


AJ's Guide To Algorithms and Data Structures



90+ Algo and DS Chapters
15+ HR questions

Aj's guide to Algorithm and Data Structure – Sample

Introduction

[Chapter 1: Introduction to algorithm and their types](#)

[Chapter 2: Performance analysis of an algorithm: Space Complexity](#)

[Chapter 3: Performance analysis of an algorithm: Time Complexity](#)

[Chapter 4: Asymptotic Notations](#)

[Chapter 5: Asymptotic Notation Big O](#)

[Chapter 6: Asymptotic Notation Big Omega and Theta](#)

Sorting Algorithms:

[Sorting Algorithm 1: Bubble sort](#)

[Sorting Algorithm 2: Selection Sort](#)

[Sorting Algorithm 3: Insertion Sort](#)

[Sorting Algorithm 4: Merge Sort](#)

[Sorting Algorithm 5: Quick Sort](#)

[Sorting Algorithm 6: Pigeonhole Sort](#)

[Sorting Algorithm 7: 3-Way Quicksort \(Dutch National Flag\) algorithm](#)

[Sorting Algorithm 8: Cocktail Sort](#)

[Sorting Algorithm 9: Radix Sort](#)

[Sorting Algorithm 10: Bucket Sort](#)

[Sorting Algorithm 11: Counting Sort](#)

[Sorting Algorithm 12: Shell Sort](#)

[Sorting Algorithm 13: Topological sort](#)

[Sorting Algorithm 14: Comb sort](#)

Searching Algorithm

[Searching Algorithm 1: Linear Search](#)

[Searching Algorithm 2: Binary Search](#)

[Searching Algorithm 3: Jump Search](#)

[Searching Algorithm 4: Interpolation Search](#)

[Searching Algorithm 5: Exponential Search](#)

[Searching Algorithm 6: Ternary Search](#)

Basic Data Structures:

[Data structure tutorial 1: Stack Data structure and Implementation using arrays.](#)

[Data structure tutorial 2: Stack Data structure and Implementation using Linked List.](#)

[Data structure tutorial 3: Singly Linked List.](#)

[Data structure tutorial 4: Doubly Linked List \[DLL\] .](#)

[Data structure tutorial 5: Circular Singly Linked List.](#)

[Data structure tutorial 6: Circular Doubly Linked List.](#)

[Data structure tutorial 7: Queue Data Structure with implementation using arrays.](#)

[Data structure tutorial 8: Queue Data Structure with implementation using linked list.](#)

[Data structure tutorial 9: Circular Queues Data structure with Implementation using arrays.](#)

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

[Data structure tutorial 10: Circular Queue Data structure with Implementation using Linked List.](#)

Trees Data Structure Tutorials:

[Tree data structure tutorial 1. Tree Data Structure Introduction](#)

[Tree data structure tutorial 2. Introduction to Binary Tree](#)

[Tree data structure tutorial 3. Binary Tree Traversal](#)

[Tree data structure tutorial 4. Binary Search Tree Introduction](#)

[Tree data structure tutorial 5. Implementation of BST](#)

[Tree data structure tutorial 6. Implementation of Binary tree](#)

[Tree data structure tutorial 7. TRIE Data structure](#)

[Tree data structure tutorial 8. Heaps](#)

[Tree data structure tutorial 9. Priority Queue](#)

[Tree data structure tutorial 10. AVL tree](#)

[Tree data structure tutorial 11. Introduction to segment trees](#)

[Tree data structure tutorial 12. Performing minimum Range query in Segment Tree and implementation](#)

[Tree data structure tutorial 13. Lazy propagation of segment trees](#)

[Tree data structure tutorial 14. Fenwick trees and implementation](#)

Graph Data Structure Tutorials:

[Graph data structure tutorial 1. Graph Introduction](#)

[Graph data structure tutorial 2. Graph Representation Adjacency Matrix](#)

[Graph data structure tutorial 3. Graph Representation Adjacency List](#)

[Graph data structure tutorial 4. Graph Traversal](#)

[Graph data structure tutorial 5. Graph Traversal using Stack and Queue](#)

[Graph data structure tutorial 6. Bipartite graph](#)

[Graph data structure tutorial 7. Graph coloring problem](#)

[Graph data structure tutorial 8. Isomorphic Graph](#)

[Graph data structure tutorial 9. Euler Graph](#)

[Graph data structure tutorial 10. Hamiltonian Graph](#)

Different types of problem solving technique

[1. Brute force approach](#)

[2. Recursion](#)

[3. Dynamic programming approach](#)

[4. Backtracking approach](#)

[5. Greedy approach](#)

[6. Two pointer approach](#)

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

Minimum Spanning Tree:

[Minimum Spanning Tree tutorial 1. Introduction to minimum spanning tree](#)

[Minimum Spanning Tree tutorial 2. Kruskal's algorithm](#)

[Minimum Spanning Tree tutorial 3. Prims Algorithm](#)

Find shortest path algorithm

[Finding shortest path algorithm tutorial 1. Bellman ford](#)

[Finding shortest path algorithm tutorial 2. Dijkstra's](#)

[Finding shortest path algorithm tutorial 3. Floyd warshalls](#)

String matching algorithms

[String matching algorithms tutorial 1. Knuth Morris Pratt String matching algorithm](#)

[String matching algorithms tutorial 2. Rabin Karp algorithm](#)

[String matching algorithms tutorial 3. Boyer–Moore string-search algorithm](#)

Knapsack Problem:

[1. Fractional knapsack](#)

[2. Knapsack](#)

Additional Problems:

[1. P, NP, NP hard, NP Complete](#)

[2. Tower of Hanoi](#)

[3. Sieve of Eratosthenes](#)

[4. Kadane Algorithm](#)

[5. Sliding Window Approach](#)

[6. Travelling Salesman problem](#)

[7. Minimum Coin Change Problem](#)

[8. Total number of ways to get denomination of coins.](#)

[9. Job Sequencing problem](#)

[10. Activity Selection Problem](#)

[11. House Robber Problem](#)

HR Interview questions and tips to answer them

[1. Expectations on oncoming topics](#)

[2. Mistakes to avoid in an interview.](#)

[3. Tell me about yourself](#)

[4. Why should we hire you?](#)

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

Aj's guide to Algorithm and Data Structure – Sample

- [5. Why do you want to work for us?](#)
- [6. What are your greatest strengths and weakness?](#)
- [7. What are your greatest achievements/ accomplishments?](#)
- [8. Any questions for us?](#)
- [9. Where do you want to see yourself in 5 years?](#)
- [10. How do you work under pressure?](#)
- [11. How do you make important decisions?](#)
- [12. What motivates you to do the best on job?](#)
- [13. Do you prefer working alone or in a team?](#)
- [14. What do you know about our company?](#)
- [15. Are you planning for further studies?](#)
- [16. What is your salary expectations?](#)

Tips for Developers to improve their skills

- [1. How to prepare for coding interview in 3 months.](#)
- [2. Tips to solve coding interview questions](#)
- [3. How to write a resume for coding interview?](#)
- [4. Tips to become good at programming](#)

Sorting algorithm 5: Quick Sort

In this chapter we shall learn about below topics:

- 5.1 Introduction
- 5.2 Steps for performing Quick Sort
- 5.3 Understanding Quick Sort with an example
- 5.4 Implementation of Quick Sort in C
- 5.5 Output of the program
- 5.6 Time complexity analysis of Quick Sort

5.1 Introduction

Quick Sort is a **divide and conquer technique**. Usually we use Quick Sort on a very large data set.

Below is a brief on how this algorithm works:

Divide: Divide the array into 2 arrays based on a pivot element.

Conquer: Sort the left array recursively and right array recursively.

Combine: Combine both sorted left and right array.

To perform Quick sort, we need the help of “pivot element”. Pivot element holds an important role in this algorithm.

5.2 Steps for performing Quick Sort

Step 1: The first step is to get a pivot element.

Step 2: Once we get the pivot element, then we create 2 half such that, the elements left to the pivot element are lesser than pivot element. The elements right to pivot element are greater than that of pivot element.

Step 3: We do Step 1 and Step 2 recursively till the array is sorted.

Steps to select pivot element:

As we discussed earlier that selecting “pivot element” is important step in this algorithm.

Step 1: We shall select lowest index as the pivot element.

Step 2: According to the pivot element, we swap the elements in their respective positions.

5.3 Understanding Quick Sort with an example

In this example, we shall select the lowest element as pivot element.

Aj's guide to Algorithm and Data Structure – Sample

Below are the points to be remembered while doing quick sort:

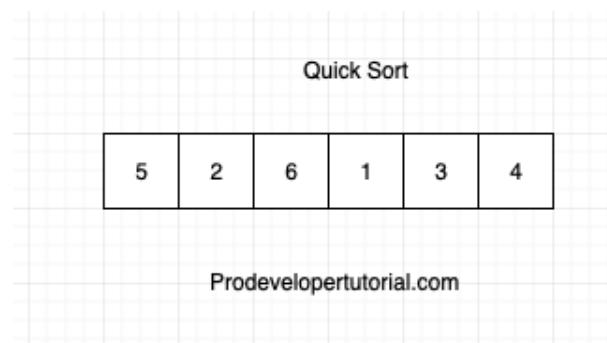
We take 2 variables, one pointing to the left end of the array and another pointing to the right end of the array.

Then while comparing with the left pointer, we check if the element is less than the pivot element. If it is less, we do nothing. If the element is greater than the pivot element we swap the elements.

Similarly, while comparing with the right pointer, we check if the element is greater than the pivot element. If the element is less than the pivot element, we swap the elements.

By doing the above 2 operations, we make sure that all the elements to the left of pivot element are less and elements to right of pivot are greater. [Revisit these points once you go through the example below].

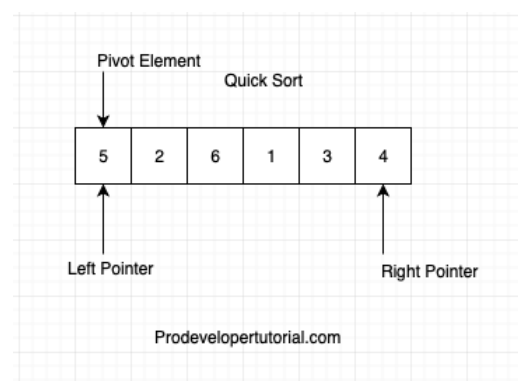
Consider the below array:



Initially the left pointer will be pointing to left part of the array.

Right pointer will be pointing to the right part of the array.

Pivot element also will be pointing to the left part of the array, as shown below:



We know that, all the elements to the right of the pivot element should be greater than the pivot element and all the elements left of the pivot will be lesser than the pivot element.

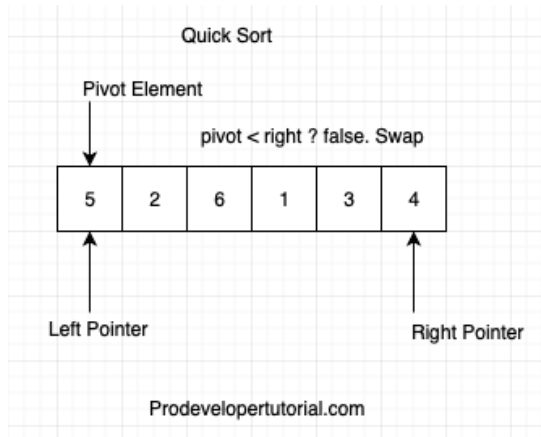
Buy the book at : <https://imojo.in/ajGuideAlgoDS>

Aj's guide to Algorithm and Data Structure – Sample

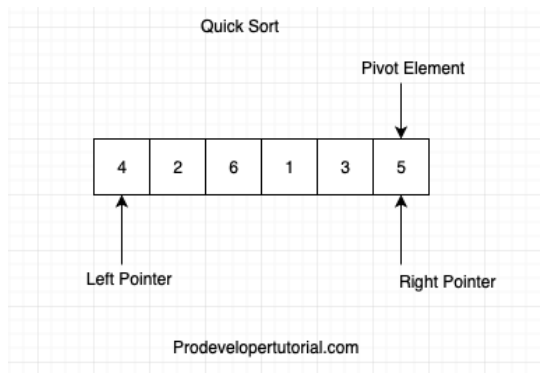
Initially, the pivot is at the left, hence we start comparing from the right and move towards left. To move an element towards left, that element should be less than the pivot element. Hence we check below condition.

First, we check if $\text{arr}[\text{pivot}] < \text{arr}[\text{right}]$? False.

Hence swap pivot and right element.



after swap, the pivot element will reach at the end.

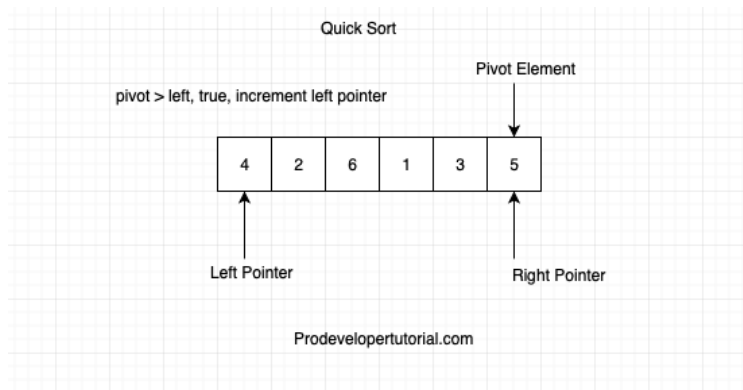


Now that the pivot element is at the right, we start comparing from left and move towards right. For an element to be at right of pivot element, it should be greater than the pivot element. hence we check below condition:

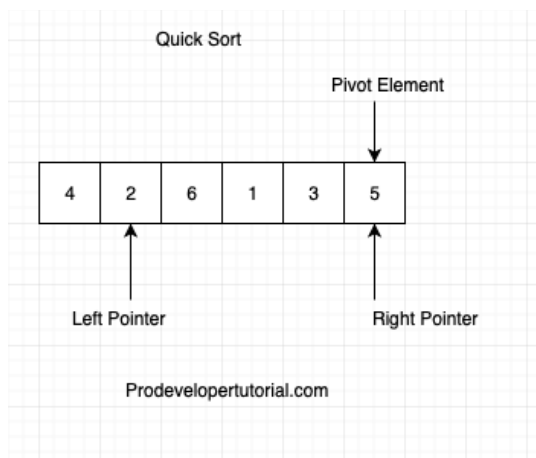
if $\text{arr}[\text{pivot}] > \text{arr}[\text{left}]$? True.

