

}

while (j < nr)

{

array[k] = rarr[j];

j++;

k++;

}

}

void mergeSort(int \*array, int l, int r)

{

int m;

if (l < r)

{

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

mergeSort(array, l, m);

mergeSort(array, m + 1, r);

merge(array, l, m, r);

}

}

int main()

{

int n;

cout << "Enter the number of elements: ";

cin >> n;

int arr[n];

cout << "Enter elements:" << endl; 2.6

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

{

cin >> arr[i];

}

cout << "Array before Sorting: ";

display(arr, n);

// Start the timer

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

// Sort the array using merge sort

mergeSort(arr, 0, n - 1);

// Stop the timer

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

// Get the duration

auto duration = duration\_cast<microseconds>(end - start);

mergeSort(arr, 0, n - 1); //(n-1) for last index

cout << "Array after Sorting: ";

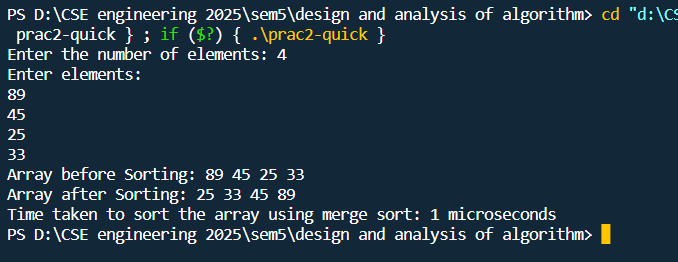
display(arr, n);

// Print the time taken to sort the array

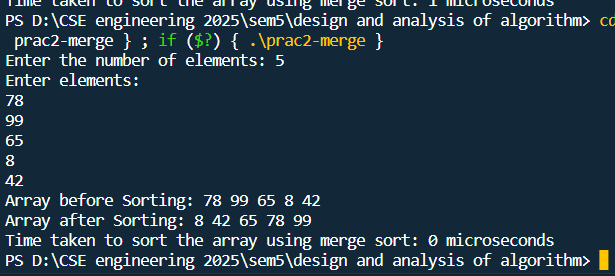
cout << "Time taken to sort the array using merge sort: " << duration.count() << " microseconds" << endl;

}

2.7

**Output:- quick sort**

**Output:- merge sort**



2.8

**Code:-**

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

typedef struct

{

int v;

int w;

float d;

} Item;

void input(Item items[], int sizeOfItems)

{

cout << "Enter total " << sizeOfItems << " item's values and weight" << endl;

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

{

cout << "Enter " << i + 1 << " Value: ";

cin >> items[i].v;

cout << "Enter " << i + 1 << " Weight: ";

cin >> items[i].w;

}

}

void display(Item items[], int sizeOfItems)

{

int i;

cout << "values: ";

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

{

cout << items[i].v << "\t";

}

cout << endl 3.1

<< "weight: ";

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

{

cout << items[i].w << "\t";

}

cout << endl;

}

bool compare(Item i1, Item i2)

{

return (i1.d > i2.d);

}

float knapsack(Item items[], int sizeOfItems, int W)

{

int i, j;

float totalValue = 0, totalWeight = 0;

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

{

items[i].d = (float)items[i].v / items[i].w;

}

sort(items, items + sizeOfItems, compare);

cout << "values : ";

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

{

cout << items[i].v << "\t";

}

cout << endl

<< "weights: ";

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

{

cout << items[i].w << "\t"; 3.2

}

cout << endl

<< "ratio : ";

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

{

cout << items[i].d << "\t";

}

cout << endl;

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

{

if (totalWeight + items[i].w <= W)

{

totalValue += items[i].v;

totalWeight += items[i].w;

}

else

{

int wt = W - totalWeight;

totalValue += (wt \* items[i].d);

totalWeight += wt;

break;

}

}

cout << "Total weight in bag " << totalWeight << endl;

return totalValue;

}

int main()

{

int W;

Item items[4]; 3.3

input(items, 4);

cout << "Entered data \n";

display(items, 4);

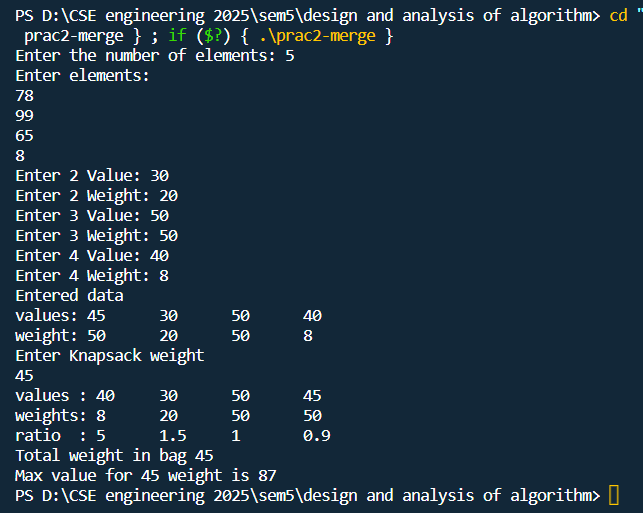
cout << "Enter Knapsack weight \n";

cin >> W;

float mxVal = knapsack(items, 4, W);

cout << "Max value for " << W << " weight is " << mxVal;

}

**Output:-**

3.4