

DESIGN AND ANALYSIS OF ALGORITHMS LAB FILE

Submitted by VINAYAK GOSWAMI (2018UCO1598)



2018UCO1598

COE SECTION-2

# **Binary Search**

CODE:

//BEGIN

#include <iostream>

using namespace std;

int binarySearch(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 binarySearch(arr, l, mid - 1, x);

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

} return -1;}

int main(void)

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

int x = 10;

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

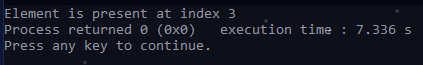
int result = binarySearch(arr, 0, n - 1, x);

(result == -1)? cout << "Element is not present in array": cout << "Element is present at index " << result;

return 0;}

//END

OUTPUT:



# **Insertion Sort**

CODE:

//BEGIN

#include <bits/stdc++.h>

using namespace std;

void insertionSort(int arr[], int n)

{int i, key, j;

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

{ key = arr[i];

j = i - 1;

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

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

j = j - 1;}

arr[j + 1] = key; } }

void printArray(int arr[], int n)

{int i;

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

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

cout << endl;}

int main()

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

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

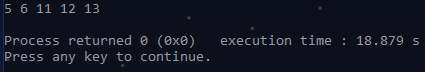
insertionSort(arr, n);

printArray(arr, n);

return 0;}

//END

OUTPUT:



# **Bubble Sort**

CODE:

//BEGIN

#include <bits/stdc++.h>

using namespace std;

void swap(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]) swap(&arr[j], &arr[j+1]);}

void printArray(int arr[], int size)

{ int i;

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

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

cout << endl;}

int main()

{ int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

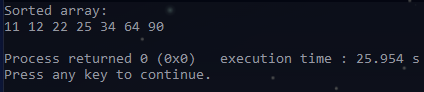
bubbleSort(arr, n);

cout<<"Sorted array: \n"; printArray(arr, n);

return 0;}

//END

OUTPUT:



# **Merge Sort**

CODE:

//BEGIN

#include<stdlib.h>

#include<stdio.h>

void merge(int arr[], int l, int m, int r)

{ int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

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

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

for (j = 0; j < n2; j++)

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

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(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); } }

void printArray(int A[], int size)

{ int i;

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

printf("%d ", A[i]);

printf("\n"); }

int main()

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

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("Given array is \n");

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

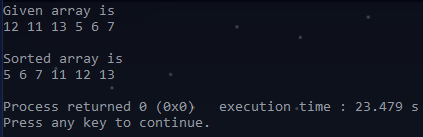
printf("\nSorted array is \n");

printArray(arr, arr\_size);

return 0; }

//END

OUTPUT:



# **Quick Sort**

CODE:

//BEGIN

#include<stdio.h>

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

{ int t = \*a; \*a = \*b; \*b = t; }

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 printArray(int arr[], int size)

{ int i;

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

printf("%d ", arr[i]);

printf("n"); }

int main()

{ int arr[] = {10, 7, 8, 9, 1, 5};

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

quickSort(arr, 0, n-1);

printf("Sorted array: n");

printArray(arr, n);

return 0; }

//END

OUTPUT:



# **Heap Sort**

CODE:

//BEGIN

#include <iostream>

using namespace std;

void heapify(int arr[], int n, int i)

{

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && arr[l] > arr[largest])

largest = l;

if (r < n && arr[r] > arr[largest])

largest = r;

if (largest != i) {

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

heapify(arr, n, largest);

}

}

void heapSort(int arr[], int n)

{

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

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

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n)

{

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

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

cout << "\n";

}

int main()

{

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

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

heapSort(arr, n);

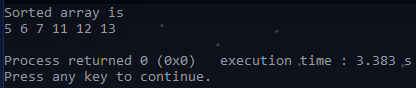
cout << "Sorted array is \n";

printArray(arr, n);

}

//END

OUTPUT:



# **Radix Sort**

CODE:

//BEGIN

#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[] = {170, 45, 75, 90, 802, 24, 2, 66};

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

radixsort(arr, n);

print(arr, n);

return 0;

}//END

OUTPUT:



# **Prims and Kruskal’s**

CODE: (Prims)

//BEGIN

#include <bits/stdc++.h>

using namespace std;

#define V 5

int minKey(int key[], bool mstSet[])

{ int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index; }

void printMST(int parent[], int graph[V][V])

{ cout<<"Edge \tWeight\n";