Hence we move the left pointer to the next element as shown below:

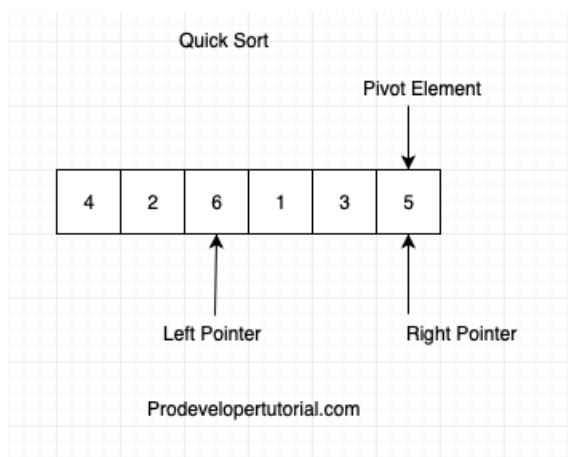
Aj's guide to Algorithm and Data Structure – Sample



after comparing:

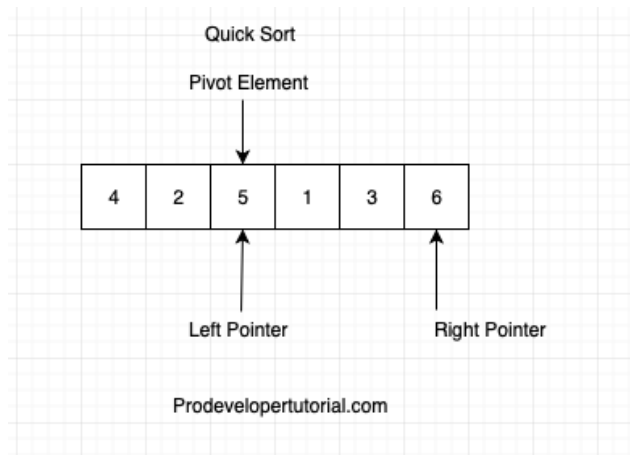


again check if $\text{arr}[\text{pivot}] > \text{arr}[\text{left}]$, if $(5 > 2)$ true. Hence increment left pointer to next index as shown below:

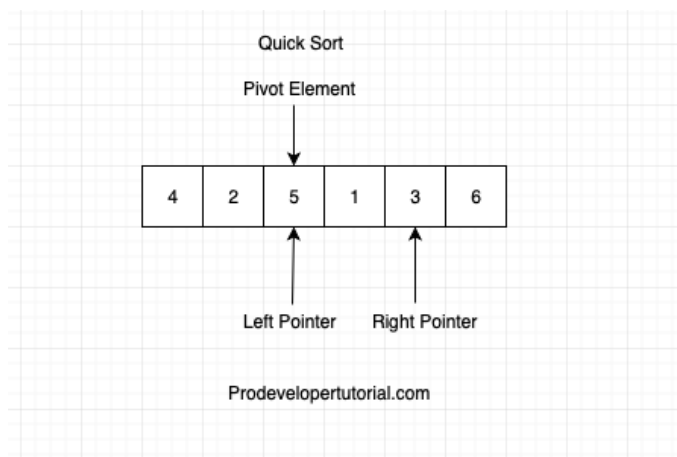


Now again, check if $\text{arr}[\text{pivot}] > \text{arr}[\text{left}]$, if $(5 > 6)$ false. Hence swap element 5 and 6.

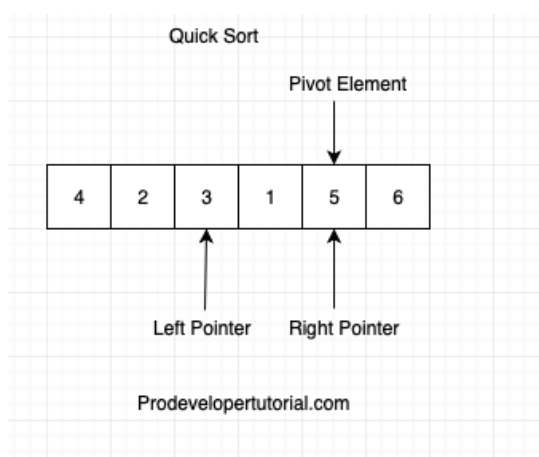
Aj's guide to Algorithm and Data Structure – Sample



Now check if $\text{arr}[\text{pilot}] < \text{arr}[\text{right}]$, if $(5 < 6)$ true. Decrement right pointer as shown below:



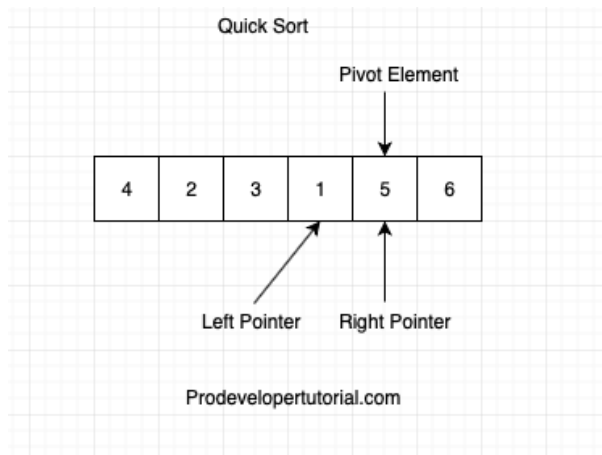
Now again check if $\text{arr}[\text{pilot}] < \text{arr}[\text{right}]$, if $(5 < 3)$ false. Hence swap pivot element and right element as shown below:



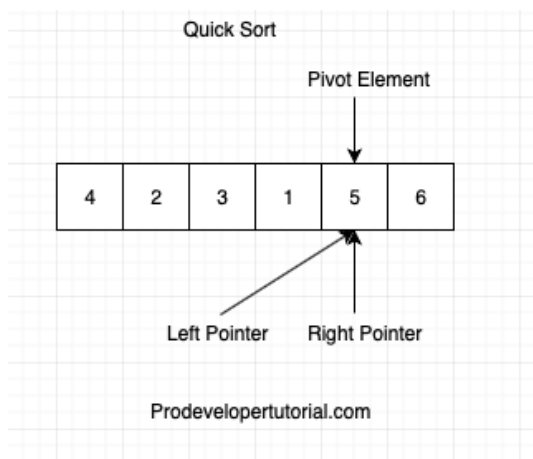
As the pivot element moved from left to right, we start comparing from left and pivot element.

check if $\text{arr}[\text{pilot}] > \text{arr}[\text{left}]$, if $(5 > 3)$ true. Increment left pointer

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

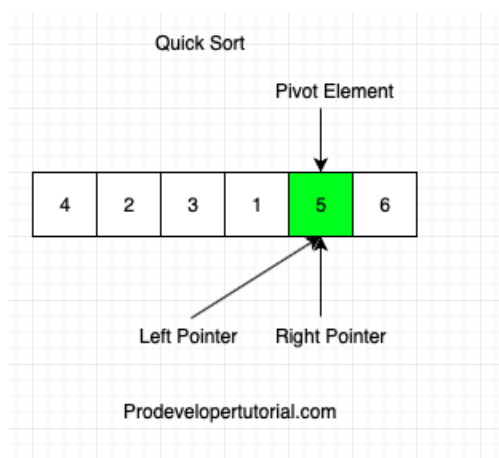


again check if $\text{arr}[\text{pilot}] > \text{arr}[\text{left}]$, if $(5 > 1)$ true. Increment left pointer.



