Let’s assume that you are given n arbitrary characters & you have to create a palindrome using those characters if possible. Propose an efficient solution for this problem & then state its worst-case running time?

The efficient solution for this problem is:-

1)create an empty list "te"

2)using a loop move through the n arbitrary characters.

3) For every character,remove it if the list "te" contains that character else add to the list.

4) if n is even then list size is 0. Or list size is 1

5) by above 2 conditions we can say it can form a palindrome or not.

Code:-

def check(s):

# Create empty list te

te = []

""" For every character in s

remove it if list "te" has

else add it to list "te" """

for i in range(len(s)):

if (s[i] in te):

te.remove(s[i])

else:

te.append(s[i])

# if character length is even list te is expected to be empty

# or if character length is odd " te" size is expected to be 1

if (len(s) % 2 == 0 and len(te) == 0 or

(len(s) % 2 == 1 and len(te) == 1)):

return True

else:

return False

n=int(input())

s=input()

if (check(s)):

print("Palindrome")

else:

print("Not a palindrome")

3. Suppose you have a dataset in which you will get n positive integers containing a huge amount of duplicates & you have to sort them. However, you won’t get more memory to sort these numbers. These numbers are decimal & they are not more than 8 digits in length. Which algorithm do you think will efficiently sort these numbers and why?

4. 4.A Bangladeshi tender company named BTC won a tender that requires it to rebuild only those roads which ensures communication from one location to another. There are multiple ways to reach from one location to another and not all the roads dictate the shortest path. As BTC is a profitable company it wants to cleverly select those roads which will guarantee them maximum profit. If there are n locations and r roads then suggest the most efficient algorithm that will help them reach their goal. Why did you suggest that algorithm? \*

1. .Suppose you have a kid brother and you have to play a game of alphabet with him. You two are given a bunch of alphabets, say n alphabets ranging from a-z and A-Z. Now suggest the most efficient sorting algorithm that can help you sort all the alphabets. How will you use the algorithm? Sorting criteria [A-Z and then a-z] \*
2. Suppose you have ten thousand four-digit numbers which also contains negative numbers. If the minimum number is added to all the numbers then we can use the counting sort. \*
3. ● True
4. ● False
5. The following array is a max heap [14, 9, 13, 7, 8, 10, 15, 4, 6, 3, 2] \*
6. ● True
7. ● False
8. In a connected weighted graph, the edge with maximum weight is never in the minimum spanning tree. \*
9. ● True
10. ● False
11. If we use insertion sort instead of counting sort in radix sort it will work. \*
12. ● True
13. ● False

6. Suppose you are given a graph in which you will find positive & negative cycles crossing all the vertices. We all know that it won’t be possible to perform the topological sort on that graph in this situation. If all the negative cycles are removed & all the vertices touch only the positive cycles then will it be possible to perform topological sort now? If it is possible then explain how & if it is not then explain why not?

Given the situation, in which we remove the negative cycles and all the vertices touch only the positive cycles, then the topological sort can be explained.

We can define a *topological ordering* of a directed graph in such a way that,

For example, we have an edge from x to y, the ordering happens that x is visited before y.

We can find all orders in such a manner that the predecessor of vertex x is processed before we process x.

Eliminating negative, we have run the case that we can find DAG, which means topological ordering can be found.

Pseudocode for topological ordering goes this way:

list LIST = empty

while (Graph G is not empty)

find a vertex v with no incoming edges

delete v from G

add v to LIST

Topological sorting is a linear ordering of vertices V such that (u,v) belong to E means that u must come before v in ordering.

In case of a cycle it is not possible to apply topological sort in any graph with cycles because a topology sort is not possible when there exists a graph with a cycle as the basic condition of finding a topological ordering is that the elements must be arranged in such a way that if a comes before b then b must come only after a but this is violated in case if there is a cycle as then there exists a route from b back to a.

An edge (a,b) defines that there exists a path from a to b and thus topological ordering is that a preceded b and thus if there is a path from b to a then this ordering is violated which occurs in case of a cycle.

7. Let’s think of a directed graph in which the source is connected to the rest of the vertices using a single edge & its weight is negative which is demonstrated here. There are a couple of algorithms for solving the single source path problem. From them, two significant algorithms are Dijkstra’s & Bellman Ford algorithms. Which one of them will efficiently solve the shortest path problem for the mentioned graph.

**Dijkstra Algorithm :**

Dijkstra shortest path algorithm finds the shortest path from a source node to all the other nodes present in the graph. It uses greedy method to calculate the shortest path.

It uses a priority queue to store the distance of each node from the source. At each iteration, it stores the distance of the minimum weight edge that is present in the priority queue and mark it visited.

Once a vertex is mark visited it cannot be revisited again. So for the next time, the unvisited vertices are compared only.

The time complexity of this algorithm is O(E log V) where E is the edges and V are the vertices.

**Bellman Ford Algorithm:**

.The algorithm also finds the shortest path from a source node to all the other nodes present in the graph. However, it is slower than the Dijkstra algorithm because In this algorithm we iterate over all the edges more than once and check if there exists one more shorted path from the source.

If we are able to find the shorter path then we will update the distance in the distance table. In this manner of each of the vertices, we iterate for n times where n is the total number of vertices present in the graph. That makes the bellman ford algorithm slower.

The time complexity of this algorithm is O(E V) where E is the edges and V are the vertices.

Even if the Dijkstra algorithm is faster it does not work if there exists any negative weight cycle present in the graph.It will provide incorrect results.

**But for the given graph it does not contain any negative edge weight cycle there is only one negative edge which does not form any cycle so the Dijkstra algorithm will work perfectly on this graph.**

1. Suppose you have cache/internal memory of 5GB & External memory of 10 GB. You have given a dataset of 5GB. You have to reduce time complexity as much as possible. Space complexity Is not ignorable. From quick sort & merge sort which algorithm will be efficient to sort this dataset?

For this type of dataset, the algorithm more efficient will be **Merge Sort**

**Reasons:**

* Merge Sort works better than quick sort in case of large dataset, the dataset in this case being 5GB.Usage of merge sort is justified.
* The worst case time complexity of quick sort is O(n\*n) but for merge sort it is O(n log n)
* Type of algorithm, Merge Sort is not in place whereas Quick Sort is in place. So while using Merge Sort if we require extra space even, we have it in form of auxilliary memory.
* Merge Sort requires extra space and for this case we have it. So efficient one will be merge sort

**Dynamic programming**, like the divide-and-conquer method, solves problems by combining the solutions to subproblems. (“Programming” in this context refers to a tabular method, not to writing computer code.) Dynamic Programming can

be assumed to be careful brute force.

Dynamic programming solves it’s subproblems exactly once and reuses them for further calculations thus making it more efficient.

Dynamic Programming = Recursion + Memoization

As we saw divide-and-conquer algorithms partition the problem into disjoint subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem. In contrast, dynamic programming applies when the subproblems overlap—that is, when subproblems share subsubproblems. In this context, a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems.

A dynamic-programming algorithm solves each subsubproblem just once and then

saves its answer in a table, thereby avoiding the work of recomputing the

answer every time it solves each subsubproblem.

**Greedy:** Build up a solution incrementally, myopically optimizing some local criterion.

**Divide-and-Conquer:** Break up a problem into sub-problems, solve each sub-problem independently, and combine solution to sub-problems to form

solution to original problem.

**Dynamic Programming:** Break up a problem into a series of overlapping sub-problems, and build up solutions to larger and larger sub-problems.

**Normal Recursive Fibonacci**: O(2^n/2).

**Recursive Fibonacci with Memoization:** T(n) = Number of unique subproblems \* cost per subproblem, O(n).

**Memoized Recursive Longest Common Subsequence:**

In the memoized version, we store results in a matrix so that any given set of arguments to LCS only produces new work (new recursive calls) once. The memoized version begins by initializing arr[i][j] to unknown for all i,j, and then proceeds. In this memoized version, our running time is now just O(mn).

[Dijkstra’s algorithm](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/) is one of the most popular algorithms for solving many single-source shortest path problems having non-negative edge weight in the graphs i.e., it is to find the shortest distance between two vertices on a graph. It was conceived by computer scientist **Edsger W. Dijkstra** in 1956 and published three years later.

**Digital Mapping Services in Google Maps:** Many times we have tried to find the distance in G-Maps, from one city to another, or from your location to the nearest desired location. There encounters the Shortest Path Algorithm, as there are various routes/paths connecting them but it has to show the minimum distance, so Dijkstra’s Algorithm is used to find the minimum distance between two locations along the path. Consider India as a graph and represent a city/place with a vertex and the route between two cities/places as an edge, then by using this algorithm, the shortest routes between any two cities/places or from one city/place to another city/place can be calculated.

**Social Networking Applications:** In many applications you might have seen the app suggests the list of friends that a particular user may know. How do you think many social media companies implement this feature efficiently, especially when the system has over a billion users. The standard Dijkstra algorithm can be applied using the shortest path between users measured through handshakes or connections among them. When the social networking graph is very small, it uses standard Dijkstra’s algorithm along with some other features to find the shortest paths, and however, when the graph is becoming bigger and bigger, the standard algorithm takes a few several seconds to count and alternate advanced algorithms are used.

**Telephone Network:** As we know, in a telephone network, each line has a bandwidth, ‘b’. The bandwidth of the transmission line is the highest frequency that that line can support. Generally, if the frequency of the signal is higher in a certain line, the signal is reduced by that line. Bandwidth represents the amount of information that can be transmitted by the line. If we imagine a city to be a graph, the vertices represent the switching stations, and the edges represent the transmission lines and the weight of the edges represents ‘b’. So as you can see it can fall into the category of shortest distance problem, for which the Dijkstra is can be used.

**IP routing to find Open shortest Path First:** Open Shortest Path First (OSPF) is a link-state routing protocol that is used to find the best path between the source and the destination router using its own Shortest Path First. Dijkstra’s algorithm is widely used in the routing protocols required by the routers to update their forwarding table. The algorithm provides the shortest cost path from the source router to other routers in the network.

**Flighting Agenda:** For example, If a person needs software for making an agenda of flights for customers. The agent has access to a database with all airports and flights. Besides the flight number, origin airport, and destination, the flights have departure and arrival time. Specifically, the agent wants to determine the earliest arrival time for the destination given an origin airport and start time. There this algorithm comes into use.

**Designate file server:** To designate a file server in a LAN(local area network), Dijkstra’s algorithm can be used. Consider that an infinite amount of time is required for transmitting files from one computer to another computer. Therefore to minimize the number of “hops” from the file server to every other computer on the network the idea is to use Dijkstra’s algorithm to minimize the shortest path between the networks resulting in the minimum number of hops.

**Robotic Path:** Nowadays, drones and robots have come into existence, some of which are manual, some automated. The drones/robots which are automated and are used to deliver the packages to a specific location or used for a task are loaded with this algorithm module so that when the source and destination is known, the robot/drone moves in the ordered direction by following the shortest path to keep delivering the package in a minimum amount of time.