# **Portions:**

1. Quick sort

2. Merge sort

3. Karatsuba

4. Strassen

5. Prims

6. Kruskals

7. Dijkstra

8. Knapsack DP

9. Warshall

10. Floyd

11. N Queens

## 1.Quick Sort

**Code:**

**Time Complexity:**

Lets T(n) be the time complexity **for best cases**

n = total number of elements

then

T(n) = 2\*T(n/2) + constant\*n

2\*T(n/2) is because we are dividing array into two array of equal size

constant\*n is because we will be traversing elements of array in each level of tree

therefore,

T(n) = 2\*T(n/2) + constant\*n

further we will devide arrai in to array of equalsize so

T(n) = 2\*(2\*T(n/4) + constant\*n/2) + constant\*n == 4\*T(n/4) + 2\*constant\*n

for this we can say that

T(n) = 2^k \* T(n/(2^k)) + k\*constant\*n

then n = 2^k

k = log2(n)

therefore,

T(n) = n \* T(1) + n\*logn = O(n\*log2(n))

lets T(n) ne total time complexity for **worst case**

n = total number of elements

T(n) = T(n-1) + constant\*n

as we are dividing array into two parts one consist of single element and other of n-1

and we will traverse individual array

T(n) = T(n-2) + constant\*(n-1) + constant\*n = T(n-2) + 2\*constant\*n - constant

T(n) = T(n-3) + 3\*constant\*n - 2\*constant - constant

T(n) = T(n-k) + k\*constant\*n - (k-1)\*constant ..... - 2\*constant - constant

T(n) = T(n-k) + k\*constant\*n - constant\*[(k-1) .... + 3 + 2 + 1]

T(n) = T(n-k) + k\*n\*constant - constant\*[k\*(k-1)/2]

put n=k

T(n) = T(0) + constant\*n\*n - constant\*[n\*(n-1)/2]

removing constant terms

T(n) = n\*n - n\*(n-1)/2

T(n) = O(n^2)

## 2. Merge sort

**Code:**

**Time Complexity:**

## 3. Karatsuba

**Code:**

**Time Complexity:**

## 4. Strassen

**Code:**

**Time Complexity:**

## 5. Prims

**Code:**

**Time Complexity:**

## 6. Kruskals

**Code:**

**Time Complexity:**

## 7. Dijkstra

**Code:**

**Time Complexity:**

## 8. Knapsack DP

**Code:**

**Time Complexity:**

It solves problems that display the properties of [***overlapping sub-problems***](https://www.geeksforgeeks.org/overlapping-subproblems-property-in-dynamic-programming-dp-1/) and [***optimal sub-structure***](https://www.geeksforgeeks.org/optimal-substructure-property-in-dynamic-programming-dp-2/) both of which are present in the 0–1 knapsack problem.

Time Complexity**:** **O (N\*W).**  
where ‘N’ is the number of weight elements and ‘W’ is the capacity of the knapsack.

## 9. Warshall

**Code:**

**Time Complexity:**

## 10. Floyd

**Code:**

**Time Complexity:**

## 11. N Queens

**Code:**

**Time Complexity:**