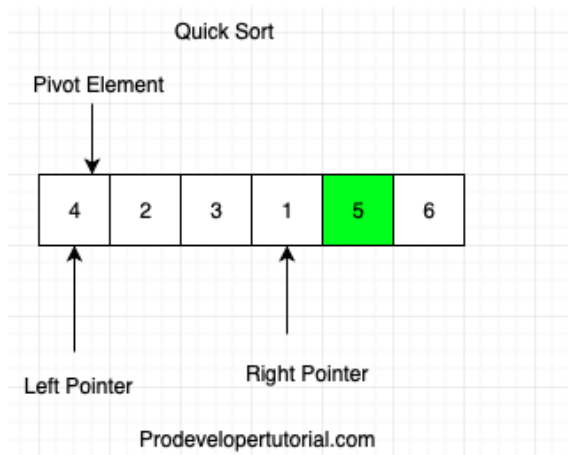
Now that left and right pointer are pointing at the same element of the array, the element 5 is at the pivot element and is in the sorted position.

All the elements left of 5 are smaller and elements to right are larger than 5 as shown below:



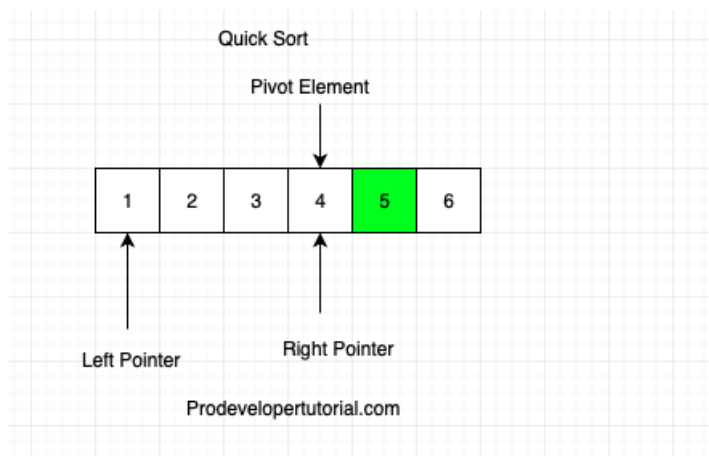
Again we have to sort the left part and right part. Again the initial array will be like below:

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

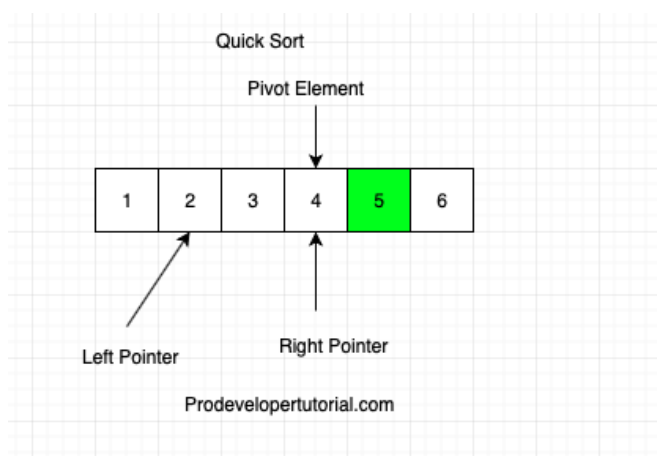


as the pivot is at left, we compare with right pointer and move towards left.

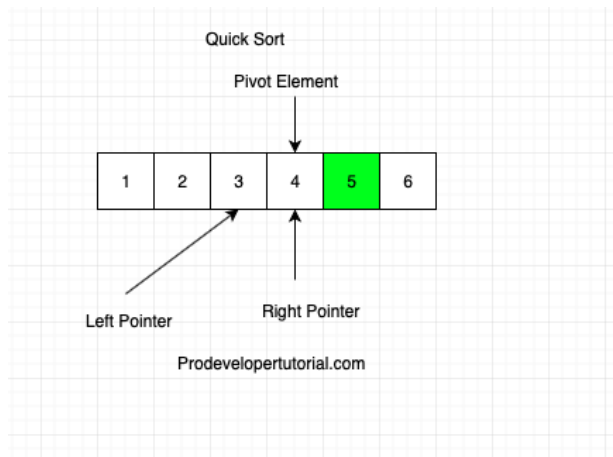
Hence check if $\text{arr}[\text{pilot}] < \text{arr}[\text{right}]$, if $(4 < 1)$ False. Hence swap pivot and right element.



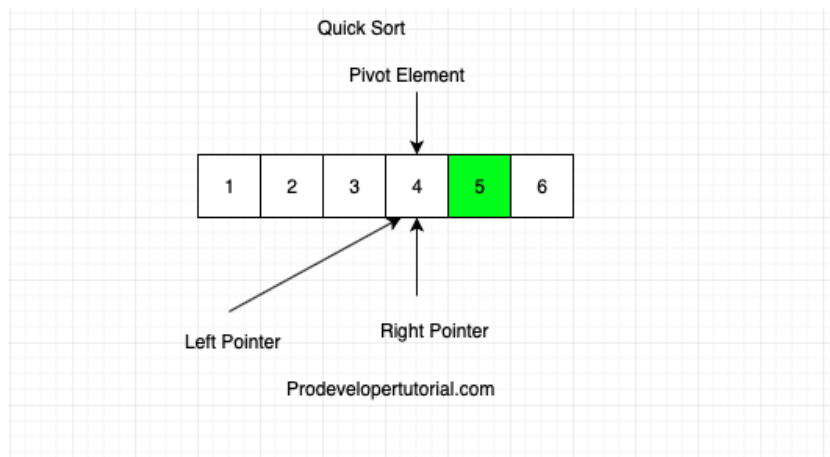
Now check if $\text{arr}[\text{pilot}] > \text{arr}[\text{left}]$, if $(4 > 1)$ true. Increment left pointer.



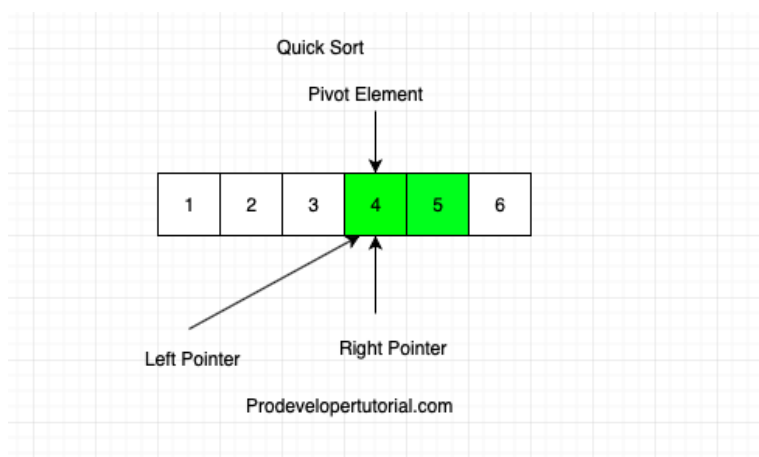
again check if $\text{arr}[\text{pilot}] > \text{arr}[\text{left}]$, if $(4 > 2)$ true. Increment left pointer by 1.



again check if $\text{arr}[\text{pilot}] > \text{arr}[\text{left}]$, if $(4 > 3)$ true. Increment left pointer by 1.



As both left and right pointer points to the same index, it means that element 4 is at it's correct position.



Hence again do the same quick sort for the left sub array "1, 2, 3".

Finally we get our sorted array.

5.4 Implementation of Quick Sort in C

```
#include<stdio.h>

void print_array(int array[], int length)
{
    int index = 0;

    printf("The sorted array is \n");

    for(index = 0 ; index < length ; index++)
    {
        printf("%d\n",array[index] );
    }
}

void swap (int *num_1, int *num_2)
{
    int temp = *num_1;
    *num_1 = *num_2;
    *num_2 = temp;
}

int partition(int array[], int lower_index, int upper_index)
{
    int pivot = lower_index;
    int i = lower_index;
    int j = upper_index;

    while(i < j)
    {
        while(array[i] <= array[pivot] && i < lower_index)
            i++;

        while(array[j] > array[pivot] )
            j--;

        if(i < j)
        {
            swap(&array[i], &array[j]);
        }
    }

    swap(&array[pivot], &array[j]);

    return j;
}
```

```
void quick_sort (int array[], int lower_index, int upper_index)
{
    int partition_index = 0;
    if(lower_index < upper_index)
    {
        partition_index = partition(array, lower_index, upper_index);
        quick_sort(array, lower_index, partition_index - 1);
        quick_sort(array, partition_index + 1, upper_index);
    }
}

int main()
{
    int length = 0;
    int array[100];
    int index = 0;

    printf("Enter the length of the array\n");
    scanf("%d", &length);

    printf("Enter the array elements of length %d\n", length);

    for (index = 0; index < length; ++index)
    {
        scanf("%d", &array[index]);
    }

    quick_sort(array, 0, length-1);

    print_array(array, length);
}
```

5.5 Output of the program

```
Enter the length of the array
5
Enter the array elements of length 5
5
4
3
2
1
The sorted array is
```


1
2
3
4
5

5.6 Time complexity analysis of Quick Sort

In the worst case, the time taken will be $O(n^2)$.

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

Minimum Spanning Tree tutorial 2: Introduction to Kruskal's algorithm

In this chapter we shall learn about below topics:

- 2.1 Introduction to Kruskal's algorithm.
- 2.2 Conditions for Kruskal's algorithm to work.
- 2.3 Working steps of Kruskal's algorithm.
- 2.4 Understanding Kruskal's algorithm with the help of an example.
- 2.5 Implementation of Kruskal's algorithm.

2.1 Introduction to Kruskal's algorithm tree:

Kruskal's algorithm is used to find MST in a graph. It is a greedy based algorithm.

Why do we call it as greedy? Because, as you will see further, we choose the shortest distance first without considering the fact what there might be more optimized path. Hence, we find another path with shortest value, we update the result with that value.

2.2 Below are the conditions for Kruskal's algorithm to work:

- 1. The graph should be connected
- 2. Graph should be undirected.
- 3. Graph should be weighted.

Let us first understand the working of the algorithm, then we shall solve with the help of an example.

2.3 Working steps of Kruskal's algorithm

The working of Kruskal's algorithm is very simple and can be understood by 3 simple steps as given below:

Step 1: Sort all the edges according to their weights.

Step 2: a. Start from the edge with smallest weight and connect them together.

- 1. Before connecting, check if it forms a cycle if it forms a cycle then reject the edge and go for the next smallest weighted edge.

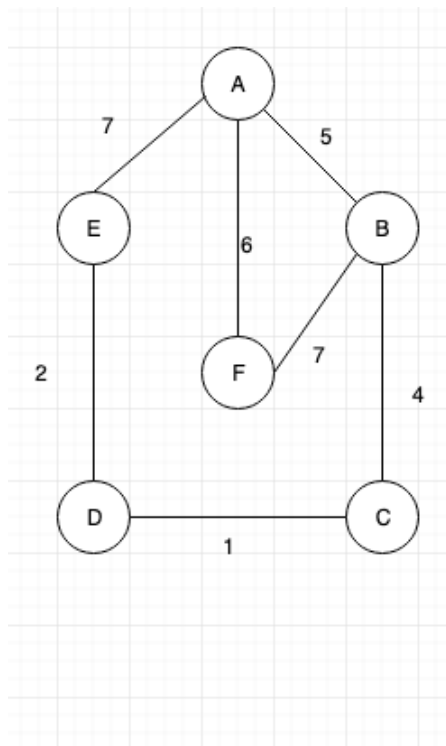
Step 3: Check if all the vertices are connected. If they are not connected, then repeat step 2 till all the vertices are connected.

Once all the tree vertices are connected to an edge, we get MST.

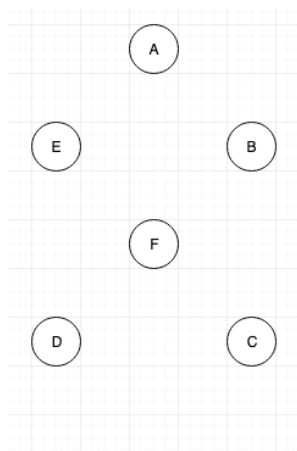
Note: As you see in Prim's algorithm that starts by choosing a root vertex, in Kruskal's algorithm we start connecting 2 vertices by choosing least weighted edges.

2.4 Understanding Kruskal's algorithm with the help of an example

Consider the graph as below

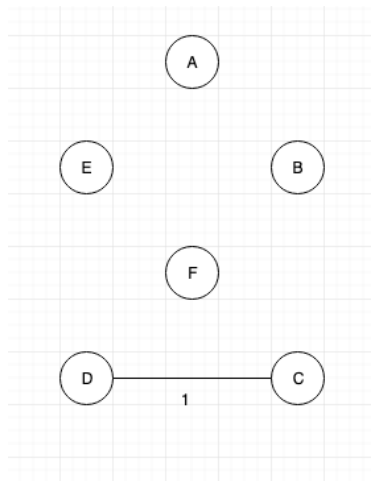


Initial Configuration will be as below:

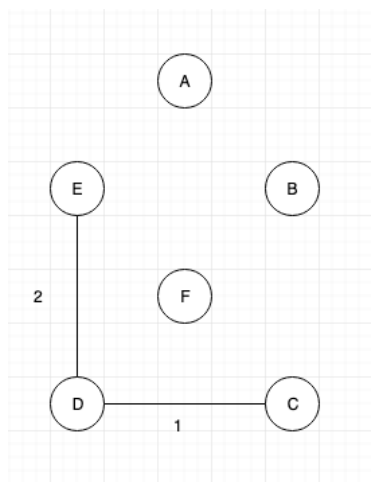


Aj's guide to Algorithm and Data Structure – Sample

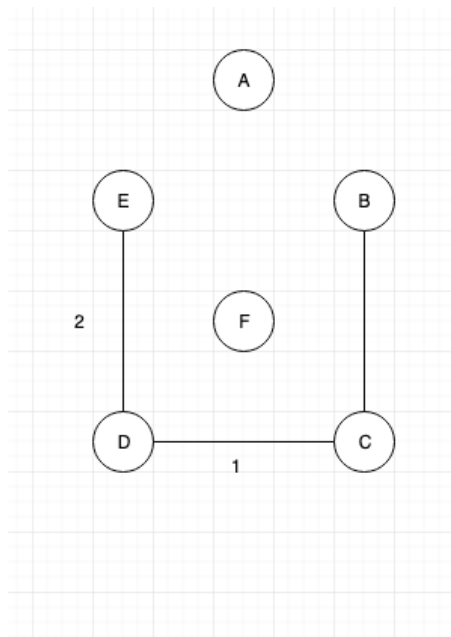
Connect the nodes "C" and "D" as their weight is minimum



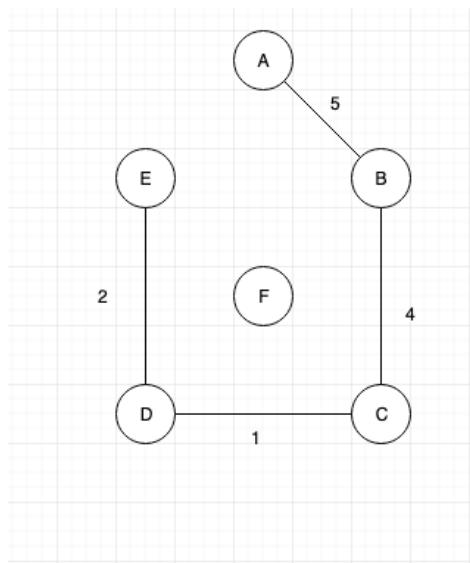
Next connect nodes "D" and "E", as they are next minimum



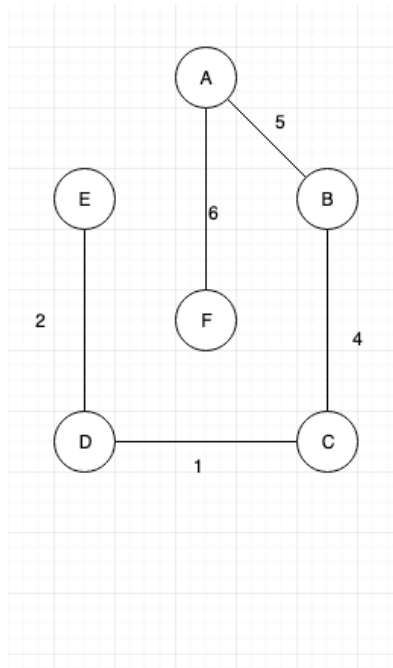
Next connect "C" and "B", as they are next minimum



Next connect "B" and "A"



Next connect "A" and F". Thus connecting all the nodes and forming a MST.



2.5 Implementation of Kruskal's algorithm.

```
#include<cstdio>
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

#define MAX 1000

struct edge
{
    int u, v, w;
};

typedef struct edge E;

int parents[MAX+5];
vector <E> graph; // Store the inputted graph (u, v, w).
vector <E> selected_edges; // Store the edges which is selected for the MST from given graph.

bool comp (const E& l, const E& r)
{

```

```
        return l.w < r.w;
    }

void make_set(int nodes)
{
    for(int i=1; i<=nodes; i++)
        parents[i] = i;

    return;
}

int findParent( int r )
{
    if( r == parents[r] ) return r;

    return parents[r] = findParent( parents[r] );
}

int Kruskal_MST (int nodes)
{
    make_set(nodes);

    sort(graph.begin(), graph.end(), comp);

    int edgeCounter=0, totalCost=0;

    int len = graph.size();

    for(int i=0; i<len; i++)
    {
        int parent_of_u = findParent( graph[i].u );
        int parent_of_v = findParent( graph[i].v );

        if( parent_of_u != parent_of_v )
        {
            parents[ parent_of_u ] = parent_of_v;
            totalCost += graph[i].w;
            selected_edges.push_back( graph[i] );

            edgeCounter++;
            if( edgeCounter == nodes-1 )
```

```
                break;
            }

        }

    return totalCost;
}

int main()
{
    E getEdge;
    int nodes, edges;

    cout<<"How many nodes: "<<endl;
    cin >> nodes;

    cout<<"How many edges: "<<endl;
    cin >> edges;

    cout<<"Enter 2 nodes & their costs (u v & w): \n";

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

        int u, v, w;
        cout<<"Edge "<< i<<endl;
        cin >> u >> v >> w;

        getEdge.u = u;
        getEdge.v = v;
        getEdge.w = w;

        graph.push_back(getEdge);
    }

    int totalCost = Kruskal_MST(nodes);

    cout<<"\\n\\nSelected Edges and their costs: \n";

    for(int i=0; i<selected_edges.size(); i++)
    {
        cout<<"Edge "<< i+1 <<": "<< selected_edges[i].u<< " -> " <<
selected_edges[i].v<< ": cost "<< selected_edges[i].w <<endl;
    }
}
```



```
cout<<"\nTotal Costs: "<< totalCost<<endl;\n\ngetchar();\nreturn 0;\n}
```

Time complexity of Kruskal's algorithm is **$O(\log V)$**

Kruskal's algorithm should be used in typical situations (sparse graphs) because it uses simpler data structures.

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

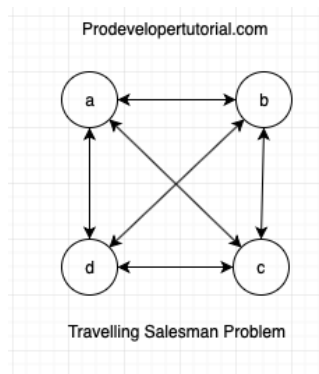
Travelling salesman problem with implementation

In this chapter we shall solve Travelling Salesman Problem with help of dynamic programming.

Problem statement:

A salesman will start from a parent city and visit all the cities only once and return to parent city. He has to do it with least cost possible.

Consider the below graph and let the parent city be “a”.



Now let's write the valid cases where the salesman visits all the cities only once and return to source city.

a -> b -> c -> d -> a

a -> d -> c -> b -> a

Below are invalid scenarios:

a -> b -> d -> c -> b -> a

Here we have visited “b” 2 times. As you can see from the above valid case, we need to find a Hamiltonian cycle.

As there can be many routes/cycles, we need to find a path with min cost.

Below is the cost of adjacency matrix.

Prodevelopertutorial.com				
	A	B	C	D
A	0	14	9	4
B	6	0	11	14
C	2	5	0	7
D	3	10	1	0
TRAVELLING SALESMAN PROBLEM				

So this problem can be solved by 3 different methods:

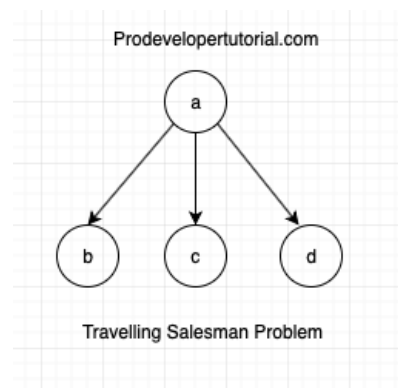
1. Brute force Approach
2. Dynamic Programming
3. Branch and Bound approach

In this tutorial we shall follow dynamic programming approach.

So we shall understand the DP approach with the help of the tree below:

From the given graph, we have chosen "a" as source vertex.

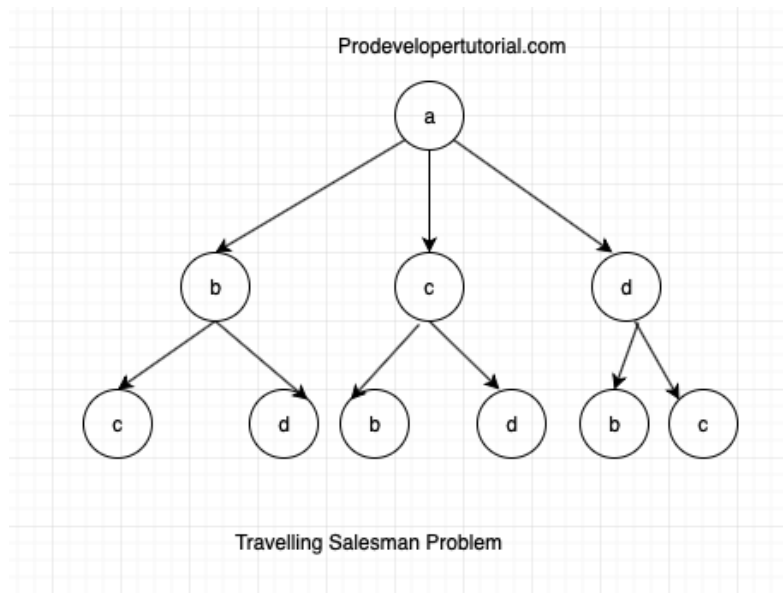
Hence from "a" we can go to "b" "c" and "d". It can be shown as below:



Again from "b" he can go to "d" or "c"

Again from "c" he can go to "b" or "d"

Again from "d" he can go to "b" or "c"



Again from node "c" he go to "c"

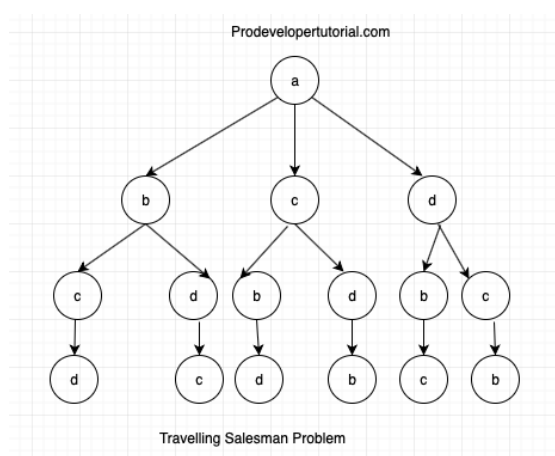
Again from node "d" he go to "c"

Again from node "b" he go to "d"

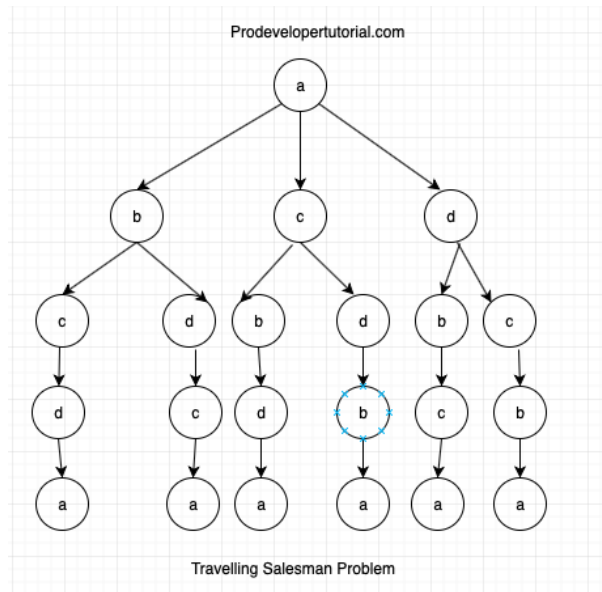
Again from node "d" he go to "b"

Again from node "b" he go to "c"

Again from node "c" he go to "b"



So above image shows all the possible paths. You need to reach to the source vertex again. Hence we can further update the tree as below:



Starting from the last path, we go from bottom to top and update the cost taking from cost table.

So start from "d" to "a", it is 3

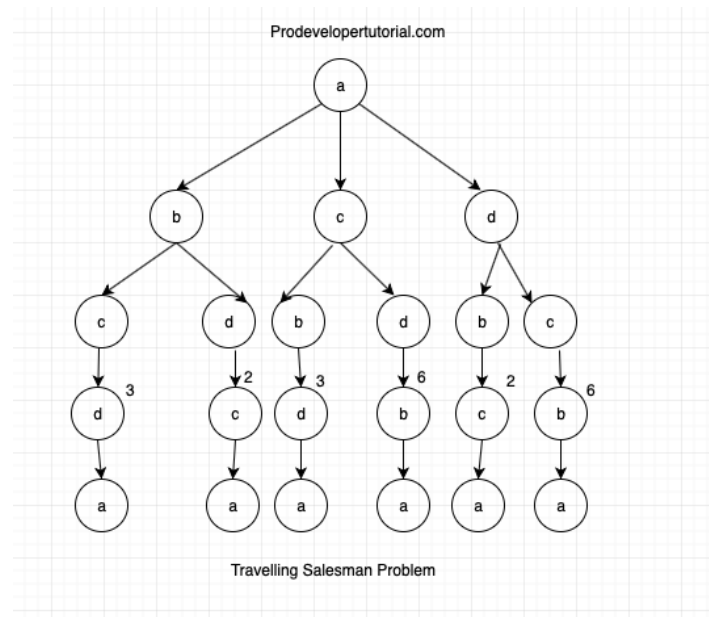
Cost from "c" to "a", it is 2

Cost from "d" to "a", it is 3

Cost from "b" to "a", it is 6

Cost from "c" to "a", it is 2

Cost from "b" to "a", it is 6



Again we shall go one level up and find the total distance:

$c \rightarrow d \rightarrow a \Rightarrow$ i.e $c \rightarrow d + d \rightarrow a \Rightarrow 7 + 3 = 10$

$d \rightarrow c \rightarrow a \Rightarrow d \rightarrow c + c \rightarrow a \Rightarrow 1 + 2 = 3$

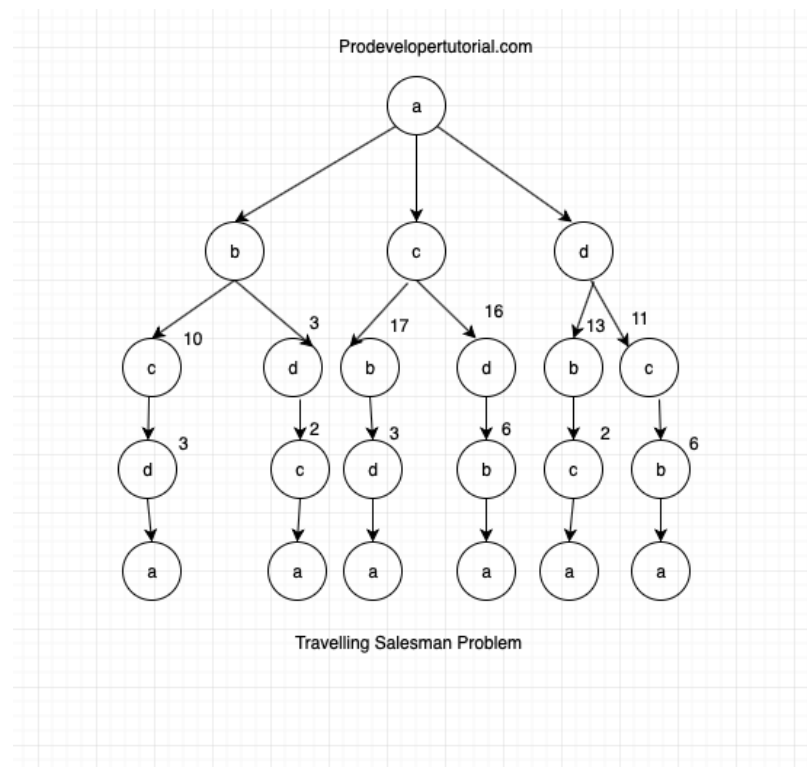
$b \rightarrow d \rightarrow a \Rightarrow b \rightarrow d + d \rightarrow a \Rightarrow 14 + 3 = 17$

$d \rightarrow b \rightarrow a \Rightarrow d \rightarrow b + b \rightarrow a \Rightarrow 10 + 6 = 16$

$b \rightarrow c \rightarrow a \Rightarrow b \rightarrow c + c \rightarrow a \Rightarrow 11 + 2 = 13$

$c \rightarrow b \rightarrow a \Rightarrow c \rightarrow b + b \rightarrow a \Rightarrow 5 + 6 = 11$

The updated tree is as below:



Now again we go one level up, and check the value for “b -> c -> d -> a”, as we have calculated value for “c -> d -> a”, we just need to add that value to the last path. Here with help of DP table we shall be able to do it efficiently.

$$b \rightarrow c + (c \rightarrow d \rightarrow a) = 11 + 10 = 21$$

$$b \rightarrow a + (d \rightarrow c \rightarrow a) = 14 + 3 = 17$$

$$c \rightarrow b + (b \rightarrow d \rightarrow a) = 5 + 17 = 22$$

$$c \rightarrow d + (d \rightarrow b \rightarrow a) = 7 + 16 = 23$$

$$d \rightarrow b + (b \rightarrow c \rightarrow a) = 10 + 13 = 23$$

$$d \rightarrow c + (c \rightarrow b \rightarrow a) = 1 + 11 = 12$$

Now that we have calculated the value for all 6 paths, at the parent node of “b”, “c”, “d”. We have 2 values, we need to find min of 2 values.

Aj's guide to Algorithm and Data Structure – Sample

For node “b” we have:

$$b \rightarrow c \rightarrow d \rightarrow a = 21$$

$$b \rightarrow d \rightarrow c \rightarrow a = 17$$

Here min is 17, take 17.

For node “c” we have:

$$c \rightarrow b \rightarrow d \rightarrow a = 22$$

$$c \rightarrow d \rightarrow b \rightarrow a = 23$$

Here min is 22. Take 22.

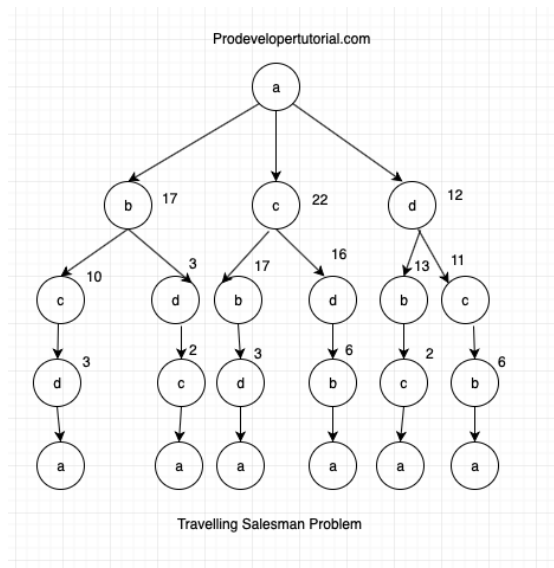
For node “d” we have:

$$d \rightarrow b \rightarrow c \rightarrow a = 23$$

$$d \rightarrow c \rightarrow b \rightarrow a = 12$$

Take 12.

The tree will be:



Now we go one last level up and calculate the final path:

$$a \rightarrow b + (b \rightarrow d \rightarrow c \rightarrow a) = 14 + 17 = 31$$

$$a \rightarrow c + (c \rightarrow b \rightarrow d \rightarrow a) = 9 + 22 = 31$$

$$a \rightarrow d + (d \rightarrow c \rightarrow b \rightarrow a) = 4 + 12 = 16$$

The min value is 16. Hence our result.

Hence if the salesman travels the path $a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$, he can visit all the cities only once and arrive at the initial city with min cost.

Implementation of Travelling Salesman Problem using C++

```
#include <vector>
#include <iostream>
#include <limits.h>
using namespace std;

int tsp(const vector<vector<int>>& cities, int pos, int visited, vector<vector<int>>& state)
{
    if(visited == ((1 << cities.size()) - 1))
```

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

```
return cities[pos][0];

if(state[pos][visited] != INT_MAX)
    return state[pos][visited];

for(int i = 0; i < cities.size(); ++i)
{
    if(i == pos || (visited & (1 << i)))
        continue;

    int distance = cities[pos][i] + tsp(cities, i, visited | (1 << i), state);

    if(distance < state[pos][visited])
        state[pos][visited] = distance;
}
return state[pos][visited];
}

int main()
{
    vector<vector<int>> cities = { { 0, 15, 4, 5 },
                                   { 14, 0, 11, 4 },
                                   { 5, 21, 0, 8 },
                                   { 5, 4, 12, 0 }
                                };
    vector<vector<int>> state(cities.size());

    for(auto& neighbors : state)
        neighbors = vector<int>((1 << cities.size()) - 1, INT_MAX);

    cout << "minimum: " << tsp(cities, 0, 1, state) << endl;
    return 0;
}
```

Output

minimum: 25

Buy the book at : <https://imojo.in/ajGuideAlgoDS>

Aj's guide to Algorithm and Data Structure – Sample

Buy the book at: <https://imojo.in/ajGuideAlgoDS>

Buy the book at : <https://imojo.in/ajGuideAlgoDS>