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

cout<<parent[i]<<" - "<<i<<" \t"<<graph[i][parent[i]]<<" \n"; }

void primMST(int graph[V][V])

{ int parent[V];

int key[V];

bool mstSet[V];

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

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++)

{ int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++)

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];}

printMST(parent, graph); }

int main()

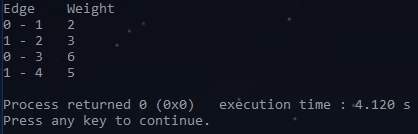
{ int graph[V][V] = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 },{ 0, 3, 0, 0, 7 },{ 6, 8, 0, 0, 9 },{ 0, 5, 7, 9, 0 } };

primMST(graph);

return 0; }

//END

OUTPUT:



CODE: (Kruskal’s)

//BEGIN

#include<bits/stdc++.h>

using namespace std;

typedef pair<int, int> iPair;

struct Graph

{ int V, E;

vector< pair<int, iPair> > edges;

Graph(int V, int E)

{ this->V = V;

this->E = E; }

void addEdge(int u, int v, int w)

{ edges.push\_back({w, {u, v}}); }

int kruskalMST(); };

struct DisjointSets

{ int \*parent, \*rnk; int n;

DisjointSets(int n)

{ this->n = n;

parent = new int[n+1];

rnk = new int[n+1];

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

{ rnk[i] = 0;

parent[i] = i; } }

int find(int u)

{ if (u != parent[u]) parent[u] = find(parent[u]);

return parent[u];}

void merge(int x, int y)

{ x = find(x), y = find(y);

if (rnk[x] > rnk[y]) parent[y] = x;

else parent[x] = y;

if (rnk[x] == rnk[y]) rnk[y]++; } };

int Graph::kruskalMST()

{ int mst\_wt = 0;

sort(edges.begin(), edges.end());

DisjointSets ds(V);

vector< pair<int, iPair> >::iterator it;

for (it=edges.begin(); it!=edges.end(); it++)

{ int u = it->second.first;

int v = it->second.second;

int set\_u = ds.find(u);

int set\_v = ds.find(v);

if (set\_u != set\_v)

{ cout << u << " - " << v << endl;

mst\_wt += it->first;

ds.merge(set\_u, set\_v); } }

return mst\_wt; }

int main()

{ int V = 9, E = 14;

Graph g(V, E);

g.addEdge(0, 1, 4);

g.addEdge(0, 7, 8);

g.addEdge(1, 2, 8);

g.addEdge(1, 7, 11);

g.addEdge(2, 3, 7);

g.addEdge(2, 8, 2);

g.addEdge(2, 5, 4);

g.addEdge(3, 4, 9);

g.addEdge(3, 5, 14);

g.addEdge(4, 5, 10);

g.addEdge(5, 6, 2);

g.addEdge(6, 7, 1);

g.addEdge(6, 8, 6);

g.addEdge(7, 8, 7);

cout << "Edges of MST are \n";

int mst\_wt = g.kruskalMST();

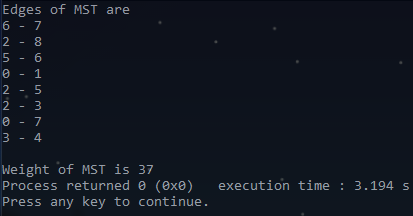
cout << "\nWeight of MST is " << mst\_wt;

return 0;

}

//END

OUTPUT:



# **Dijkstra’s Algorithm**

CODE:

//BEGIN

#include <limits.h>

#include <stdio.h>

#define V 9

int minDistance(int dist[], bool sptSet[])

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

int printSolution(int dist[], int n)

{

printf("Vertex Distance from Source\n");

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

printf("%d tt %d\n", i, dist[i]);

}

void dijkstra(int graph[V][V], int src)

{

int dist[V];

bool sptSet[V];

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

dist[i] = INT\_MAX, sptSet[i] = false;

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX

&& dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist, V);

}

int main()

{

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 }, { 4, 0, 8, 0, 0, 0, 0, 11, 0 }, { 0, 8, 0, 7, 0, 4, 0, 0, 2 }, { 0, 0, 7, 0, 9, 14, 0, 0, 0 }, { 0, 0, 0, 9, 0, 10, 0, 0, 0 }, { 0, 0, 4, 14, 10, 0, 2, 0, 0 }, { 0, 0, 0, 0, 0, 2, 0, 1, 6 }, { 8, 11, 0, 0, 0, 0, 1, 0, 7 }, { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

return 0;

}//END

OUTPUT:

